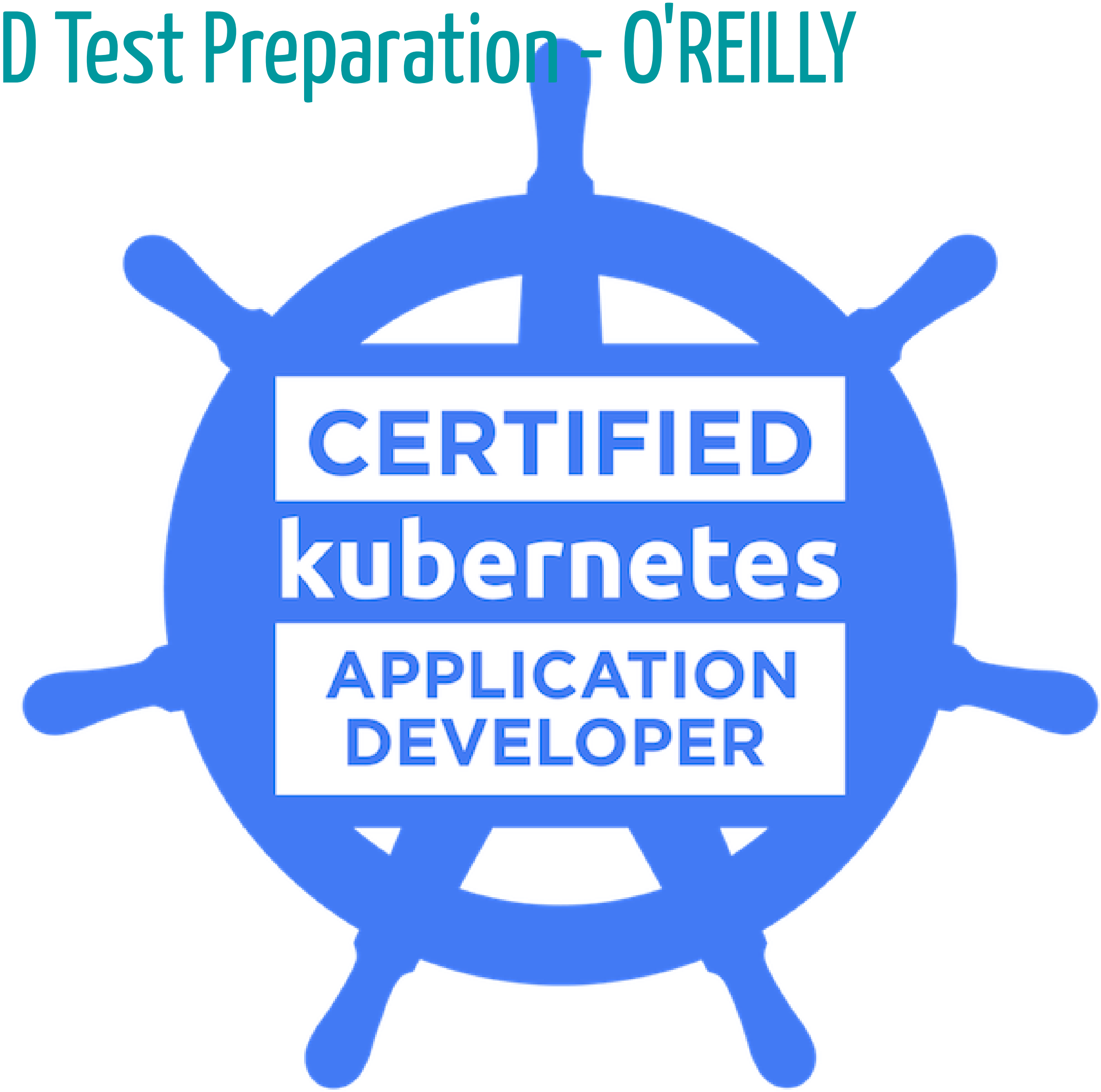
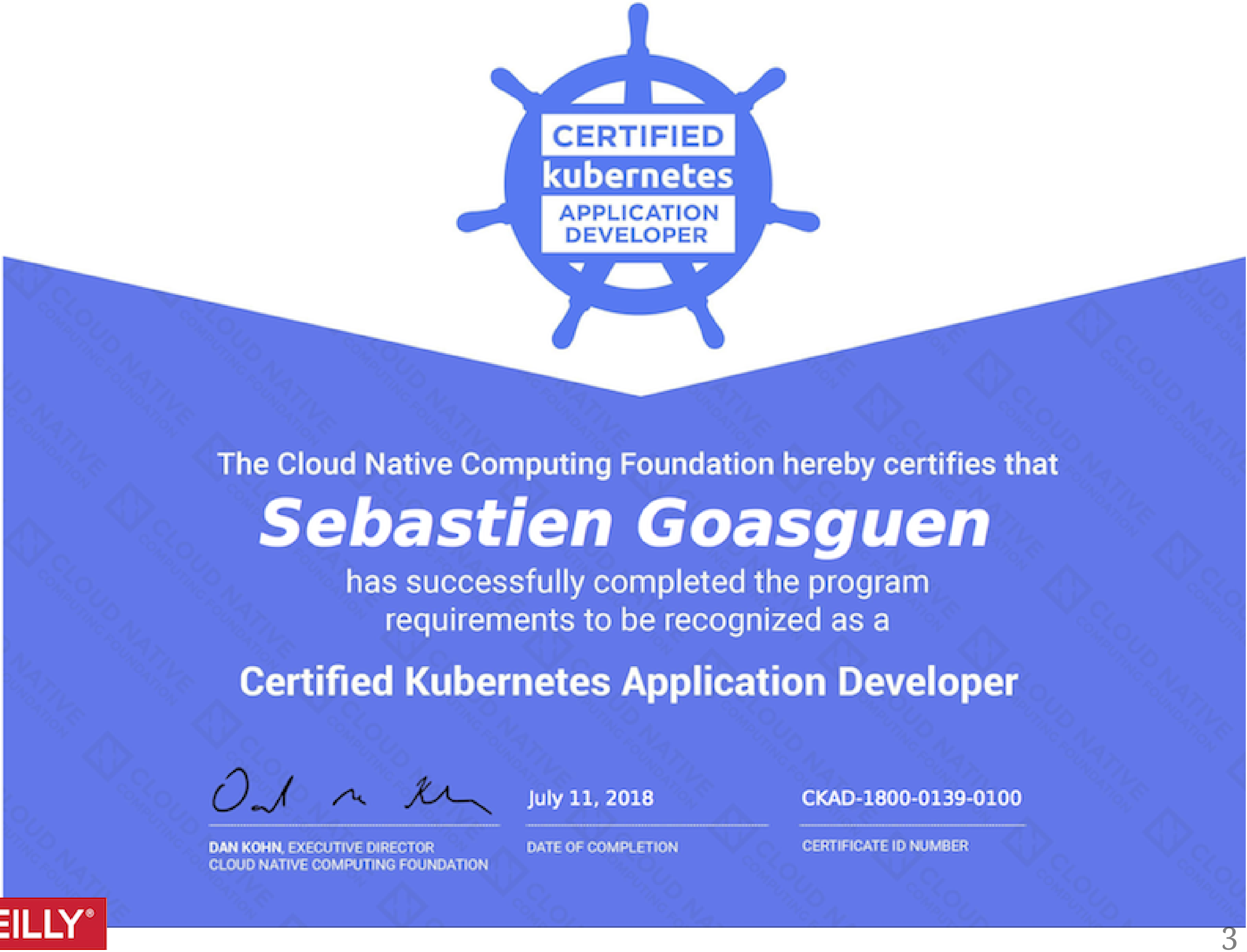
CKAD Test Preparation - O'REILLY



By sebgoa

By Sebastien Goasguen, author of the Docker Cookbook and co-author of Kubernetes cookbook.

@sebgoa [https://github.com/triggermesh) **@triggermesh**



CERTIFIED
kubernetes
APPLICATION
DEVELOPER

The Cloud Native Computing Foundation hereby certifies that

Sebastien Goasguen

has successfully completed the program
requirements to be recognized as a

Certified Kubernetes Application Developer

DAN KOHN, EXECUTIVE DIRECTOR CLOUD NATIVE COMPUTING FOUNDATION	July 11, 2018 DATE OF COMPLETION	CKAD-1800-0139-0100 CERTIFICATE ID NUMBER

Pre-requisities

- minikube , https://github.com/kubernetes/minikube
- or Docker for Desktop (Mac/Windows)
- kubectl , https://kubernetes.io/docs/user-guide/prereqs/
- git

Manifests here:

https://github.com/sebgoa/oreilly-kubernetes

Minikube

Minikube is open source and available on GitHub.

Install the latest release. e.g on OSX:

You will need an "Hypervisor" on your local machine, e.g VirtualBox, KVM, Fusion

```
    $ minikube start
```

Kubernetes Training

Goal: Review of API objects and practice to get ready for CKAD

Questions, questions, questions, questions !!!!!

