Topics

- What is a Container Manager?
- What is Kubernetes?
- Kubernetes Core Concepts?
  - POD
  - Deployment
  - Services
  - Namespaces
- Kubernetes Advanced Concepts
  - Ingress
  - Secrets
  - Config Maps
  - Custom Resource Definitions

Container Bootcamp

Kubernetes Training

2: What is a Container Manager?

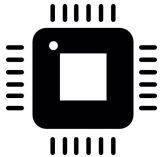
INNOQ

Cloud Native

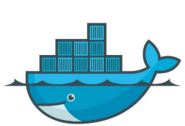
Applications adopting the principles of Microservices packaged and delivered as Containers orchestrated by Kubernetes running on top of Cloud infrastructure

I,C,P as a Service: Terms

IaaS	Virtualization Networking ACLs Firewalls ...



CaaS	Container Runtime Images Container Manager ...



PaaS	“Ready to Use” Solution Stack Logging Monitoring Persistent volumes ...

Advantages *aaS

- Cost Savings
- Pay what you need (although provider needs spare too)
- Faster Time to Market
- Scalability (for peak demands)
- Resiliency
- Easy business growth (or reduction)

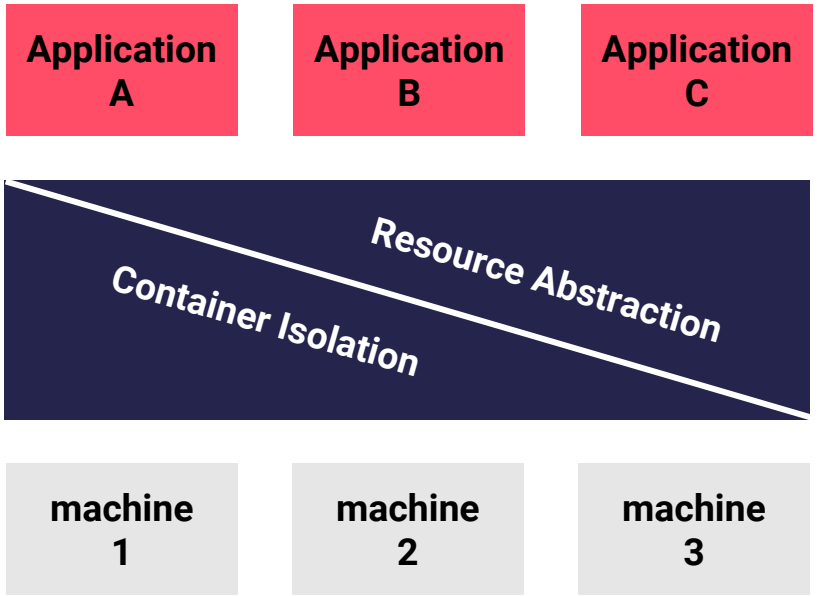
Is that all?

Without influence to

Development, Test, Deployment, Production, Architecture?

Aim: Ressourcen Abstraction

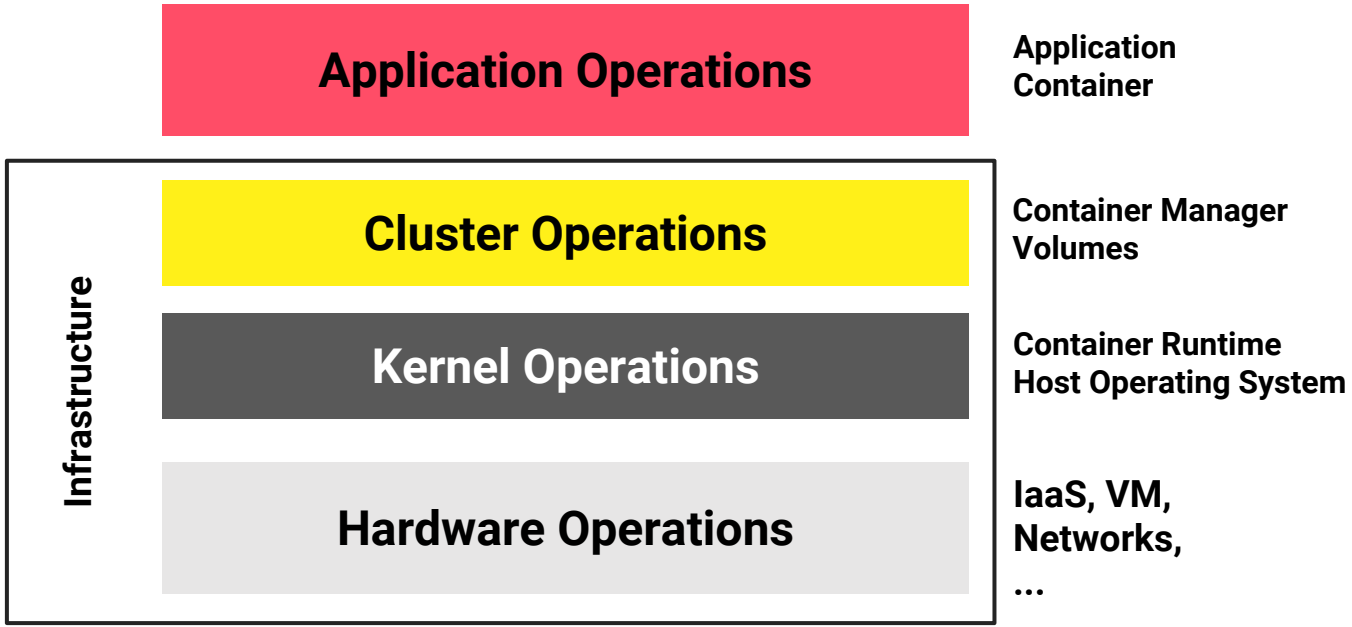
Applications consume Resources, not Machines



