Xebia

CKAD

Certified Kubernetes Application Developer

Xebia

How to access your Kubernetes cluster

We use Instruqt for this training, which is a very cool online learning platform.

This means you can access Kubernetes from your browser!

Please go to https://instruqt.com/ and create your free account if you haven't already.

CKAD

CERTIFIED KUBERNETES APPLICATION DEVELOPER

CKAD Curriculum

Knowledge, Skills and Abilities that a Certified Kubernetes Application Developer can be expected to demonstrate:

- 13% Core Concepts
- 18% Configuration
- 10% Multi-Container Pods
- 18% Observability
- 20% Pod Design
- 13% Services & Networking
- 8% State Persistence

https://github.com/cncf/curriculum/blob/master/CKAD_Curriculum_V1.18.pdf

CKAD Exam

- The online, proctored, performance-based test consists of a set of performance-based items (problems) to be solved in a command line and is expected to take approximately two (2) hours to complete. **You have a free retake.**
- Example questions:

  - Create a file called question1.yml that declares a deployment in the ckadq1 namespace, with six replicas running the nginx:1.7.9 image

  - Make sure that each pod has the label app=nginx

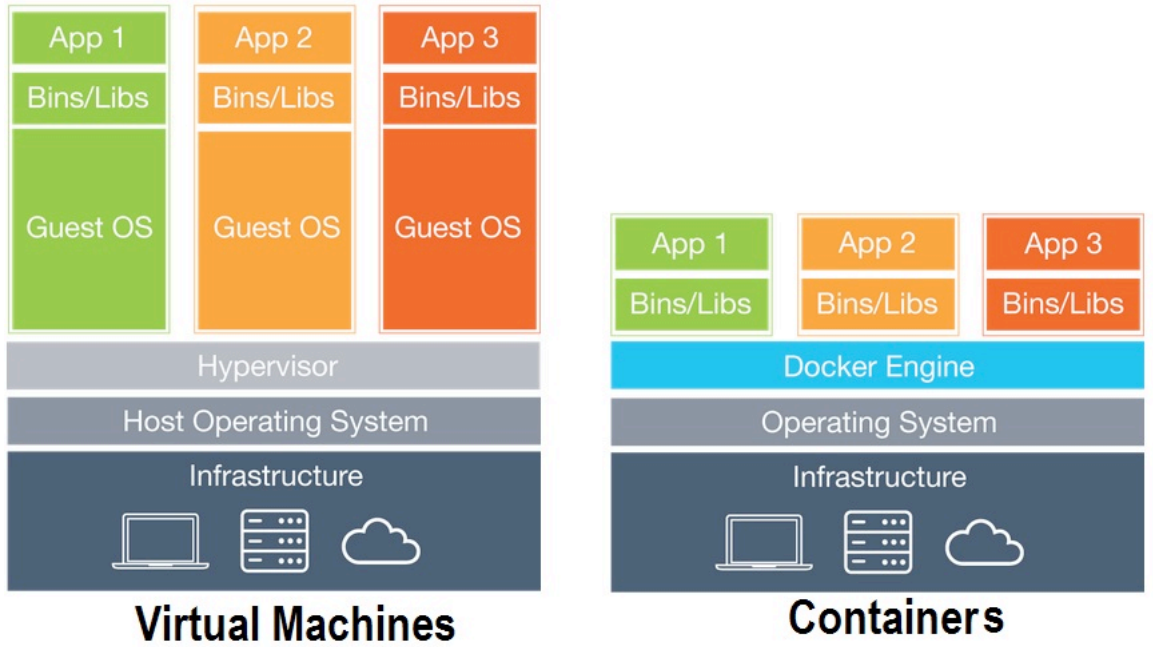
  - The deployment should have the label client=user

  - Configure the deployment so that when the deployment is updated, the existing pods are killed off before new pods are created to replace them

Background and Context

Short recap - The ‘What and Why’ of Containers

- Package your application and its dependencies in a single portable unit
- Containers are small self-contained executable software packages that can be started almost instantly
- Drastically reduce overhead
- More efficient use of resources (better bin packing)



Docker syntax

- The Docker command on the right can be used to run containers on any (Linux) Docker host indefinitely.
- Recurring Docker syntax that can be mapped to k8s:
  - → Environment variables (ConfigMap)
  - → Volumes (volumeMounts)
  - → Port mapping
  - → Images and tags
  - → Entrypoint (command)
  - → Cmd (args)

```
$ docker image pull nginx:1.19
…
$ docker container run                          \
   -d                                           \
   --restart=always                             \
   --name nginx                                 \
   -e "foo=bar"                                 \
   -p "8080:80"                                 \
   -v /tmp:/data                                \
  library/nginx:1.19 nginx -g "daemon off;"
```

What is Kubernetes?

Kubernetes or “K8s” is a Greek word which means “helmsman”.

Main purpose is to **schedule workloads in containers across your infrastructure**. It is a container orchestration system with lots of features that will **keep growing**.

Some of the features are:

- Self-healing
- Autoscaling
- Naming and Service Discovery
- Resource monitoring and logging
- Load balancing
- Rolling update or rollback
- ...

What Kubernetes is not

Kubernetes is not a platform as a service (PaaS)

- Does not build or deploy your source code. Use your existing CI/CD solutions for that.
- Not a dictatorship of what kind of **logging**, **monitoring, alerting** or even markup language to use (YAML, JSON)
- Does not come with application workloads-as-a-service such as databases, middleware built-in

Why choose Kubernetes?