Agenda

Morning:

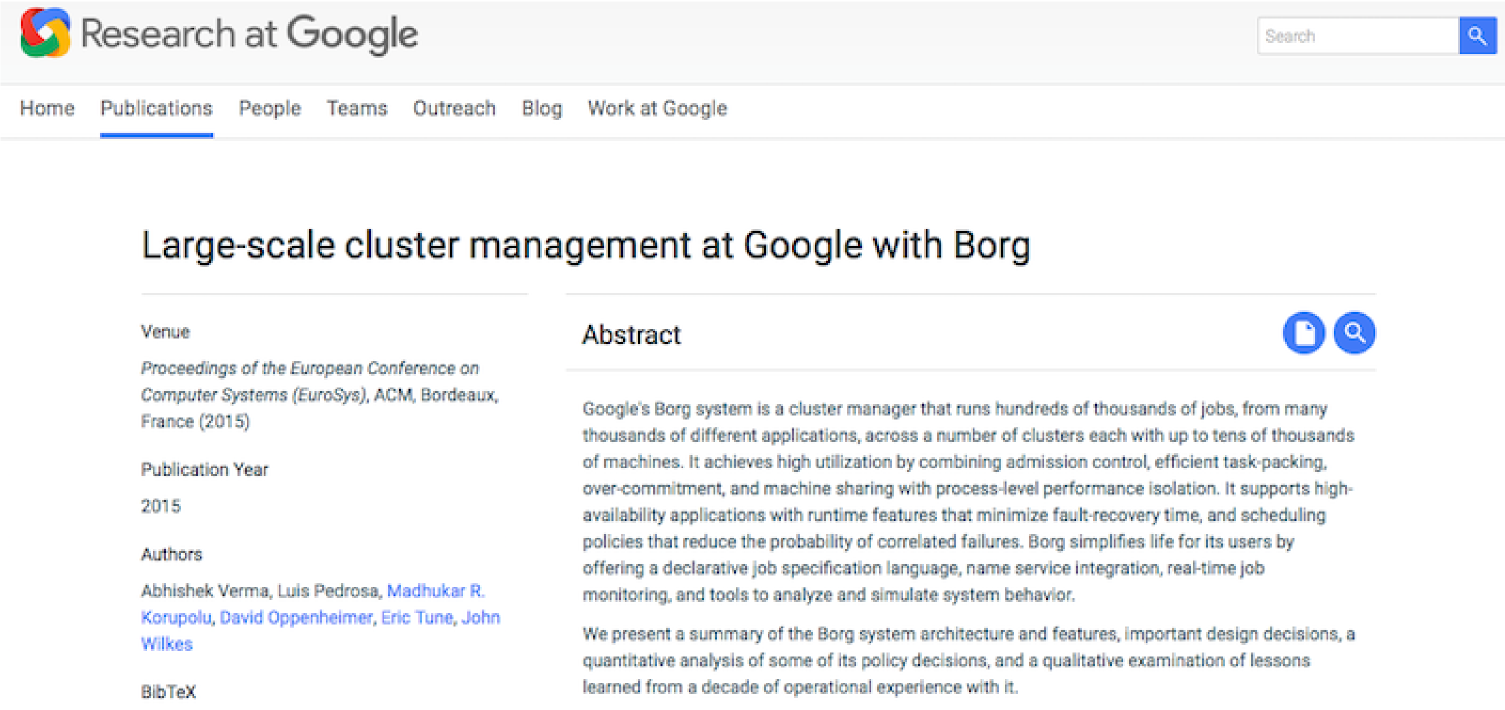
- Review of most common API objects
- Focus on the Pod Specification

Afternoon:

- Practice
- Practice

Borg Heritage

- Borg was a Google secret for a long time.
- Orchestration system to manage all Google applications at scale
- Finally described publicly in 2015
- Paper explains ideas behind Kubernetes



Large-scale cluster management at Google with Borg

Venue

Proceedings of the European Conference on Computer Systems (EuroSys), ACM, Bordeaux, France (2015)

Publication Year

2015

Authors

Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, John Wilkes

BibTeX

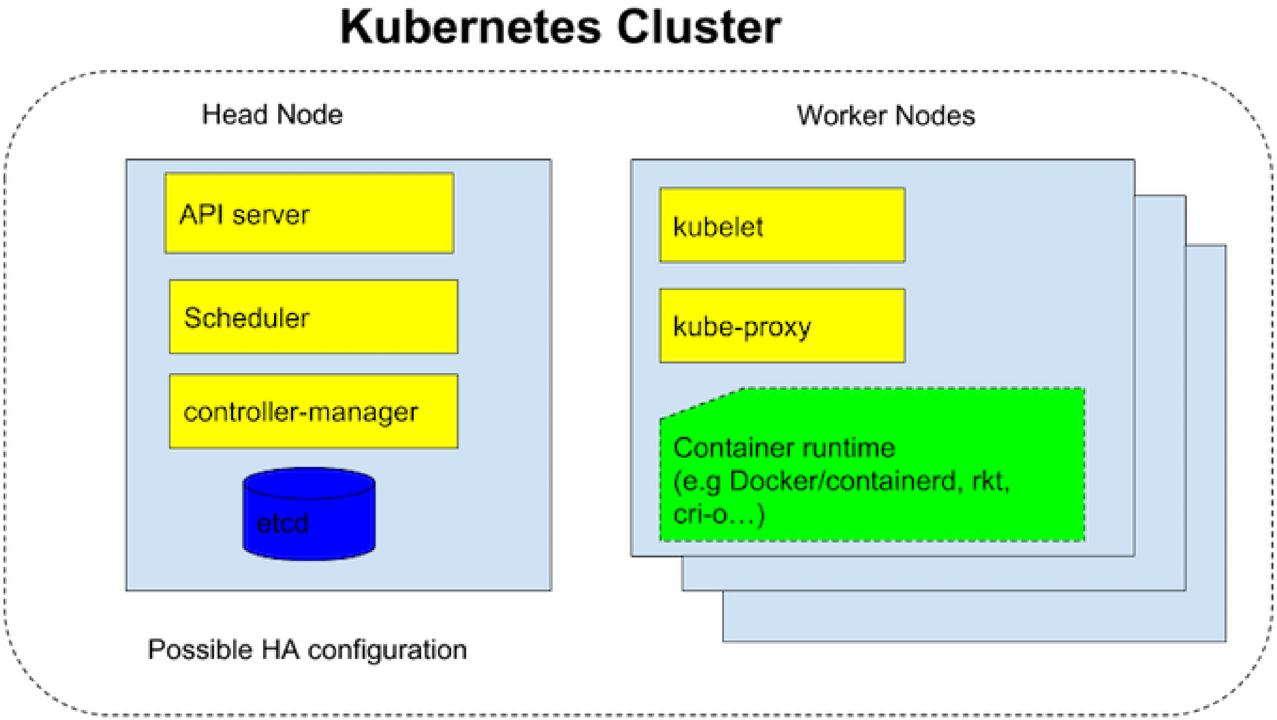
Abstract

Google's Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines. It achieves high utilization by combining admission control, efficient task-packing, over-commitment, and machine sharing with process-level performance isolation. It supports high-availability applications with runtime features that minimize fault-recovery time, and scheduling policies that reduce the probability of correlated failures. Borg simplifies life for its users by offering a declarative job specification language, name service integration, real-time job monitoring, and tools to analyze and simulate system behavior.

We present a summary of the Borg system architecture and features, important design decisions, a quantitative analysis of some of its policy decisions, and a qualitative examination of lessons learned from a decade of operational experience with it.

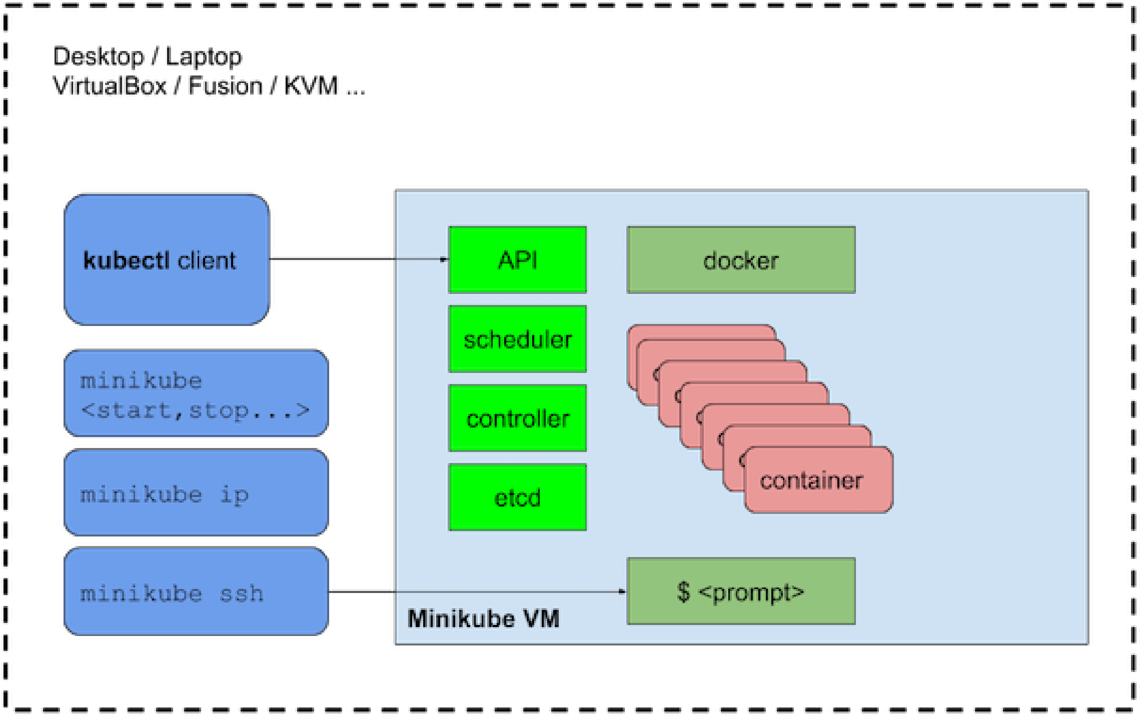
What is it really ?