Why?

- **Resiliency**
- **Scalability**
- **. . .**

Advantage: Decoupling Operations



Goals:

1. **Any operations should be Application Oriented**

2. **Platform should automate routine tasks**

   (placement, healthchecks, healing, scaling, …)

3. **NFR should be moved to the platform**

   (discovery, jobs, log aggregation, metrics collection, …)

4. **Allow developers to code the business domain**

Container Bootcamp

Kubernetes

3: What is Kubernetes?

INNOQ

What is Kubernetes?

Kubernetes is an open-source platform for

- **Automating deployment**
- **Scaling**
- **Operations of application containers across clusters of hosts**
- **Providing container-centric infrastructure.**

It is:

- **portable: public, private, hybrid, multi-cloud**
- **extensible: modular, pluggable, hookable, composable**
- **self-healing: auto-placement, auto-restart, auto-replication, auto-scaling**

Kubernetes is not

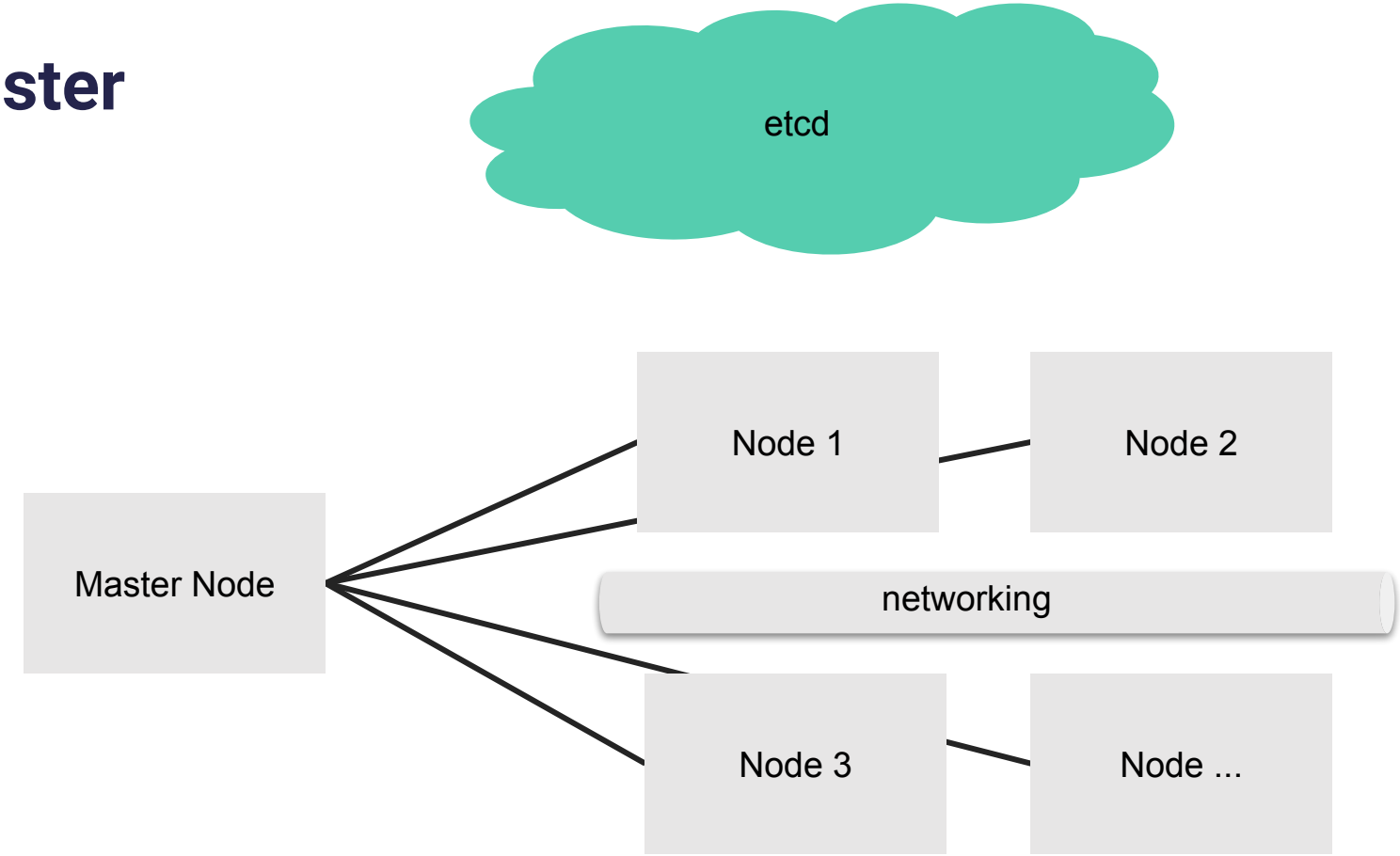
… an all-inclusive PaaS (Platform as a Service)

and does not:

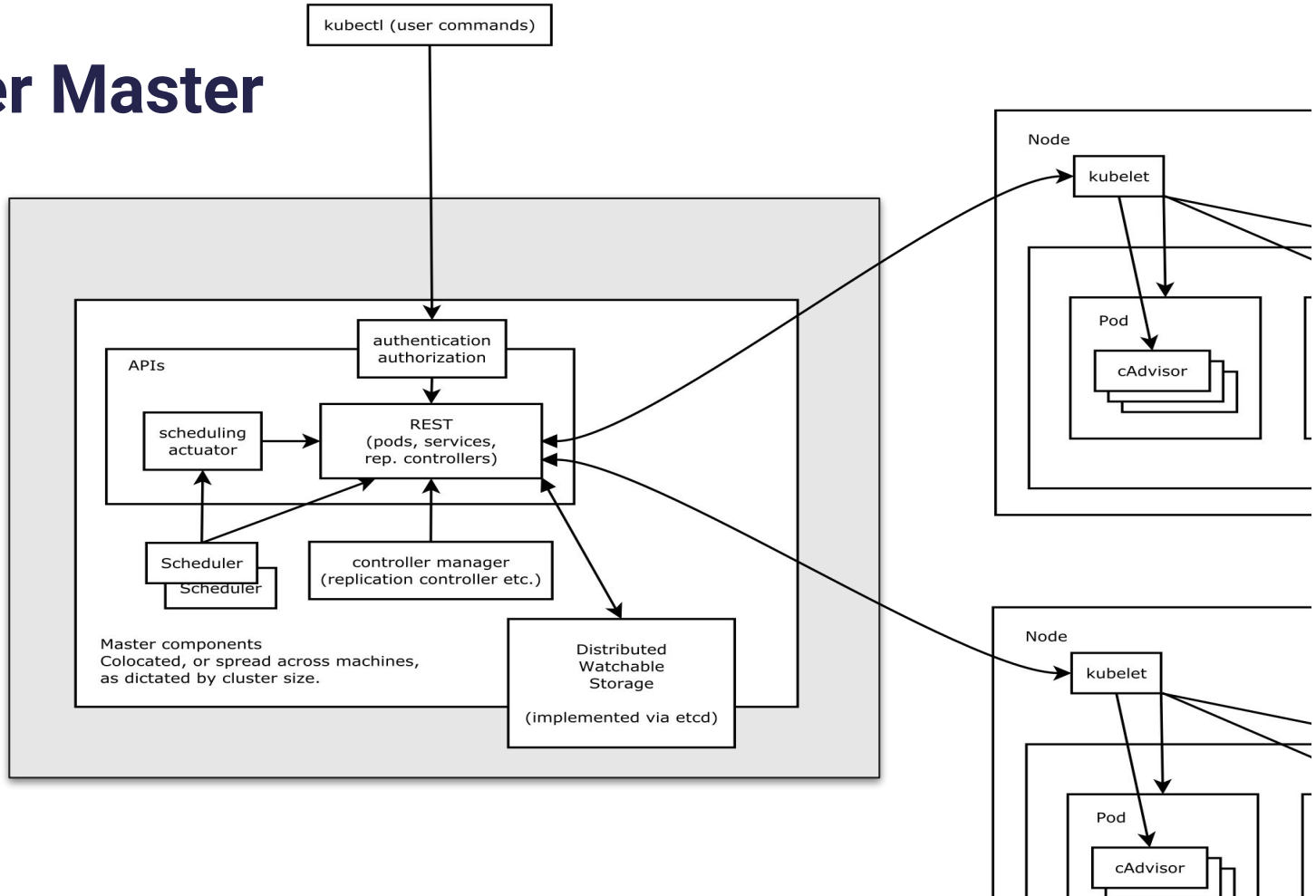
- … dictate application frameworks
- … restrict the set of supported language runtimes
- … distinguish “Apps” from “Services”
- … provide any middleware or framework
- … provide a logging/monitoring/alerting or storage system
- … have an opinion in the source-to-image or CI/CD workflow space
- … have a click-to-deploy service marketplace