- No limits on what kind of workload/application is supported. If it runs in a container, it will run on Kubernetes
- Define how your application runs in a declarative manner
- Manage your applications' lifecycle in a flexible and uniform way; deploy, monitor, scale and upgrade your application
- Orchestrates computing, networking and storage on behalf of user workloads
- Portable across infrastructure providers (GKE, EKS, AKS)
- As pointed out earlier, continuously extended with new pluggable/optional features

Core Concepts and Commands

CERTIFIED KUBERNETES APPLICATION DEVELOPER

Core concepts and Commands

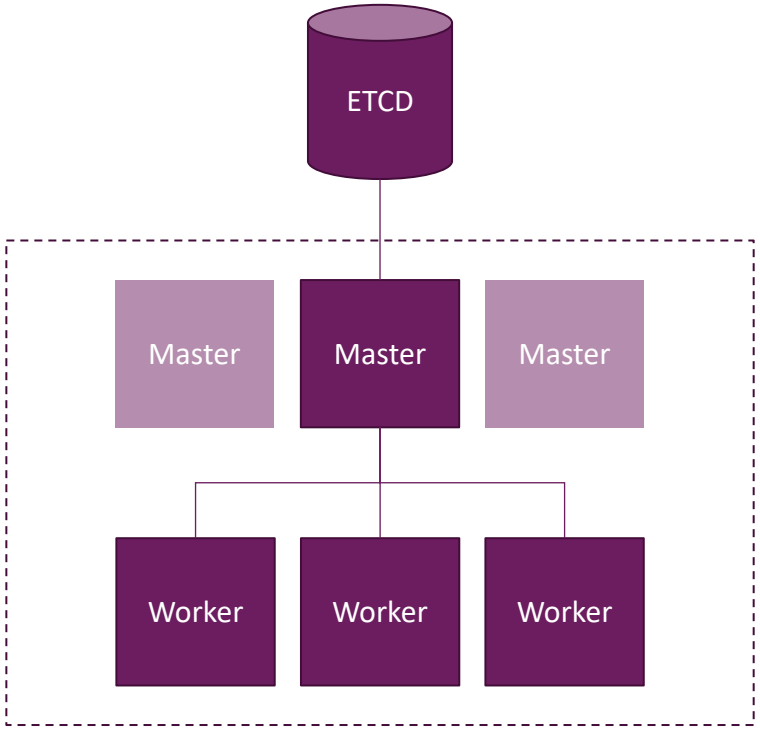
Go to Instruqt and start the Kubernetes cluster

- https://play.instruqt.com/xebia/tracks/ckad-core-concepts-and-commands

What does a typical Kubernetes cluster look like?

- Kubernetes clusters typically consist of two types of nodes:
  a. The **master** node(s) - responsible for running the **control plane**
  b. The **worker** nodes - responsible for running the actual **workload**

- Besides the cluster nodes the state of the cluster is stored in a key-value store named **etcd**



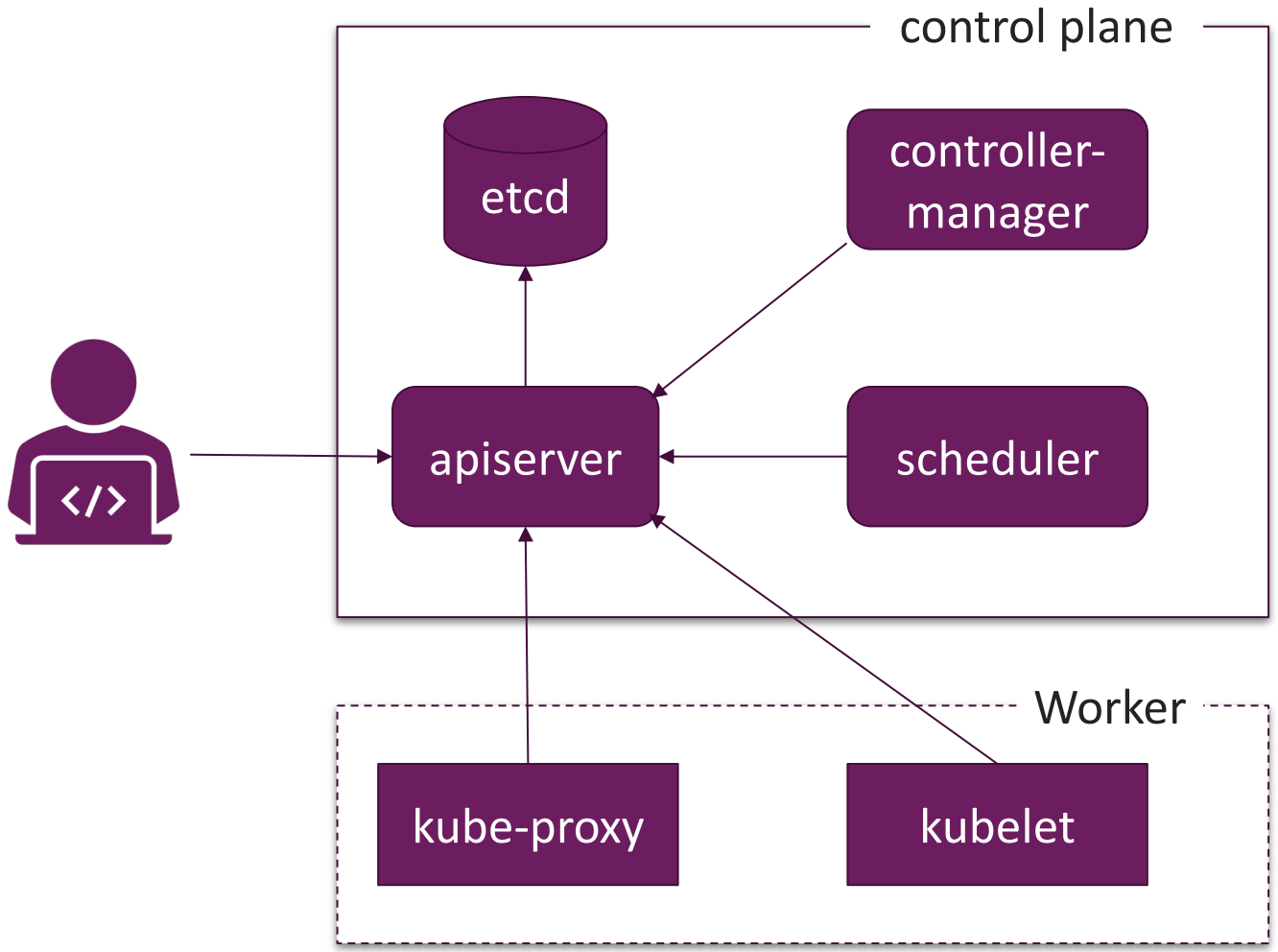
Core components of a Kubernetes cluster

The control plane consists of four components

- **API Server**
- **Controller Manager**
- **Scheduler**
- **etcd**

Each worker in the cluster runs the following agents

- **Kubelet**
- **Kube-proxy**



Kubectl

Interaction with Kubernetes happens via the command line.