- A resource manager with lots of HA features
- A scheduler to place containers in a cluster
- Deployed as services on VMs or Bare-metal machines



Minikube

Minikube is open source and available on GitHub.

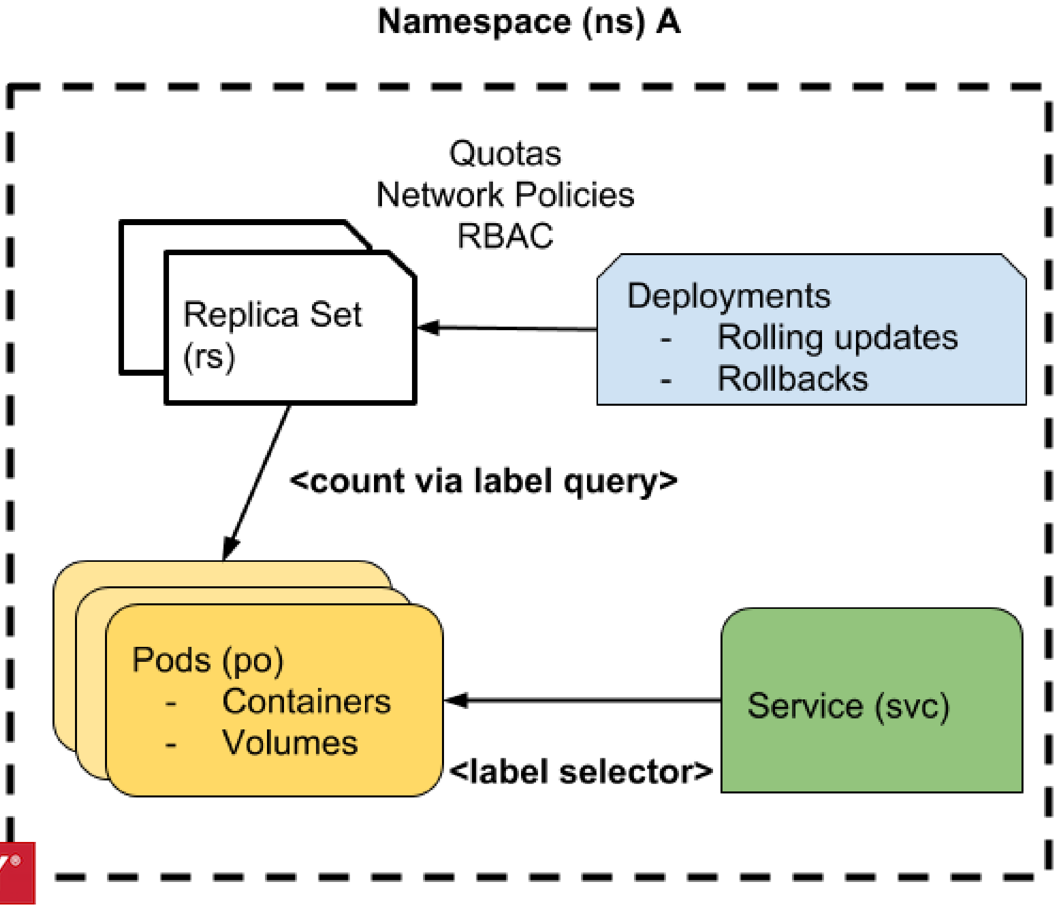


Part I: API Review

- Pods, ReplicaSets, Deployments
- Secrets, ConfigMaps
- `kubectl create` and `kubectl apply`

Core Objects

See "Introduction to Kubernetes course"



Check API Resources with `kubectl`

Check it with `kubectl`:

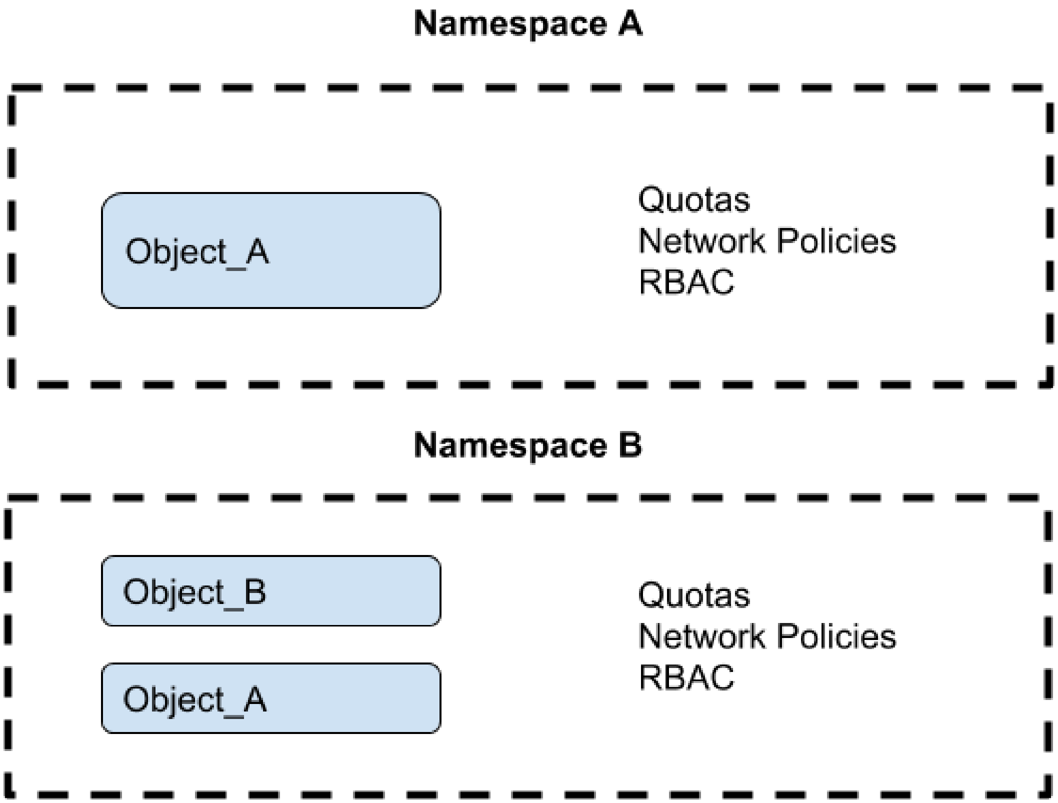
```
$ kubectl get pods
$ kubectl get rc
$ kubectl get ns
```

But there is much more

```
$ kubectl proxy &
$ curl http://127.0.0.1:8001
{
"paths": [
"/api",
"/api/v1",
"/apis",
...
$ curl http://127.0.0.1:8001/api
```

Namespaces

Every request is namespaced e.g `GET https://192.168.99.100:8443/api/v1/namespaces/default/pods`



Labels

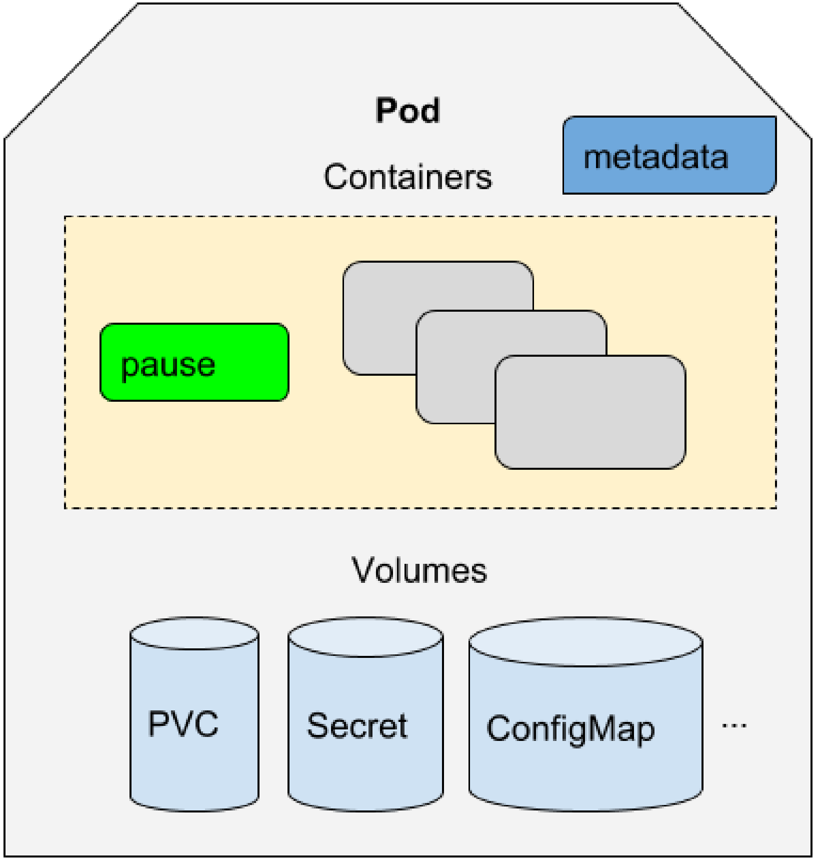
You will have noticed that every resource can contain labels in its metadata. By default creating a deployment with `kubectl run` adds a label to the pods.

```
apiVersion: v1
kind: Pod
metadata:
...
  labels:
    pod-template-hash: "3378155678"
    run: ghost
```