K8s Processes

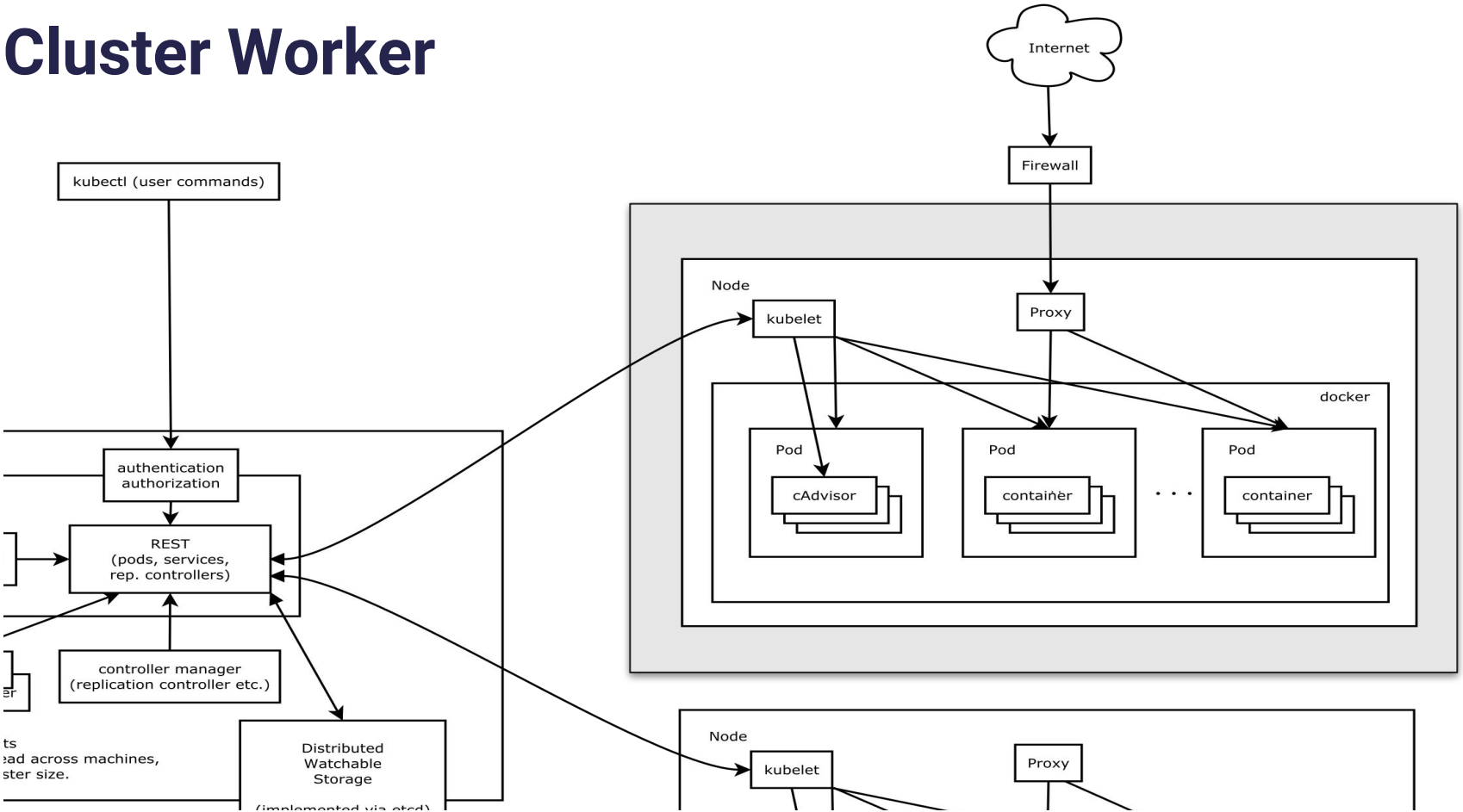
Cluster



Cluster Master



Cluster Worker



Cluster Modules

- **kubelet**
  Manages pods and their containers, their images, their volumes, etc.
- **kube-proxy**
  A distributed multi-tenant, round-robin load-balancer
- **etcd**
  Distributed reliable key-value store, stores all persistent master state
- **API Server**
  Mainly processes REST operations, validates them, and updates the corresponding objects in etcd
- **Scheduler**
  binds unscheduled pods to nodes
- **Controller Manager Server**
  Performs all other cluster-level functions (f.e. node discovery)

API Server

- Is a HTTP API to your cluster
  - For Users
  - For code running in containers

- `kubectl` is the API client

- API reference (object reference) can be found here: https://kubernetes.io/docs/api-reference/v1.9/