Use the **kubectl** command-line tool for this

```
$ kubectl get pods

$ kubectl get services

$ kubectl get nodes
```

Namespaces

Kubernetes supports multiple ‘virtual clusters’ backed by the same physical cluster. These virtual clusters are *namespaces*

- Namespace provide isolation on the API server level

  → Who can read/write resources in a namespace

- To specify the namespaces in kubectl you must use -n per request or set the default namespace in your context

- Resources across all namespaces can be queried with *--all-namespaces*

- E.g. ***kubectl get all --all-namespaces***

```
$ kubectl get pods –n %NS%

$ kubectl config set–context default ––namespace=%NS
```

API Resources and Namespaces

- Most resources are namespaced but not all
  → With: pods, deployments, etc.
  → Without: nodes, namespaces

- Look at the list of API resources

```
$ kubectl create namespace $NAME

$ kubectl get namespace

$ kubectl get namespace $NAME –o yaml

$ kubectl api-resources ––namespaced=true

$ kubectl api-resources ––namespaced=false
```

Kubernetes Pods

- A **pod** is the smallest unit of scheduling in Kubernetes and has these characteristics

  → It consists of one or more containers

  → The containers in a pod shares a network stack

  → The containers in a pod always run on one node

- A **pod** is always defined by a YAML/JSON manifest

```
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest
```

Cluster Commands

- Defining resources is one thing but how do we schedule these?
- Kubectl primitives to manipulate resources
  - → Create (*imperative*)
  - → Replace (*imperative*)
  - → Delete (*imperative*)
  - → **Apply (*declarative*)**
  - → Edit (*declarative*, opens an editor to adjust definition)

```
$ kubectl version
Client Version: version.Info{*}
Server Version: version.Info{*}

$ kubectl create –f pod.yml
pod "nginx" created

$ kubectl delete –f pod.yml
pod "nginx" deleted

$ kubectl replace –f pod.yml
Error from server (NotFound): error when replacing
"pod.yml": pods "nginx" not found

$ kubectl apply –f pod.yml
pod "nginx" created

$ kubectl apply –f pod.yml
pod "nginx" configured
```

Lab: Core concepts

Go to Instruqt and do the ‘Core concepts’ track

https://play.instruqt.com/xebia/tracks/ckad-core-concepts-and-commands

Try to solve all the challenges in this track.

Application deployments

Application deployments

Go to Instruqt and start the Kubernetes cluster

- https://play.instruqt.com/xebia/tracks/ckad-application-deployments

Retrieving Kubernetes Resource Information

- Use the **kubectl get** command to get a list of resources from the cluster
- Use the **kubectl describe** command for explicit information about a resource
- Use the **kubectl explain** for Kubernetes API information

  → This command is important for the exam!

```
$ kubectl get pods
NAME      READY     STATUS    RESTARTS   AGE
nginx     1/1       Running   1          1m

$ kubectl get nodes
NAME    STATUS    ROLES     AGE       VERSION
machine Ready     <none>    1m        v1.10.0
```

Kubectl output formatting

Kubectl has a -o flag to set the output format. A very welcome feature when scripting around kubectl.

Some examples which can be used:

- kubectl get … -o **yaml**
- kubectl get … -o **wide**
- kubectl get … -o **json**
- kubectl get … -o **jsonpath=…**
- kubectl get … -o **go-template=…**
- kubectl get … -o **custom-columns=…**

If you want to parse the output of kubectl in a script you can add --no-headers to disable the column names.

Lab: Output formatting

Go back to Instruqt and try executing the following assignments:

1. Fetch the YAML pod definition of the pod you made previously and store it into a file.

2. Create an overview of all pods, showing on which nodes they run

3. Output a list of all pods and show which image they are running using custom-columns

4. (Bonus) Use the json output and use https://github.com/json-path/JsonPath to help you craft a jsonpath filter with kubectl returning just the image name

1)

```
$ kubectl get pod nginx –o yaml
```

2)

```
$ kubectl get pods –o wide
```

3)

```
$ kubectl get pods –o=custom-
columns=NAME:.metadata.name,IMAGE:.spec.containers
[0].image
```

4)

```
$ kubectl get pods –o
jsonpath={.items.*.spec.containers[0].image}

nginx:1.7.9

$ kubectl get pod nginx –o
jsonpath={.spec.containers[0].image}

nginx:1.7.9
```

Creating and running Pods

- **Kubectl run** runs simple pods from the command line

  → Very similar docker run

- **Kubectl run** is a very powerful command to quickly generate pod manifests.

  → Not really used in production environments (we'll see later why)

  → Used a lot during your exam ☺

```
$ kubectl run --image=nginx nginx
pod/nginx created

$ kubectl run -it --rm --image=busybox busybox -- sh
If you don't see a command prompt, try pressing enter.
/ #


$ kubectl run --image=nginx nginx --dry-run=client --
oyaml > pod.yaml

$ cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

Application deployments

Go to Instruqt and start the Kubernetes cluster

- https://play.instruqt.com/xebia/tracks/ckad-application-deployments

Try to solve all the challenges in this track.

Resource Metadata

Resource Metadata

Go to Instruqt and start the Kubernetes cluster

- https://play.instruqt.com/xebia/tracks/ckad-resource-metadata

Using labels to identify pods

- Key/value pairs that are attached to objects, such as pods
- Labels are intended to be used to specify identifying attributes of objects that are meaningful and relevant to users, but do not directly imply semantics to the core system
- Labels can be used to organize and to select subsets of objects
- **Try the commands on the right**

Kubectl commands

```
# Add a label
$ kubectl label pods nginx env=prod
pod/nginx labeled

# Remove a label
$ kubectl label pods nginx env-
pod/nginx labeled
```

YAML Definition

```
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    mylabel: mytestapp
spec:
  containers:
  - name: nginx
    image: nginx:latest
```

Selecting resources based on labels

- Label selectors can be used to identify a set of objects. There are two types of selectors:
  - → Equality-based label != foo
  - → Set-based label in (foo, bar)
- Combining multiple values will behave as a logical AND
- Not all resources support all selectors
  - → Only equality-based: Service, ReplicationController
  - → Both: Job, Deployment, ReplicaSet, Daemonset and all newer resources

Kubectl commands