You can then query by label and display labels in new columns:

```
    $ kubectl get pods -l run=ghost
    NAME READY STATUS RESTARTS AGE
    ghost-3378155678-eq5i6 1/1 Running 0 10m
    $ kubectl get pods -Lrun
    NAME READY STATUS RESTARTS AGE RUN
    ghost-3378155678-eq5i6 1/1 Running 0 10m ghost
    nginx-3771699605-4v27e 1/1 Running 1 1h nginx
```

Become Friends with Pods



Kubectl *Pod* commands

```
kubectl logs ...
kubectl describe ...
kubectl explain ...
kubectl exec ...
kubectl label ...
kubectl annotate ...
```

and tricks

```
kubectl get pods ... -o json | jq ..
kubectl run ...--dry-run -o json
kubectl get pods .... --export
```

Powerful REST based API

YAML or JSON definitions for objects

```
$ kubectl --v=9 get pods
...
```

You can `get` every object, as well as `delete` them

Exercise

- Use *curl* to list Pods
- Use *curl* to create a Pod
- Use *curl* to delete a Pod

ResourceQuota Object

Create a *oreilly* ns from a file:

```
apiVersion: v1
kind: Namespace
metadata:
  name: oreilly
```

Then create a *ResourceQuota* to limit the number of Pods

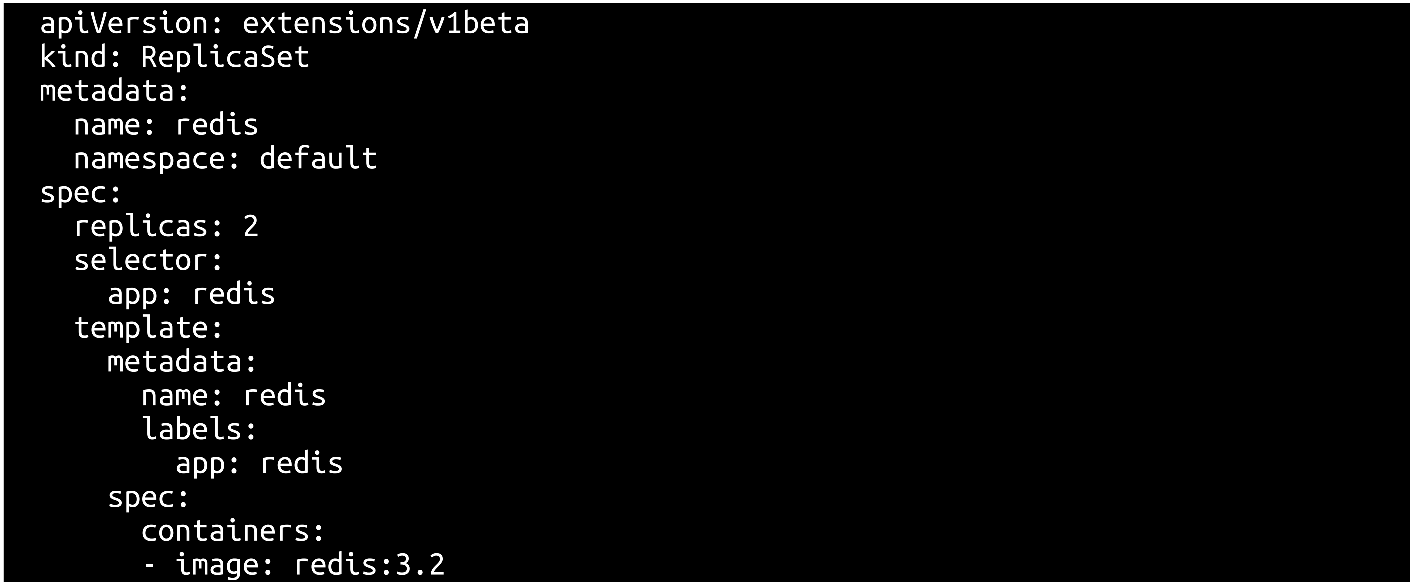
```
$ cat rq.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: object-counts
spec:
  hard:
    pods: "1"
...
$ kubectl create -f rq.yaml --namespace=oreilly
```

Then test !

ReplicaSet Object

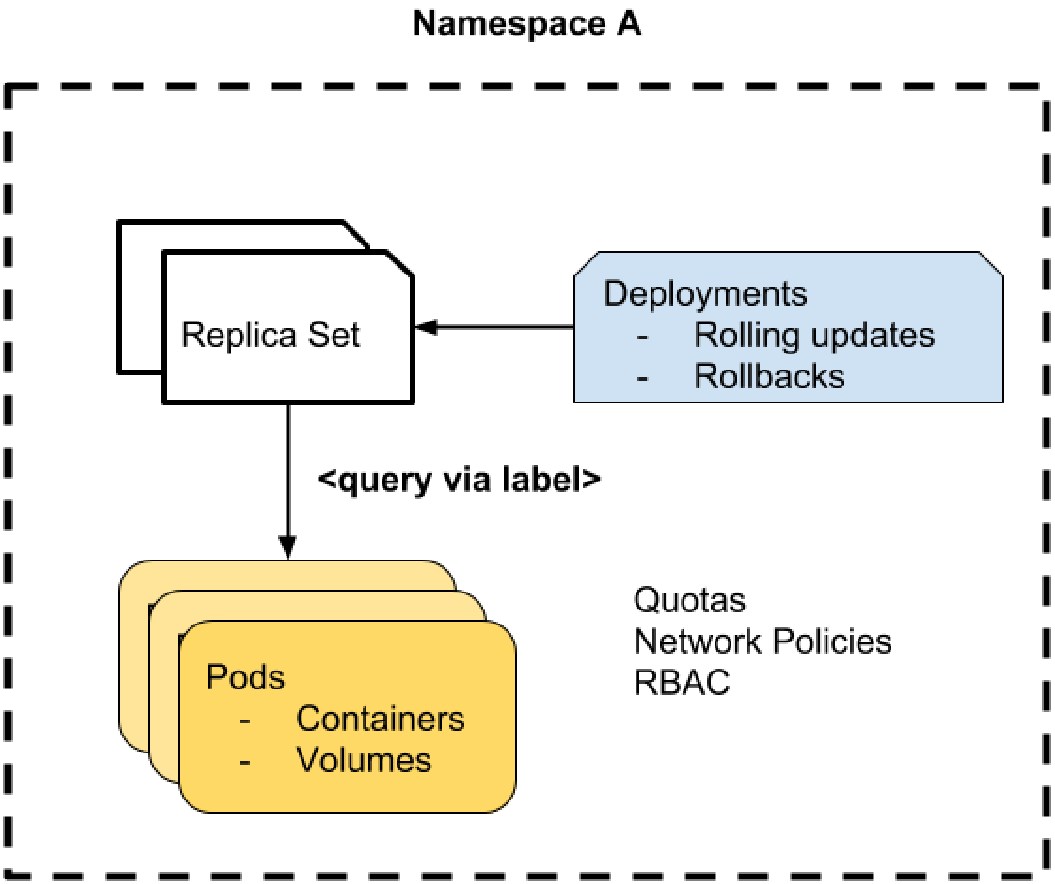
Same as all Objects. Contains *apiVersion*, *kind*, *metadata*

But also a *spec* which sets the number of replicas, and the selector. An RC insures that the matching number of pods is running at all time. The *template* section is a Pod definition.



```
apiVersion: extensions/v1beta
kind: ReplicaSet
metadata:
  name: redis
  namespace: default
spec:
  replicas: 2
  selector:
    app: redis
  template:
    metadata:
      name: redis
      labels:
        app: redis
    spec:
      containers:
      - image: redis:3.2
```

Deployments



Namespace A

Replica Set

Deployments
- Rolling updates
- Rollbacks

<query via label>

Pods
- Containers
- Volumes

Quotas
Network Policies
RBAC

Scaling and Rolling update of Deployments

Just like RC, Deployments can be scaled.

```
$ kubectl scale deployment/nginx --replicas=4
deployment "nginx" scaled
$ kubectl get deployments
NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE
nginx 4 4 4 1 12m
```

What if you want to update all your Pods to a specific image version. *latest* is not a version number...

```
$ kubectl set image deployment/nginx nginx=nginx:1.10 --all
```

What the RS and the Pods.

```
$ kubectl get rs --watch
NAME DESIRED CURRENT AGE
nginx-2529595191 0 0 3m
nginx-3771699605 4 4 46s
```

You can also use `kubectl edit deployment/nginx`

Accessing *Services*

Now that we have a good handle on creating resources, managing and inspecting them with `kubectl`. The elephant in the room is how do you access your applications ?