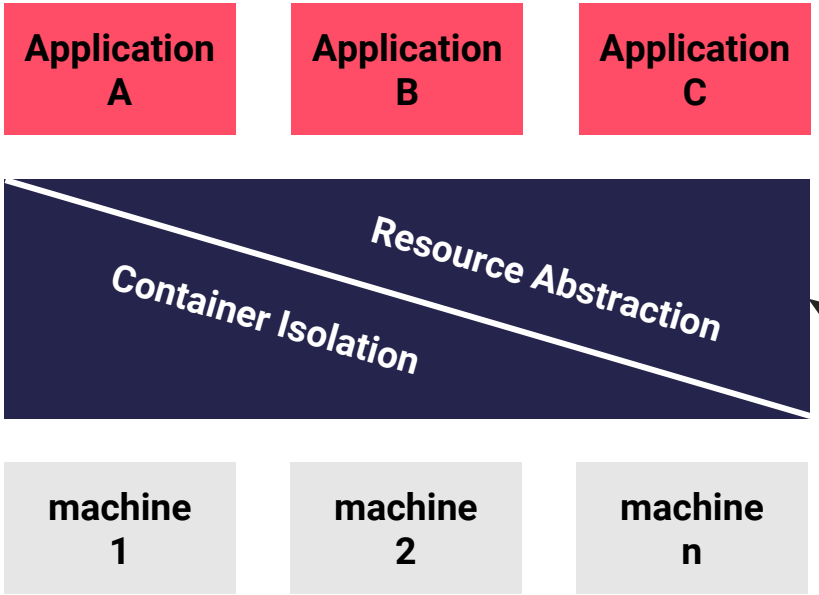
  . . .

Container Bootcamp

Kubernetes

4: Kubernetes Core Concepts?

Ressourcen Abstraction



YAML / JSON / HTTP

- Service Endpoints
- Group of Container
- Configurations
- Network
- ...

K8s Abstractions

- **Pod**
- **Deployment**
- **Service / Service Discovery**
- **Persistent Volumes**
- **Config Maps**
- **StatefulSets**
- **Network Policies**
- **Ingress**

- Images
- Labels and Selectors
- Replication Controller
- Secrets
- Names
- Namespaces
- Nodes
- Security Context
- Service Accounts
- Annotations
- Daemon Sets
- Ingress Resources
- Horizontal Pod Autoscaling
- Jobs
- Resource Quotas
- Replica Sets

Setup for Exercises

1. ~~http://60minutek8s.ch.innoq.io/ (daimler/tss)~~
2. Setup kubectl to point to our test cluster.
   -> Copy the `kubeconfig` to a folder of your choice

```
$ export KUBECONFIG="$PWD/kubeconfig-X"
$ set KUBECONFIG="/path/to/kubeconfig"
$ kubectl get all
No resources found.
```

3. Play around with `get pod`, `get node`, `get service` …

Setup for Exercises

1. Install Bash Completion

```
$ source <(kubectl completion bash)

or

$ source <(kubectl completion zsh)
```

Bash Completion see: `https://kubernetes.io/docs/tasks/kubectl/install/`

Which Kubernetes Objects to Learn?

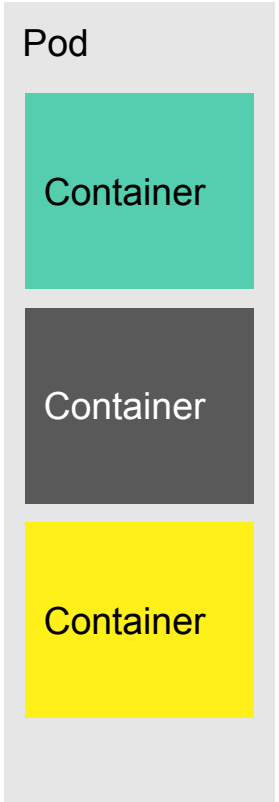
- PODs
- Deployment
- Services
- Namespaces


POD

What is a pod?

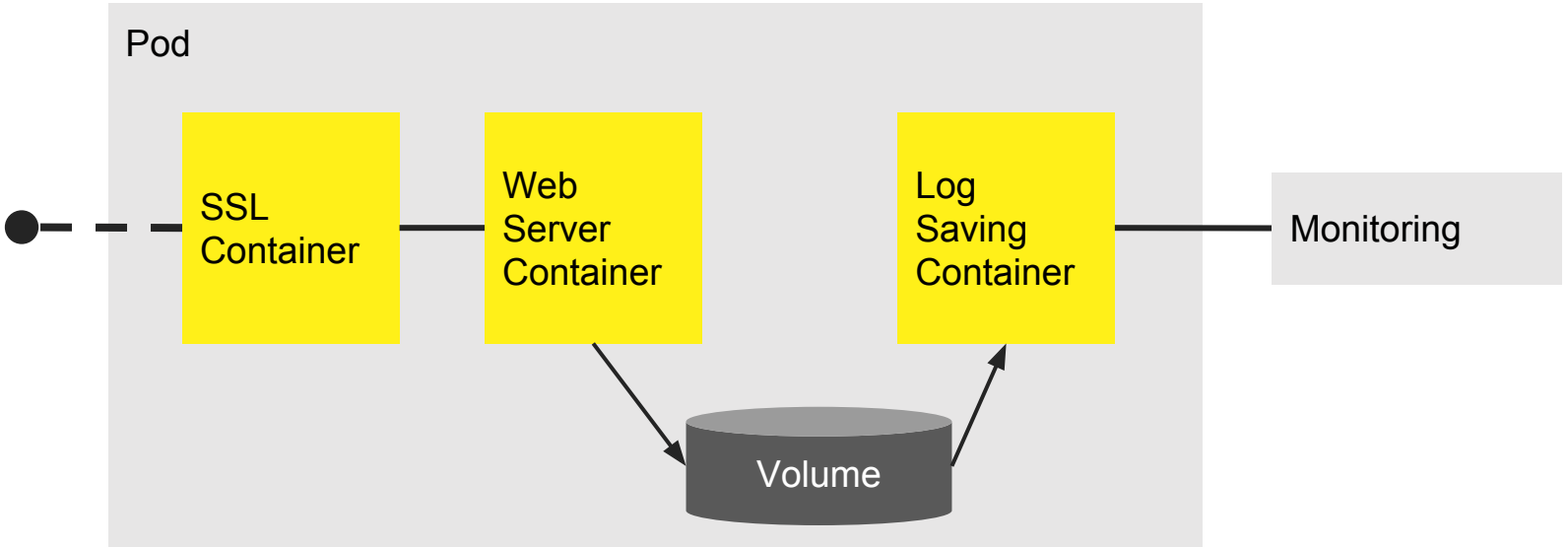
“Logical Host”