```
$ kubectl get pods –l env=prod
```

YAML Definition

```
selector:
  matchLabels:
    component: redis
  matchExpressions:
  – key: tier
    operator: In
    values:
    – cache
  – key: environment
    operator: NotIn
    values:
    – dev
```

Using annotations to add metadata

- You can use either labels or annotations to attach metadata to Kubernetes objects.
- **Labels can be used to select objects** and to find collections of objects that satisfy certain conditions. In contrast, **annotations are not used to identify and select objects**.
- The metadata in an annotation can be small or large, structured or unstructured, and can include characters not permitted by labels.
- Use annotations for non-identifying information. For use-cases outside of k8s internals.
- **Try the commands on the right**

Kubectl commands

```
# Add an annotation
$ kubectl annotate pods nginx env=prod
pod/nginx annotated

# Remove an annotation
$ kubectl anootate pods nginx env-
pod/nginx annotated
```

YAML Definition

```
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: cafe-ingress-with-annotations
  annotations:
    kubernetes.io/description: “Very important"
spec:
  ...
```

Lab: Resource Metadata

Go to Instruqt and start the Kubernetes cluster

- https://play.instruqt.com/xebia/tracks/ckad-resource-metadata

Try to solve the **first five** challenges in this track.

Workload Resources and Security

CERTIFIED KUBERNETES APPLICATION DEVELOPER

Workload Resources and Security

Go to Instruqt and start the Kubernetes cluster

- https://play.instruqt.com/xebia/tracks/ckad-workload-resources-and-security

Resource limits

To specify resource limits, Kubernetes uses the following concepts:

Requests	Limits
• Requests are used by the scheduler to determine which pods can be colocated on a node. • Requests can be exceeded without penalty	• Limits are used to specify the maximum of a resource a pod can use. • Exceeding a limit will result in throttling (CPU) or an OOMkill (Memory)

- **Specifying proper limits and requests on your pods is vital for the scheduler to be able to assign them to nodes effectively**.

- **Improperly set requests and limits can also result in pods hampering performance of other pods on their node.**

Specifying resource limits

Limits are specified on the container level

CPU

- Specified in units of cores
- Fractions are allowed (0.1, 100m)
- Smallest precision is 1m

Memory

- Specified in units of bytes
- You can use suffixes such as M or K but also the power of two equivalents (Mi, Ki)
- Smallest precision is 1 byte

```
---
apiVersion: v1
kind: Pod
metadata:
  name: default-mem-demo
spec:
  containers:
  - name: nginx
    image: nginx:latest
    resources:
      limits:
        memory: 512Mi
        cpu: 1
      requests:
        memory: 256Mi
        cpu: 0.5
```

Quality of Service for pods

Based on the CPU/Memory request, Kubernetes assigns one of three QoS classes to a pod:

- **Guaranteed:** limits are equal to requests
- **Burstable:** request are set and the values are below the values of the limit
- **BestEffort:** no request or limits are set

Note that QoS classes are pod wide so for a pod to be guaranteed the resources of all containers need to match the rules listed above.

The assigned classes can be seen by looking at the qosClass on the pod level.

Resource Quota

- A resource quota can be set per namespace for capacity management.
- Enforcing a limit ensures that workers do not get overloaded by run away containers.
- If you have this in place, pods without resource requests/limits may not be scheduled

```
---
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-demo
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

LimitRange

- A LimitRange is a resource which you can set on a per namespace basis to set defaults for all Pods in that namespace.
- This ensures that new pods without requests will now be able to be scheduled.

```
---
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-limit-range
spec:
  limits:
  - default:
      cpu: 1
      memory: 1024Mi
    defaultRequest:
      cpu: 0.5
      memory: 1024Mi
    type: Container
```

SecurityContext

- A security context defines privilege and access control settings for a Pod or Container. Security context settings include:
  - → Discretionary Access Control
  - → Running as privileged or unprivileged
  - → Linux Capabilities
  - → AllowPrivilegeEscalation
  - → ..and more
- Used to increase security by reducing the blast radius on any security breaches

```
---
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: sec-ctx-vol
    emptyDir: {}
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: sec-ctx-vol
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```

Lab: Workload Resources and Security

Go to Instruqt and start the Kubernetes cluster

- https://play.instruqt.com/xebia/tracks/ckad-workload-resources-and-security

Try to solve the **second** challenge in this track.

Configuration management

Configuration Management

Go to Instruqt and start the Kubernetes cluster

- https://play.instruqt.com/xebia/tracks/ckad-configuration-management

ConfigMaps

- ConfigMaps allow you to decouple configuration artifacts from image content to keep containerized applications portable.
- There are multiple ways to create config maps, on the right there's an example yaml file.
- You can create configmaps directly with kubectl create through commandline arguments

Kubectl commands

```
$ kubectl create configmap my-config    \
  --from-literal=key1=config1           \
  --from-literal=key2=config2


$ kubectl create configmap my-config    \
  --from-file=path/to/bar
```

YAML definition

```
---
kind: ConfigMap
apiVersion: v1
metadata:
  name: redis-config
data:
  REDIS_TEST_ENV: "true"
  redis-config: |
    maxmemory 2mb
    maxmemory-policy allkeys-lru
```

Environment variables

- Besides using ConfigMaps it's also possible to use environment variables to pass parameters
- Both by specifying each key individually using the **env:** block
- Or by using the **envFrom:** block to import an entire ConfigMap

```
---
apiVersion: v1
kind: Pod
metadata:
  name: redis
spec:
  containers:
  - name: test-container
    image: k8s.gcr.io/busybox
    command: [ "/bin/sh", "-c", "env" ]
    envFrom:
    - configMapRef:
        name: redis-config
    env:
    - name: DEMO_FAREWELL
      value: "Such a sweet sorrow"
```

Secrets

- Objects of type secret are intended to hold sensitive information, such as passwords, OAuth secrets, or SSH keys.
- Putting this information in a secret is safer and more flexible than putting it verbatim in a pod definition or in a docker image
- Secrets are not really secrets*. They are merely base64 encoded values which are stored in the ETCD store.

*) As of 1.7 encryption at rest is supported but has to be turned on manually.
https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/