The answer is Services, another Kubernetes object. Let's try it:

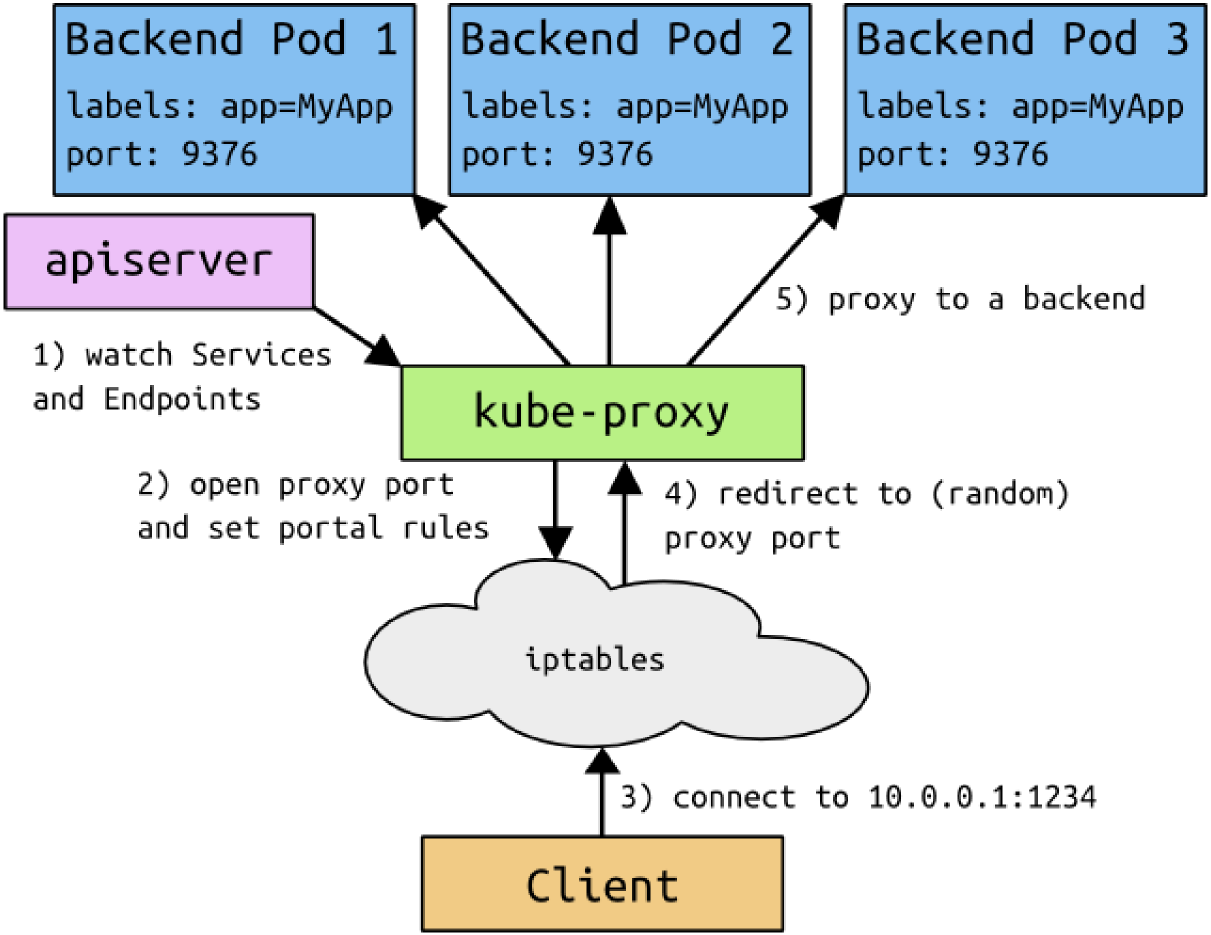
```
  $ kubectl expose deployment/nginx --port=80 --type=NodePort
  $ kubectl get svc
  NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE
  kubernetes 10.0.0.1 <none> 443/TCP 18h
  nginx 10.0.0.112 nodes 80/TCP 5s
  $ kubectl get svc nginx -o yaml
```

```
apiVersion: v1
kind: Service
...
spec:
  clusterIP: 10.0.0.112
  ports:
  - nodePort: 31230
...
```

```
  $ minikube ip
  192.168.99.100
```

Open your browser at `http://192.168.99.100:<nodePort>`

Services Diagram



Service Types

Services can be of three types:

- ClusterIP
- NodePort
- LoadBalancer

LoadBalancer services are currently only implemented on public cloud providers like GKE and AWS. Private cloud solutions also may implement this service type if there is a Cloud provider plugin for them in Kubernetes (e.g CloudStack, OpenStack)

ClusterIP service type is the default and only provides access internally (except if manually creating an external endpoint).

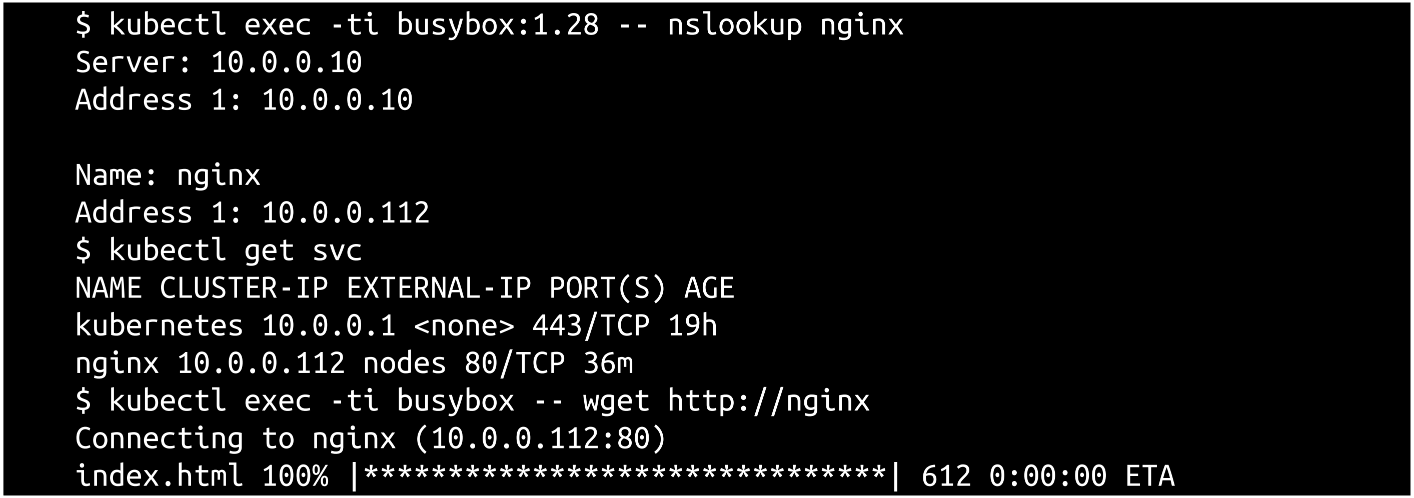
NodePort type is great for debugging, but you need to open your firewall on that port (NodePort range defined in Cluster configuration). Not recommended for public access.

Exercise

- Run `kubectl proxy`
- Open your browser and find the correct URL to access your service

DNS

A DNS service is provided as a Kubernetes add-on in clusters. On GKE and minikube this DNS service is provided by default. A service gets registered in DNS and DNS lookup will further direct traffic to one of the matching Pods via the ClusterIP of the service.



```
$ kubectl exec -ti busybox:1.28 -- nslookup nginx
Server: 10.0.0.10
Address 1: 10.0.0.10

Name: nginx
Address 1: 10.0.0.112
$ kubectl get svc
NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes 10.0.0.1 <none> 443/TCP 19h
nginx 10.0.0.112 nodes 80/TCP 36m
$ kubectl exec -ti busybox -- wget http://nginx
Connecting to nginx (10.0.0.112:80)
index.html 100% |*******************************| 612 0:00:00 ETA
```

Exercise: WordPress

Create a deployment to run a MySQL Pod.

```
$ kubectl run mysql --image=mysql:5.5 --env=MYSQL_ROOT_PASSWORD=root
$ kubectl expose deployments mysql --port 3306
```

And now wordpress:

```
$ kubectl run wordpress --image=wordpress --env=WORDPRESS_DB_HOST=mysql --
env=WORDPRESS_DB_PASSWORD=root
$ kubectl expose deployments wordpress --port 80 --type LoadBalancer
```

BREAK

Part II: Other Objects and a bit more focus on Pods

- DaemonSets
- StatefulSets
- CronJobs
- Jobs
- Ingress
- Persistent Volume Claims
- …

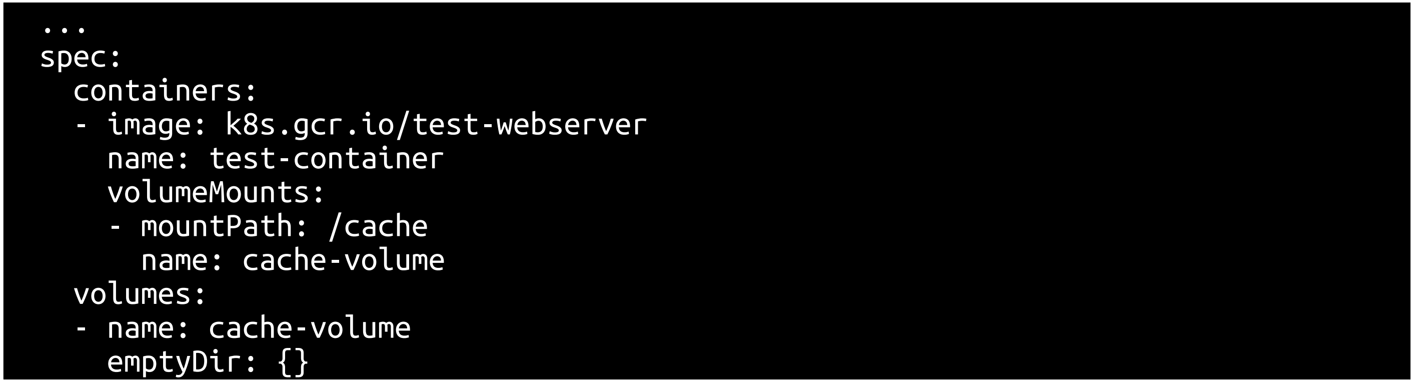
e.g CronJob

A Pod that is run on a schedule

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

Volumes

Define array of volumes in the Pod spec. Define your volume types.



```
...
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
```