- it contains one or more containers which are tightly coupled
- co-located and co-scheduled
- run in a shared context
- shared storage (volumes)
- has its own IP



Why one or more container?

Sidecar / Sidekick pattern



POD example

```
apiVersion: v1
kind: Pod
metadata:
  name: redis-django
  labels:
    app: web
spec:
  containers:
    - name: key-value-store
      image: redis
    - name: frontend
      image: django
```

Exercise

1. Create a first Pod

```
$ watch kubectl get po

$ kubectl create -f myclock.yaml
```

myclock.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: clock
  labels:
    app: training
spec:
  containers:
    - name: my-clock
      image: innoq/clock
```

```
$ kubectl get pod
$ kubectl get po clock -o json
$ kubectl get po clock -o jsonpath='{.spec.containers[0].image}'
$ kubectl logs clock
$ kubectl logs -f clock
$ kubectl describe po clock
```

JSONPath - XPath for JSON

Exercise

1. Create a Webserver Pod

```
$ kubectl create -f myserver.yaml

$ kubectl port-forward server 8080:80
```

myserver.yaml

```
apiVersion: v1
kind: Pod
metadata:
 name: server
 labels:
  app: training
spec:
 containers:
  - name: my-server
    image: innoq/k8s-training-webserver
```

2. Open Terminal with `kubectl logs -f server`

3. Check http://localhost:8080

4. Check all known `kubectl get/describe` commands...

Can we connect to a Pod IP? Yes, we can...

```
$ kubectl describe po server
Name:         server
. . .
IP:       10.244.3.27
. . .
$ kubectl create -f busybox.yaml
$ kubectl exec busybox -it sh
/ # telnet 10.244.3.27 80
GET /
<!doctype html>
<html lang="en">
  <head>
    <title>K8s Training</title>
  </head>
  <body>
    <h1>Hello Kubernetes Training</h1>
  </body>
</html>
Connection closed by foreign host
/ # exit
```

```
apiVersion: v1
kind: Pod
metadata:
  name: busybox
spec:
  restartPolicy: Always
  containers:
  - image: busybox
    command:
      - sleep
      - "3600"
    imagePullPolicy: IfNotPresent
    name: busybox
```

Command, Args in a Pods definition

When you create a Pod, you can define a command and arguments for the containers that run in the Pod.

This overwrites, whatever is provided by the container image