```
$ echo -n 'admin' > ./username.txt

$ echo -n '1f2d1e2e67df' > ./password.txt

$ kubectl create secret generic db-user-pass      \
--from-file=./username.txt                        \
--from-file=./password.txt                        \
secret "db-user-pass" created

$ kubectl get secret db-user-pass -o yaml
apiVersion: v1
kind: Secret
metadata:
  name: db-user-pass
  namespace: default
  resourceVersion: "1137"
  selfLink,creationTimestamp,uid omitted
type: Opaque
data:
  password.txt: MWYyZDFlMmU2N2Rm
  username.txt: YWRtaW4=
type: Opaque
```

Xebia

Updating ConfigMaps and Secrets

When a ConfigMap or secret is updated, what happens depends on how you get the values into the pod:

- In pods that have ConfigMap/Secret values mounted as environment variables, the environment variables are **not** updated (this is a Linux thing)
- It's up to you to restart the pod
- In pods that have ConfigMap/Secret values mounted on the filesystem, the values actually change on the filesystem (this may take a while)
- It's up to your application to pick this up, or up to you to restart the pod

Lab: ConfigMaps and Secrets

Go to Instruqt and do the Configuration Management track

- https://play.instruqt.com/xebia/tracks/ckad-configuration-management

Try to solve the following challenges:

- Create a configmap

- Create a configmap from a file

- Create a configmap which is loaded in a pod

- Create a secret

Pod Management

Pod Management

Go to Instruqt and start the Kubernetes cluster

- https://play.instruqt.com/xebia/tracks/ckad-pod-management

Desired state: declarative vs imperative

- What we want is to be able to specify:

  → Give me **x** copies of image **y** running at all times

  → During a rolling upgrade a max of **10%** may be unavailable

- Instances of pods are ephemeral and should be treated as such

- **Bottomline: do not try to manage individual pods manually, Kubernetes is designed to do this for you!**

Replication Controllers

- A ReplicationController ensures that a specified number of pod replicas are running at any one time. In other words, a ReplicationController makes sure that a pod or a homogeneous set of pods is always up and available.

- This object is meant as a building block and will not support any more features than ensuring that the number of pods matching its selector are equal to the replica count.

ReplicaSet

- Exactly the same as a Replication Controller with one change: it supports the new set-based selector requirements rather than just the equality based ones
- **ReplicaSets are the foundation of all modern Kubernetes deployments**

```
---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
        ports:
        - containerPort: 80
```

Deployments: building on top of ReplicaSets

- When deploying applications on Kubernetes, currently the most feature rich resource is the **Deployment**. It provides declarative updates for Pods and thus ReplicaSets.

- You describe a desired state in a Deployment object, and the Deployment controller changes the actual state to the desired state. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Lab: Pod Management

Go to Instruqt and do the Pod Management track

- https://play.instruqt.com/xebia/tracks/ckad-pod-management

Try to solve the **first three** challenges

Debugging Pod, ReplicaSet or Deployment Failures

- Try debugging from the pod level, first see if the pod is created using:

  → ***kubectl get pods | grep -i $PODNAME***

- If the pod is created but is in CrashLoopBackOff state check:

  → Application logs using: ***kubectl logs $PODNAME***

  → The events of the pod using: ***kubectl describe pod $PODNAME***

- If the pod is not created check the events of the created ReplicaSet:

  → Find the name using ***kubectl get rs | grep -i $DEPLOYMENT_NAME*** and then use describe to find the events

- If there is no ReplicaSet check the events of the Deployment:

  → Use ***kubectl describe deployment $DEPLOYMENT_NAME***

DaemonSet and StatefulSet

DaemonSet

- Used to create a pod for every node in the cluster.
- Lifetime of the pod is tied to the lifetime of the node.

StatefulSet

- Used for pods that require:
  - → Stable, unique network identifiers.
  - → Stable, persistent storage.
  - → Ordered, graceful deployment and scaling.
  - → Ordered, graceful deletion and termination.
  - → Ordered, automated rolling updates.
- Previously known as PetSets
- G.A. since 1.9

State Management

State Management

Go to Instruqt and start the Kubernetes cluster

- https://play.instruqt.com/xebia/tracks/ckad-state-management

Volumes

- When specifying pods it's possible to indicate that you want to mount volumes.
- The Kubelet will take care of mounting these volumes on the node where the pods starts
- Some commonly used ones include:
  - → emptyDir
  - → hostPath
  - → local (similar to hostPath, but smarter)
  - → nfs

See https://kubernetes.io/docs/concepts/storage/volumes/ a complete reference

```
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx:latest
    name: nginx
    volumeMounts:
    - mountPath: /usr/share/nginx/html
      name: content-volume
  volumes:
  - name: content-volume
    hostPath:
      path: /data/html
      type: Directory
```

ConfigMap and Secrets Volumes

- ConfigMaps and Secrets can be mounted as a volume
- Applications inside the pod can access the content of the resource on the mountPath you specify
- In this case `/redis-master/redis.conf` will contain the content of the key `redis-config` defined in the ConfigMap