Using Secrets

To avoid passing secrets directly in a Pod definition, Kubernetes has an API object called *secrets*. You can create, get, delete secrets. They can be used in Pod templates.

```
    $ kubectl get secrets
    $ kubectl create secret generic --help
    $ kubectl create secret generic mysql --from-literal=password=root
```

ConfigMap

To store a configuration file made of key value pairs, or simply to store a generic file you can use a so-called config map and mount it inside a Pod

```
    $ kubectl create configmap velocity --from-file=index.html
```

The mount looks like this:

```
...
spec:
  containers:
  - image: busybox
...
    volumeMounts:
    - mountPath: /velocity
      name: test
    name: busybox
  volumes:
  - name: test
    configMap:
      name: velocity
```

For persistency use PV and PVC

```
kubectl get pv
kubectl get pvc
```

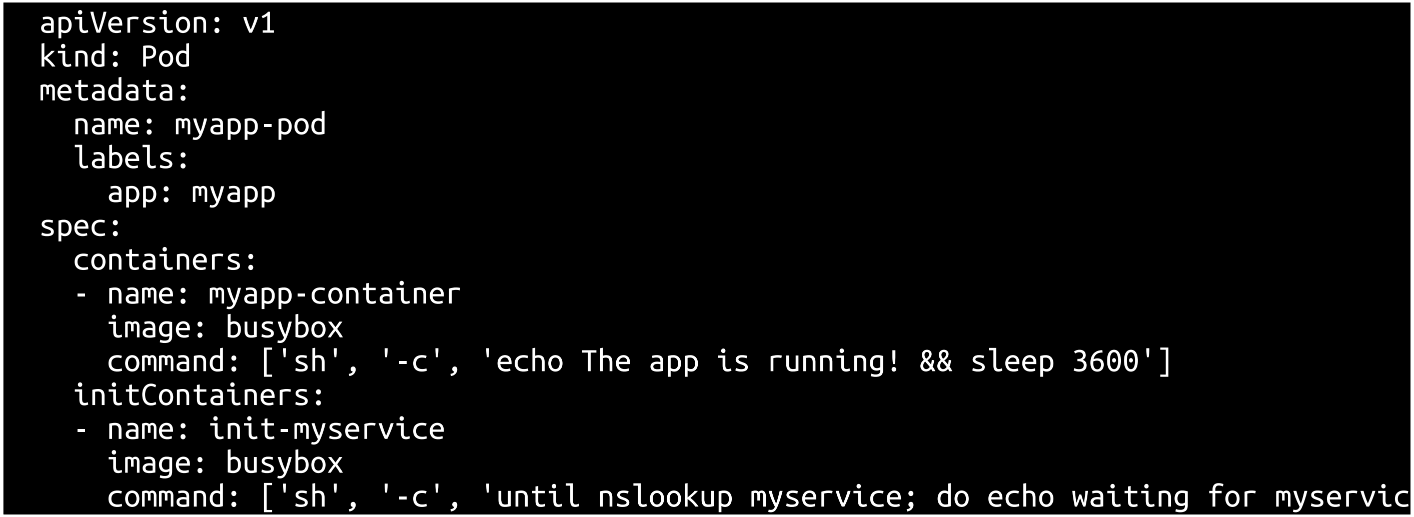
In Minikube dynamic provisioning is setup, you only need to write a volume claim

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi
```

Init-containers

Maybe you want to do some prep work before starting a container. Prep a file system, run some provisioning script…They run to completion and then the app starts.

You can run an *initializing* container:



```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice
    image: busybox
    command: ['sh', '-c', 'until nslookup myservice; do echo waiting for myservic
```

Use Volumes with Init-containers

Example the git initializer

```
apiVersion: v1
kind: Pod
metadata:
  name: git-repo-demo
spec:
  initContainers:
    - name: git-clone
      image: alpine/git # Any image with git will do
      args:
        - clone
        - --single-branch
        - --
        - https://github.com/kubernetes/kubernetes # Your repo
        - /repo
      volumeMounts:
        - name: git-repo
          mountPath: /repo
  containers:
    - name:  busybox
      image: busybox
      args: ['sleep', '100000'] # Do nothing
      volumeMounts:
        - name: git-repo
          mountPath: /repo
  volumes:
    - name: git-repo
      emptyDir: {}
```

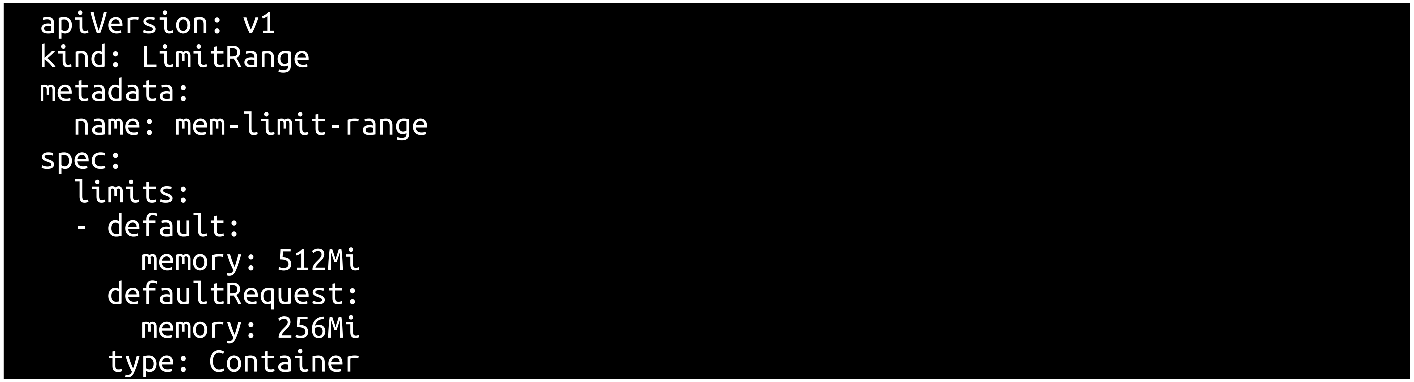
Requests and Limits

Great example to follow in the docs

```
apiVersion: v1
kind: Pod
metadata:
  name: memory-demo
  namespace: mem-example
spec:
  containers:
  - name: memory-demo-ctr
    image: polinux/stress
    resources:
      limits:
        memory: "200Mi"
      requests:
        memory: "100Mi"
    command: ["stress"]
    args: ["--vm", "1", "--vm-bytes", "150M", "--vm-hang", "1"]
```

Default Limit Range in a Namespace

You can create default per namespace which will be automatically added in each Pod manifest.



```
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-limit-range
spec:
  limits:
  - default:
      memory: 512Mi
    defaultRequest:
      memory: 256Mi
    type: Container
```

Similar for CPUs And this goes in pair with quotas

Probes

- Liveness probe to know when to restart a container
- Readiness probe to know when to send traffice to it

Both can be an `exec` and `http` call or a `tcp` socket connection.

```
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
```

Relax isolation

Share the process namespace betwen all containers in a Pod

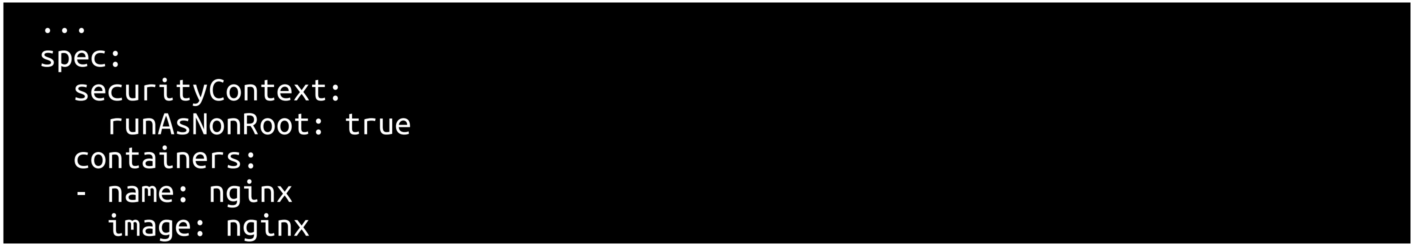
```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  shareProcessNamespace: true
  containers:
  - name: nginx
    image: nginx
  - name: shell
    image: busybox
...
```

Useful for debugging ...

> Container filesystems are visible to other containers in the pod through the /proc/$pid/root link. This makes debugging easier, but it also means that filesystem secrets are protected only by filesystem permissions.

Security Context

Set uid, gid, selinux, fs permissions, capabilities at the Pod or container level.



```
...
spec:
  securityContext:
    runAsNonRoot: true
  containers:
  - name: nginx
    image: nginx
```

ServiceAccount

Pods can talk to the Kubernetes API server using a *service account*

It used to be that this service account had full privileged access to the API server ...:(Now you need to grant it privileges to do anything, see RBAC roles and rolebinding.

A namespace has a default service account. Pods in a namespace will use this service account. Otherwise create a new service account.