```
apiVersion: v1
kind: Pod
metadata:
  name: command-demo
  labels:
    purpose: demonstrate-command
spec:
  containers:
  - name: command-demo-container
    image: debian
    command: ["printenv"]
    args: ["HOSTNAME", "KUBERNETES_PORT"]
```

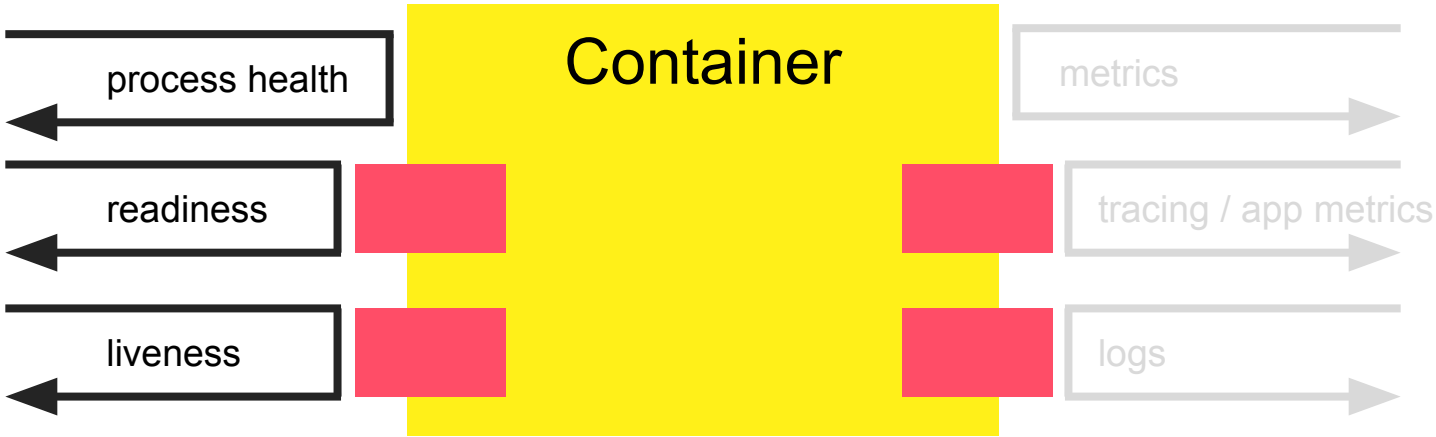
A Pods Lifecycle

Lifecycle of a Pod

- **Pending:** The pod has been accepted by the system, but one or more of the container images has not been created
- **Running:** The pod has been bound to a node, and all of the containers have been created
- **Succeeded:** All containers in the pod have terminated in success, and will not be restarted
- **Failed:** All containers in the pod have terminated, at least one container has terminated in failure

Container Probes

Observable Interior



Container Probes

- **LivenessProbe:**
  indicates whether the container is running
  - On failure, container will be killed and subjected to its RestartPolicy
  - Default state of Liveness before the initial delay is Success
  - State when no probe is provided is Success.

- **ReadinessProbe:**
  indicates whether the container is ready to service requests
  - On failure, the pod’s IP address will be removed from all services
  - State before the initial delay is Failure
  - State when no probe is provided is Success

Container Probes

A Probe is a diagnostic performed periodically on a container (!)

- **ExecAction**: executes a command inside the container
  (exit status code 0 on success)

- **TCPSocketAction**: a tcp check against the container's IP address
  (port is open on success)

- **HTTPGetAction**: an HTTP Get against the container's IP address
  (200 >= status code < 400, on success)

Exercise

1. Create a livenessProbe

```
$ kubectl delete po server

$ kubectl create -f myserver.yaml
```

What happens?

What is CrashLoopBackOff?

How to fix this?

myserver.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: server
  labels:
    app: training
spec:
  containers:
    - name: my-server
      image: innoq/k8s-training-webserver
      livenessProbe:
        httpGet:
            path: /healthz
            port: 80
        periodSeconds: 1
```

Example: Probe with HTTP

- **when "host" is not defined, "PodIP" will be used**
  - **f.e.** `host: my-host`
- **when "scheme" is not defined, "HTTP" scheme will be used. Only "HTTP" and "HTTPS" are allowed**
  - **f.e.** `scheme: HTTPS`

```
livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
          - name: X-Custom-Header
            value: Awesome
      initialDelaySeconds: 15
      timeoutSeconds: 1
```

Example: Probe with exec

- Command is executed
  - Initially after 5 seconds
  - Every 5 seconds

```
livenessProbe:
  exec:
    command:
    - cat
    - /tmp/healthy
  initialDelaySeconds: 5
  periodSeconds: 5
```

Example: Probe with exec

```
apiVersion: v1
kind: Pod

metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - name: liveness
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 12; rm -rf /tmp/healthy; sleep 600
    image: busybox
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
```

Container Probes: When to use what?

- A container should crash on its own, whenever it encounters an issue
  -> use `RestartPolicy` (Always, OnFailure)
  -> so no liveness probe needed

- If you'd like to start sending traffic to a pod only when a probe succeeds
  -> specify a ReadinessProbe

- If a container wants the ability to take itself out of service for maintenance
  -> specify a ReadinessProbe

Exercise

1. *Start pod* `liveness.yaml`
2. *Watch events with* `kubectl describe pod liveness-exec`

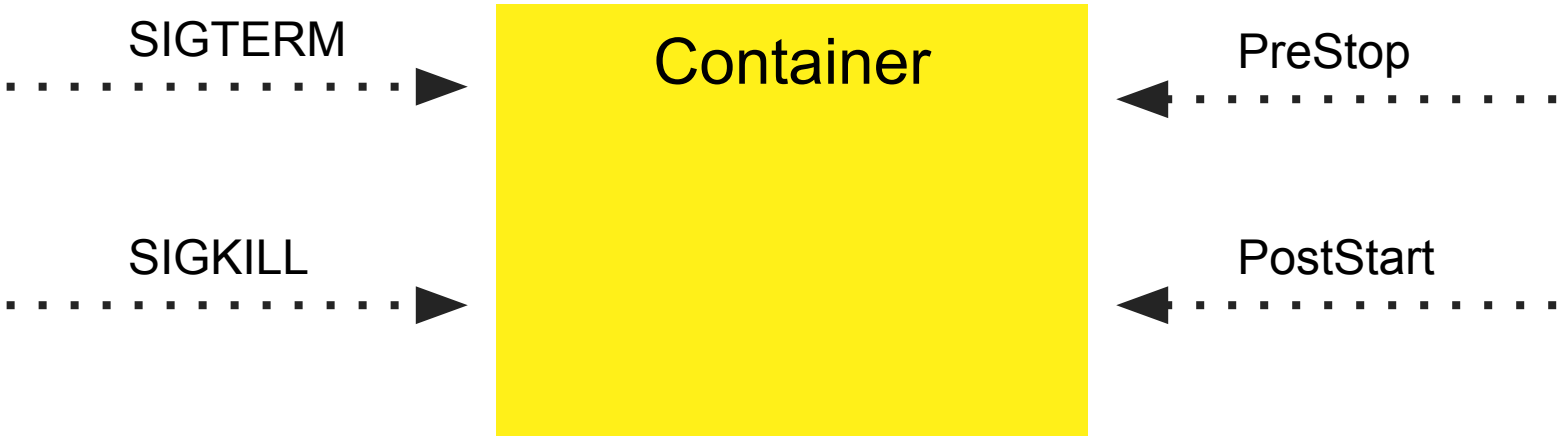
How fast does it get restarted?