```
---
apiVersion: v1
kind: Pod
metadata:
  name: redis
spec:
   containers:
   - name: redis
     image: kubernetes/redis:v1
     volumeMounts:
     - mountPath: /redis-master-data
       name: data
     - mountPath: /redis-master
       name: config
   volumes:
     - name: data
       emptyDir: {}
     - name: config
       configMap:
         name: example-redis-config
         items:
         - key: redis-config
           path: redis.conf
```

Persistent Volumes

- A **PersistentVolume** (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using a Storage Class. PVs are implemented as plugins.
- A **PersistentVolumeClaim** (PVC) is a request for storage by a user. It decouples the concrete volumes from the pod.
- A **StorageClass** provides a way for administrators to describe the (dynamic) “classes” of storage they offer. Different classes might map to quality-of-service levels, or to backup policies, or to arbitrary policies determined by the cluster administrators.
- PersistentVolume access modes:

  → ReadWriteOnce – the volume can be mounted as **read-write** by a **single** worker (e.g. SSD disk)

  → ReadOnlyMany – the volume can be mounted **read-only** by **many** workers (e.g. CephFS)

  → ReadWriteMany – the volume can be mounted as **read-write** by **many** workers (e.g. NFS)

PersistentVolume modes

Volume Plugin	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
AWSElasticBlockStore	✓	-	-
AzureFile	✓	✓	✓
AzureDisk	✓	-	-
CephFS	✓	✓	✓
Cinder	✓	-	-
FC	✓	✓	-
Flexvolume	✓	✓	-
Flocker	✓	-	-
GCEPersistentDisk	✓	✓	-
Glusterfs	✓	✓	✓
HostPath	✓	-	-
iSCSI	✓	✓	-
Quobyte	✓	✓	✓
NFS	✓	✓	✓

Static PersistentVolume

```yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: task-pv
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 100Gi
  awsElasticBlockStore:
    volumeID: vol-867g5kii
    fsType: ext4
```

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: task-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
```

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
       claimName: task-pvc
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
```

Dynamic PersistentVolume

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
       claimName: task-pvc
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
```

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: task-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: slow
  resources:
    requests:
      storage: 100Gi
```

```yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "10"
  fsType: ext4
```

```yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: task-pv
spec:
  capacity:
    storage: 100Gi
  accessModes:
    - ReadWriteOnce
  awsElasticBlockStore:
    volumeID: vol-867g5kii
    fsType: ext4
```

Lab: State Management

Go to Instruqt and do the State Management track

- https://play.instruqt.com/xebia/tracks/ckad-state-management

Try to solve the **first** and **third** challenges

Application exposure

Application Exposure

Go to Instruqt and start the Kubernetes cluster

- https://play.instruqt.com/xebia/tracks/ckad-application-exposure

Services

- A Kubernetes Service is an abstraction which defines a logical set of pods and a policy by which to access them.
- The set of pods targeted by a Service is (usually) determined by a Label Selector

Ports are configured by configuring:

→ protocol

→ port

→ targetPort

Kubectl command

```
$ kubectl expose deployment hello-world \
  --type=NodePort                       \
  --name=my-service
```

YAML Definition

```
---
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

Service types

- ClusterIP - Used for services that are only needed **within the cluster**
- NodePort - Used for exposing services on a **static worker port**
- LoadBalancer - Used for exposing the service using an **external load balancer**
- ExternalName - Used to map the service to an external name using a **DNS CNAME**

Discovering services

DNS

```
$ nslookup redis.default.kubernetes.local
Server:         10.0.16.2
Address:        10.0.16.2#53

Non-authoritative answer:
Name:   redis.default.kubernetes.local
Address: 10.0.0.11
```

Environment variables

```
$ env | grep -i REDIS
REDIS_MASTER_SERVICE_HOST=10.0.0.11
REDIS_MASTER_SERVICE_PORT=6379
REDIS_MASTER_PORT=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP_PROTO=tcp
REDIS_MASTER_PORT_6379_TCP_PORT=6379
REDIS_MASTER_PORT_6379_TCP_ADDR=10.0.0.11
```

Special Service Types

Services without selector

- Works the same as a normal services but Endpoints are created manually.
- Allows you to add non Kubernetes resources to a service

Headless services

- Load balancing is disabled
- When resolving the service name all endpoints are returned

```
---
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376

---
kind: Endpoints
apiVersion: v1
metadata:
  name: my-service
subsets:
  - addresses:
      - ip: 1.2.3.4
    ports:
      - port: 9376
```

Lab: Expose a deployment

Go to Instruqt and do the Application Exposure track

- https://play.instruqt.com/xebia/tracks/ckad-application-exposure

Try to solve the **five** challenges

Ingress

- Ingress can provide load balancing, SSL termination and name-based virtual hosting
- Very leaky abstraction but the best we have
- Relies on an Ingress controller deployed by admins to implement. Examples are nginx/haproxy/kong/etc
- Points directly to services and their port
- Annotations are used configure additional functionality that is specific to the used Ingress controller
- Ingress controller can be supplied by cloud provider (GCE)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: test-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        backend:
          serviceName: test
          servicePort: 80
```

Batch processing

Batch Processing

Go to Instruqt and start the Kubernetes cluster

- https://play.instruqt.com/xebia/tracks/ckad-batch-processing

Batch Processing

There are two batch processing resources:

- Job
- CronJob

Batch processing resources are used for jobs that only run for a **finite** amount of time and the pods created for them are disposed afterwards.

Job

Jobs are suited for workloads that need to start, do something and exit.

- A Job manages one or more pods and runs until success.
- If one of the pods managed by the job fails or is deleted then the job will run a new pod until it succeeds

```
apiVersion: batch/v1
kind: Job
metadata:
  name: process-item
spec:
  template:
    spec:
      containers:
      - name: job
        image: busybox
        command: ["sh", "-c", "expr 1 + 2 && sleep 5"]
      restartPolicy: Never
    backoffLimit: 4
```

CronJob (beta)

CronJob creates a Job to start at specific times given a schedule in cron format.

- Ideally design the jobs to be idempotent.
  → Because it can happen that no job or two jobs are created, though rare.
- The example schedules to run the first minute of every hour from 0200 to 1400 UTC on Sunday, Monday, Friday, and Saturday
- ‘concurrencyPolicy’ by default is ‘Allow’ meaning new jobs will be started if old ones are still running. Other values are ‘Forbid’ and ‘Replace’.
- ‘suspend’ to tell the controller to not start new executions, does not apply to already started jobs. I.e., you can suspend an already running cronjob by adding the suspend flag in a new kubectl apply round.