```
 kubectl get ns
 kubectl get sa
 kubectl create ns kude
 kubectl get sa -n kude
```

And the Pod spec:

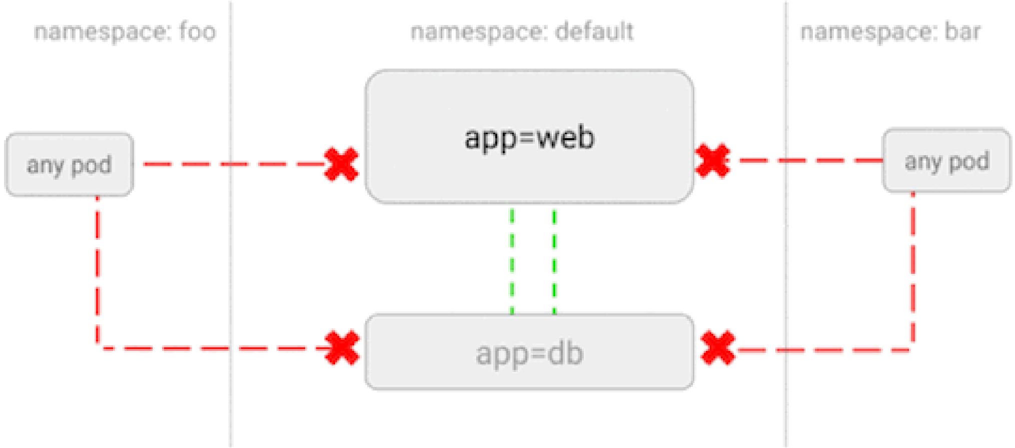
```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  serviceAccountName: kude
```

Network Policies

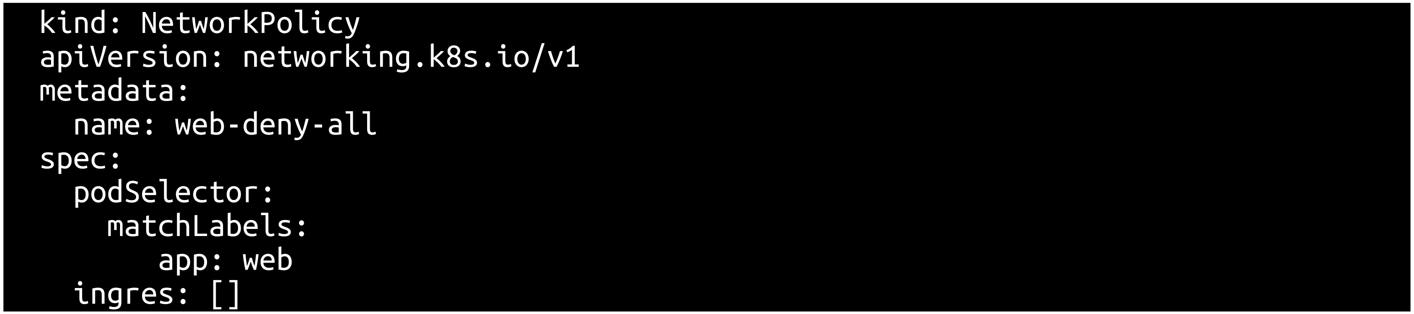
You need a Networking add-on that has a network policy controller.

Check Ahmet's tutorial https://ahmet.im/blog/kubernetes-network-policy/

And his repo https://github.com/ahmetb/kubernetes-networkpolicy-tutorial



Deny All Network Policy



```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-deny-all
spec:
  podSelector:
    matchLabels:
       app: web
  ingres: []
```

Imperative/ Declarative

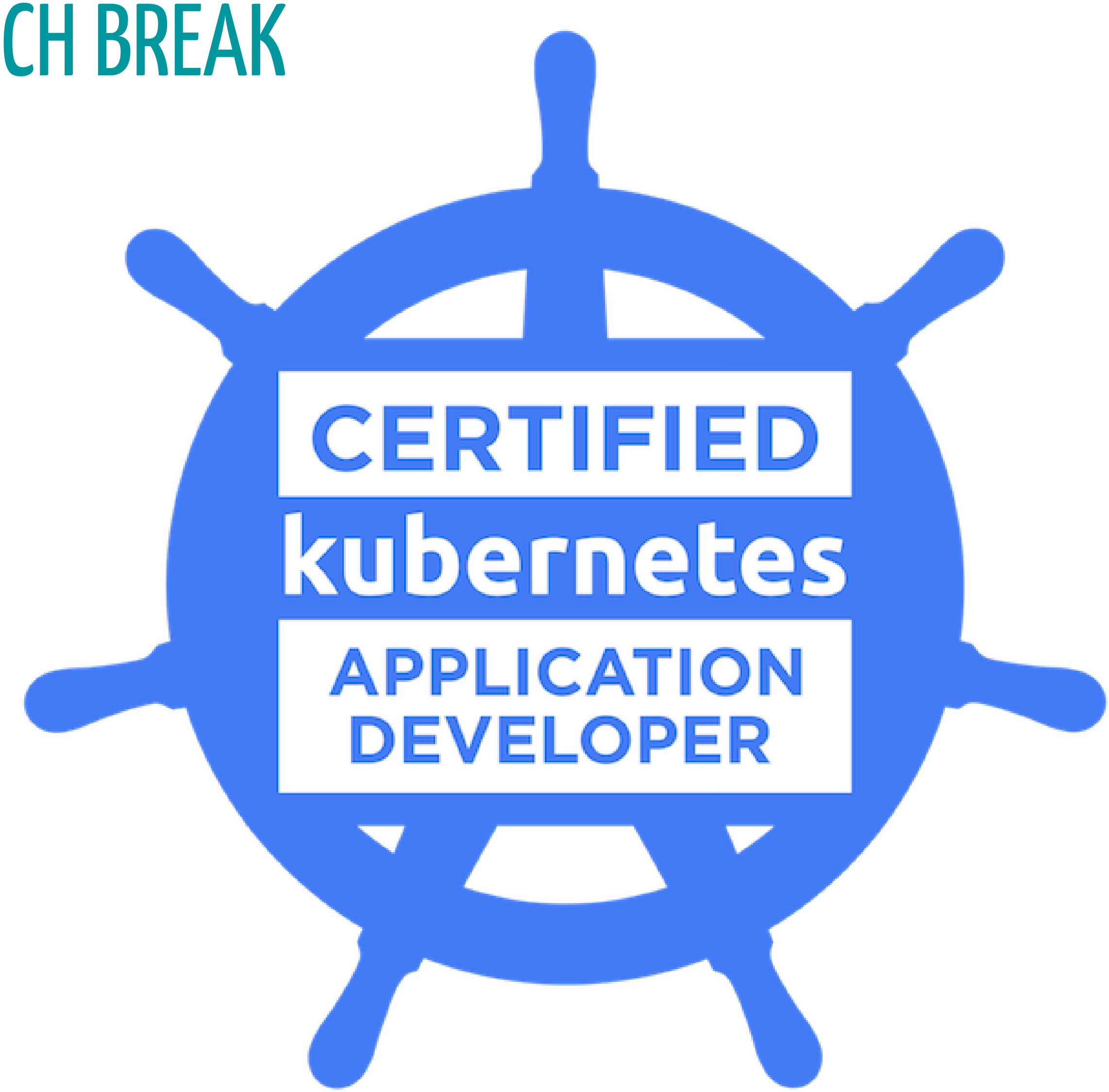
See a blog about it

```
kubectl create ns ghost
kubectl create quota blog --hard=pods=1 -n ghost
kubectl run ghost --image=ghost -n ghost
kubectl expose deployments ghost --port 2368 --type LoadBalancer -n ghost
kubectl run --generator=run-pod/v1 foobar --image=nginx
```

Get the manifests and become more declarative

```
kubectl get deployments ghost --export -n ghost -o yaml
kubectl create service clusterip foobar --tcp=80:80 -o json --dry-run
kubectl replace -f ghost.yaml -n ghost
kubectl apply -f <object>.<yaml,json>
```

LUNCH BREAK



The Exam

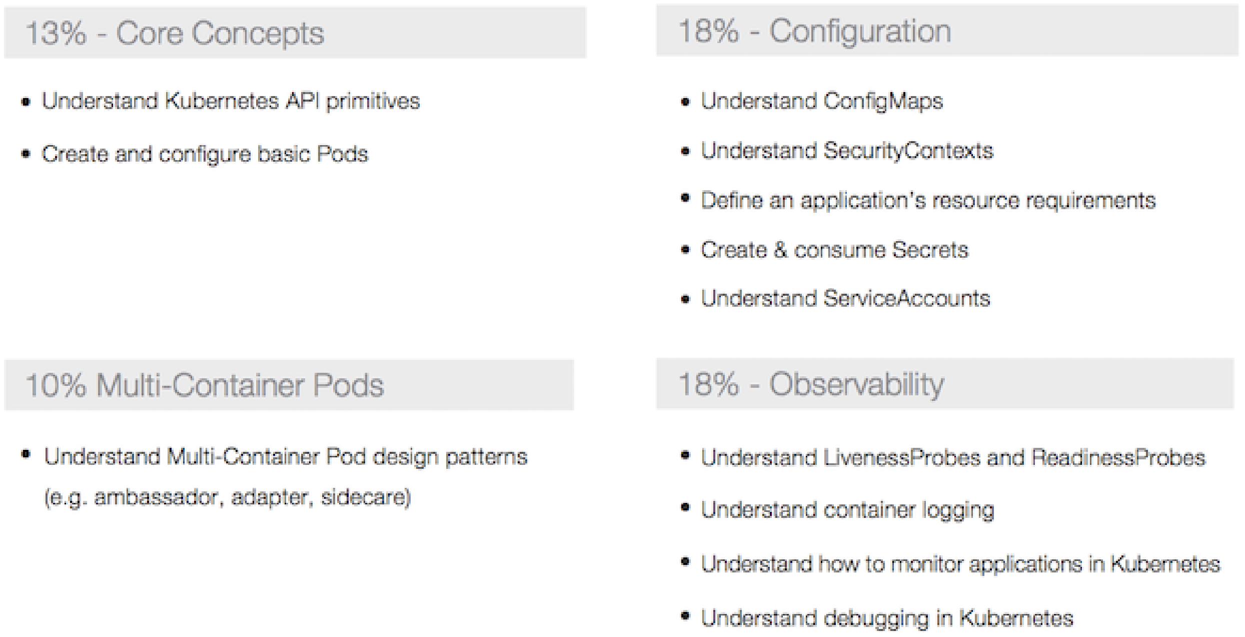
- Logistics
- What to expect
- Bit of advice

From https://training.linuxfoundation.org/certification/certified-kubernetes-application-developer-ckad/

This exam curriculum includes these general domains and their weights on the exam:

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

Curriculum



13% - Core Concepts

- Understand Kubernetes API primitives
- Create and configure basic Pods

18% - Configuration

- Understand ConfigMaps
- Understand SecurityContexts
- Define an application's resource requirements
- Create & consume Secrets
- Understand ServiceAccounts

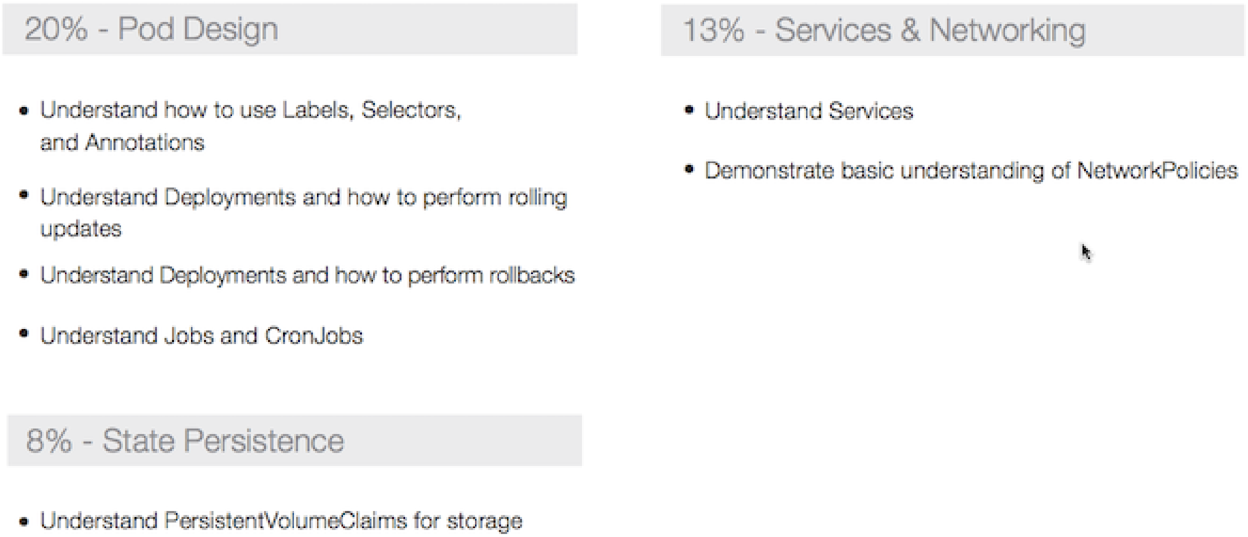
10% Multi-Container Pods

- Understand Multi-Container Pod design patterns (e.g. ambassador, adapter, sidecare)

18% - Observability

- Understand LivenessProbes and ReadinessProbes
- Understand container logging
- Understand how to monitor applications in Kubernetes
- Understand debugging in Kubernetes

Curriculum



20% - Pod Design

- Understand how to use Labels, Selectors, and Annotations
- Understand Deployments and how to perform rolling updates
- Understand Deployments and how to perform rollbacks
- Understand Jobs and CronJobs

8% - State Persistence

- Understand PersistentVolumeClaims for storage

13% - Services & Networking

- Understand Services
- Demonstrate basic understanding of NetworkPolicies

What to expect

- Intense
- Some questions are easy some take more time
- Know your `vi`
- Look up the documentation and paste

Review the `kubectl` cheat sheet

`kubectl` Cheat sheet

```
kubectl config use-context my-cluster-name
kubectl get pods -o wide
kubectl get services --sort-by=.metadata.name
kubectl get pods -o json | jq '.items[] ...
kubectl edit
kubectl run -i --tty busybox --image=busybox -- sh
kubectl port-forward my-pod 5000:6000
kubectl top
kubectl exec
kubectl cp
kubectl scale
kubectl run .... --dry-run -o json
kubectl get ... -- export
```

Practice - Pods

Create a Pod with name `newyork` and container image `redis`

Practice - Pods

Create a Pod with name `newyork` and container image `redis`

Create a Pod with name `albany` a container image `busybox` that sleeps and define an environment variable `VELOCITY` whose value is `rocks`

Practice - Pods

Create a Pod with name `newyork` and container image `redis`

Create a Pod with name `albany` a container image `busybox` that sleeps and define an environment variable `VELOCITY` whose value is `rocks`

Create a Pod with two containers, one name `foo`, the other one named `bar` . The first one with the image nginx and the second one with the image redis. The Pod should be called `foobar` and have the label `foo=bar`

Practice - Pods

Given the following manifest, set the container resource request for memory to 128 Mega bytes.

```
apiVersion: v1
kind: Pod
metadata:
  name: question10
spec:
  containers:
  - name: nginx
    image: nginx
```

Tip: Use the `polinux/stress` container from the documentation to convince yourself that it works. i.e run the example in the doc

Practice - Pods

Create a deployment object called `foo123` with 2 replicas that uses image nginx.

Once the pods are running scale the deployment to 4

Expose the deployment via a service

Practice -- init-containers

A Python app is containerized in a Docker image `mypytonapp`, it needs some modules defined in a `requirements.txt` file.

Configure an init container that installs the dependencies via a shared volume in a Pod that runs the Python app

Practice - Deployments

Perform a rolling update of Deployment `foo123` by changing the image of container `foo` from `nginx` to `runseb/2048`

Pratice - Services

Given the manifest for a Pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: question10
spec:
  containers:
  - name: nginx
    image: nginx
```

Expose it to the internet by creating a service. Fix any potential issues that may arise.

Practice - Networking

List the network Policies and figure out why they are not working ?

tip Try configure minikube for testing network policies and run a few tests from Ahmet's blog.

Practice - CronJob

Write a cronjob manifest that outputs the date every 5 minutes to `stdout`

Practice - Configuration

Given a file `file.txt` containing the sentence `I will pass CKAD`, mount this file inside a Pod using a configMap and copy the file from the Pod back to the host.

extra what is the size limit of a ConfigMap ?

Practice - Configuration

Given a file `file.txt` containing the sentence `I will pass CKAD`, mount this file inside a Pod using a configMap and copy the file from the Pod back to the host.

extra what is the size limit of a ConfigMap ?

Create a secret and mount it inside a Pod using a Volume.

Practice - Configuration

Given a file `file.txt` containing the sentence `I will pass CKAD`, mount this file inside a Pod using a configMap and copy the file from the Pod back to the host.

extra what is the size limit of a ConfigMap ?

Create a secret and mount it inside a Pod using a Volume.

Do it again but using an environment variable

Practice - Persistency

Create a PVC that requests 500 Megabytes and use this PVC to make the data of a mysql Pod persistent.

Practice - Monitoring

Find the logs of the Pod called `x`

extra what would you use to aggregate the logs of all containers ?

Practice - Monitoring

Find the logs of the Pod called `X`

extra what would you use to aggregate the logs of all containers ?

Among all the Pods running in cluster `Y`, find the Pod that consumes the most CPUs

Practice - Security

Write a Dockerfile for an image that can run in a Pod with a securityContext that does not let run as root.

Write the Pod manifest

Practice - Probes

Come up with an example to showcase the behavior of Liveness and readinessprobes

Practice - Service Account

Create a namespace

Extract the JWT token of the default service account

Set the `kubectl config` profile to access the cluster using this new service account.

Create the RBAC roles to be able to create Pods in the created namespace

Practice - Service Account

Write a toy Python (or your preferred language) app that you containerize (write a Dockerfile) that needs to call the k8s API server

Demonstrate that with the default service account the app calls fail and that when you give the service account the proper privileges it runs properly

tip check out the `kubectl auth can-i` command

Bottom line

- Make sure you know `vi`
- Make sure you know your YAML syntax/linting
- Make sure you know the structure of API objects
- Make sure you know how to navigate the Kubernetes documentation quickly
- Make sure you know `kubectl`

Thank You

And good luck tomorrow