```
# ┌──────────────── minute (0 – 59)
# │ ┌────────────── hour (0 – 23)
# │ │ ┌──────────── day of month (1 – 31)
# │ │ │ ┌────────── month (1 – 12)
# │ │ │ │ ┌──────── day of week (0 – 6)
# │ │ │ │ │
# │ │ │ │ │
# │ │ │ │ │
# * * * * * command to execute
```

```
---
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: my-cronjob
spec:
  schedule: "1 2-14 * * 0-1,5-6"
  concurrencyPolicy: Allow
  suspend: false
  startingDeadlineSeconds:
  jobTemplate: ...
```

Lab: Batch Processing

Go to Instruqt and do the Batch Processing track

- https://play.instruqt.com/xebia/tracks/ckad-batch-processing

Try to solve the **first** challenge

Network Policies

Network Policies

Go to Instruqt and start the Kubernetes cluster

- https://play.instruqt.com/xebia/tracks/ckad-network-policies

NetworkPolicy

Specifies how groups of pods are allowed (whitelisting) to communicate with each other and other network endpoints.

- Network plugin must support them
- NetworkPolicy' resources use labels to select pods and define rules for them
- By default pods are non-isolated, allowing all traffic. Pods become isolated by having a NetworkPolicy select them. Once selected it will reject any connection not allowed by any networpolicy
- Example: Only allow access to pods with label "run=nginx" from pods with label "access=true"

```
---
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: access-nginx
spec:
  podSelector:
    matchLabels:
      run: nginx
  ingress:
  - from:
    - podSelector:
        matchLabels:
          access: "true"
```

Networkpolicies: Traffic direction

- The policyTypes field indicates whether or not the given policy applies to ingress traffic to selected pod, egress traffic from selected pods, or both.
- If no policyTypes are specified on a NetworkPolicy then by default Ingress will always be set and Egress will be set if the NetworkPolicy has any egress rules.

```
---
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: access-nginx
spec:
  podSelector:
    matchLabels:
      run: nginx

  policyTypes:
    - Ingress
    - Egress

  ingress:
  - from:
    - podSelector:
        matchLabels:
          access: "true"
  egress:
    - {}
```

Lab: Network Policies

Go to Instruqt and do the Network Policies

- https://play.instruqt.com/xebia/tracks/ckad-network-policies

Try to solve the **first** challenge

Application Health monitoring

Application Health Monitoring

Go to Instruqt and start the Kubernetes cluster

- https://play.instruqt.com/xebia/tracks/ckad-application-health-monitoring

Container Probes

For Kubernetes to monitor your application health it needs to know what to monitor.

Liveness Probes:	**Readiness Probes:**
• Indicates whether the container is running • Checks if it is still responding to requests from the outside • On failure: Kills the container and restarts it	• Indicates whether the container is ready to receive traffic • Checks for all dependencies and if startup has completed • On failure: Removes the container from the load balancer.

What would you put in the call to these probes?

Configuring Probes

Probes are defined on a per **container** basis

Three types to choose from:

- **exec** *[command]*
- **httpGet** *[path, port, httpHeaders, scheme, host]*
- **tcpSocket** *[port]*

Parameters:

- initialDelaySeconds
- periodSeconds
- timeoutSeconds
- successThreshold
- failureThreshold

```
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx:latest
    name: nginx
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
```

Lab: Application Health Monitoring

Go to Instruqt and do the application health monitoring track

- https://play.instruqt.com/xebia/tracks/ckad-application-health-monitoring

Try to solve the first **two** challenges

Deployment Patterns

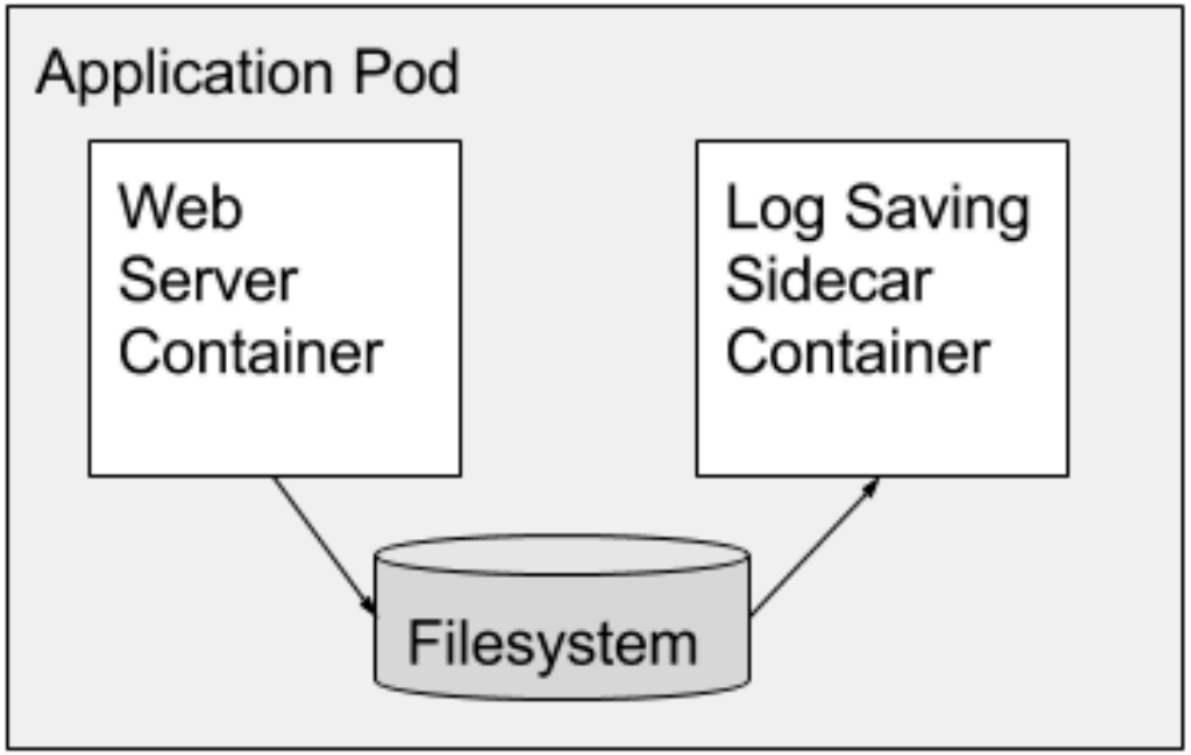
Multi-container Pod Patterns

Pods can hold more containers. And there are some established patterns you can use:

- **Sidecar** container pattern
- **Ambassador** container pattern
- **Adapter** container patterns

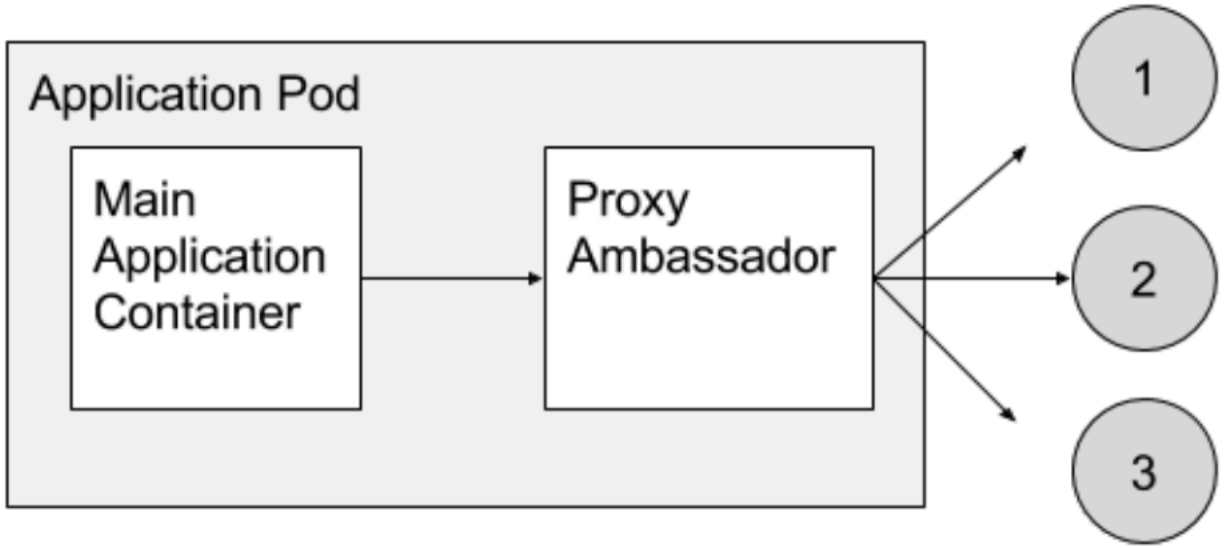
Sidecar Container Pattern

Sidecar containers extend and enhance the “main” container, they take existing containers and make them better.



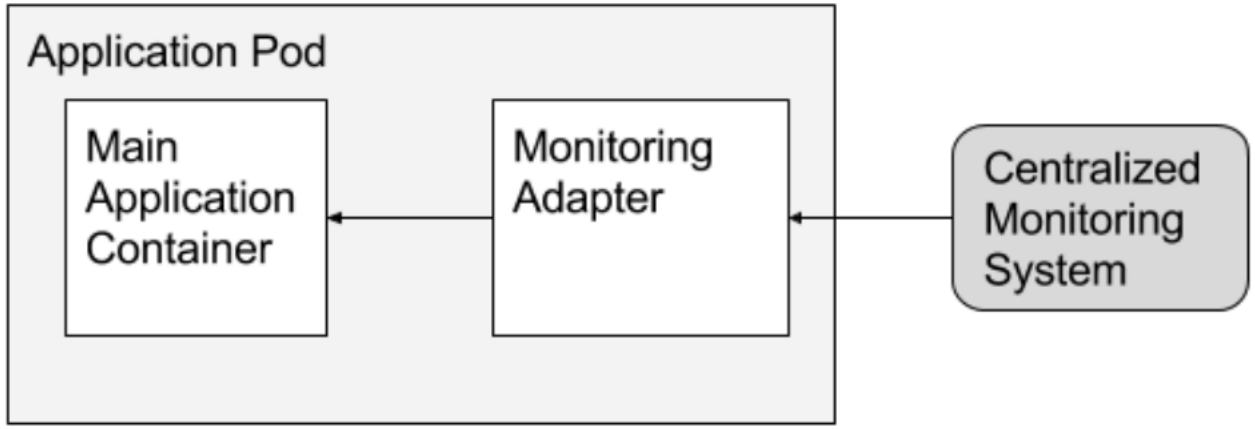
Ambassador Container Pattern

Proxy a local connection to the world



Adapter Container Pattern

Adapter containers standardize and normalize output



Horizontal Pod Autoscaler

- Scales the number of pods according to metrics or custom metrics
- Each 30 seconds (--horizontal-pod-autoscaler-sync-period) the controller manager queries resource utilization against metrics in each HPA
- The HPA will then scale the RC or Deployment

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

Kubernetes API Access

ServiceAccount

- By default every pod will get a **ServiceAccount** token mounted into the pod. The default location is:

  `/var/run/secrets/kubernetes.io/serviceaccount`

- This token can be used to authenticate against the API to fetch information or to execute commands.

- Running kubectl in a pod will automatically pick up this token but you can also manually use it with curl as a bearer token.

```
$
KUBE_TOKEN=$(</var/run/secrets/kubernetes.io/serviceacc
ount/token)

$ curl -sSk -H "Authorization: Bearer $KUBE_TOKEN" \
https://$KUBERNETES_SERVICE_HOST:$KUBERNETES_PORT_443_T
CP_PORT/api/v1/namespaces/default/pods/
```

Closing remarks

CKAD Exam Tips

- Need to know the kubectl commands and YAML syntax
- Terminal knowledge is required
- You have full access to kubernetes.io and kubectl make the most of it!
- Know the layout of the documentation!
- Browse the YAML API: kubectl explain <object>.<subobject> [--recursive]
- Convert resources to YAML: kubectl get <resource> --export=true -o yaml
- Ensure you know what namespaces means and how to use it with kubectl commands
- Don’t spend too much time on one question. Do the questions with weight 2% - 4% later.

Where to go from here?

- **Browse the Kubernetes documentation**
  - → A lot of additional information (skip the cluster config docs)
  - → Learn to navigate the documentation structure
- **Practice, practice, practice!**
  - → You will need to build up muscle memory to get the required speed during the exam
  - → Try experimenting with the different resource types available
  - → Try moving some existing applications onto a cluster


ebia