And how often?

3. *Remove the Pod with* `kubectl delete pod liveness-exec`

Lifecycle Hooks

Lifecycle conformance



Deployment

Deployment

- A Deployment controller provides declarative updates for Pods and ReplicaSets, therefore
  - Supervises multiple pods across multiple nodes
  - Ensures that a specified number of pod “replicas” are running
  - Similar to a process supervisor
  - recommend, even if your application requires only a single pod

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
        role: backend
    spec:
      containers:
      - name: nginx
        image: nginx:1.12
        ports:
        - containerPort: 80
```

Deployment

- **Deployment creates pods from template**

Pod
Template
for Pod creation

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
        role: backend
    spec:
      containers:
      - name: nginx
        image: nginx:1.12
        ports:
        - containerPort: 80
```

Exercise

1. *Create a Deployment via* `nginx.yaml`
2. *What shows us* `kubectl get all` *?*
3. *Describe one of it's created Pods and the Deployment (labels?)*
4. *Delete one* `nginx-???` *Pod. Does it get restarted?*
5. `kubectl delete deployment nginx --grace-period=0`

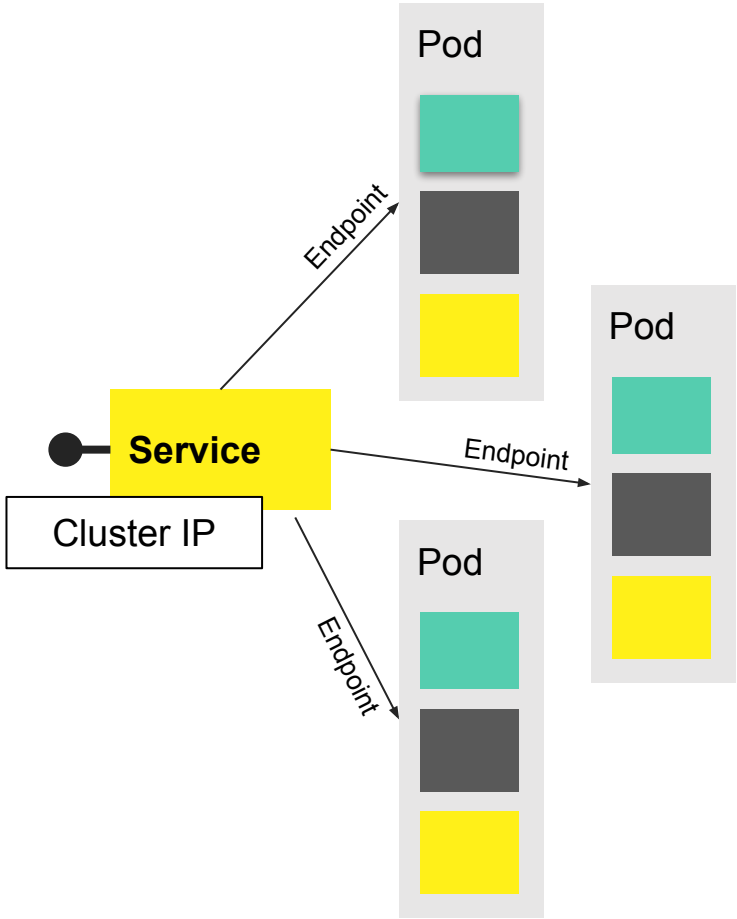
Services

What is a Service?

- Provides Service Discovery by DNS
- Is a distributed multi-tenant, round-robin load-balancer
- is an abstraction which defines access to f.e. a logical set of Pods

Service

- Is an object, similar to Pod
- Delegates ports
- Has a name
- Selects a set of endpoints
  - PODs as endpoints (using Selectors)
  - Specific endpoints (as Endpoints Object)
- “Speaks” UDP/TCP
- Has different *Service Types*



Exercise Service

1. Create a Service www:

```
$ kubectl create -f myservice.yaml

$ kubectl describe service www
```

myservice.yaml

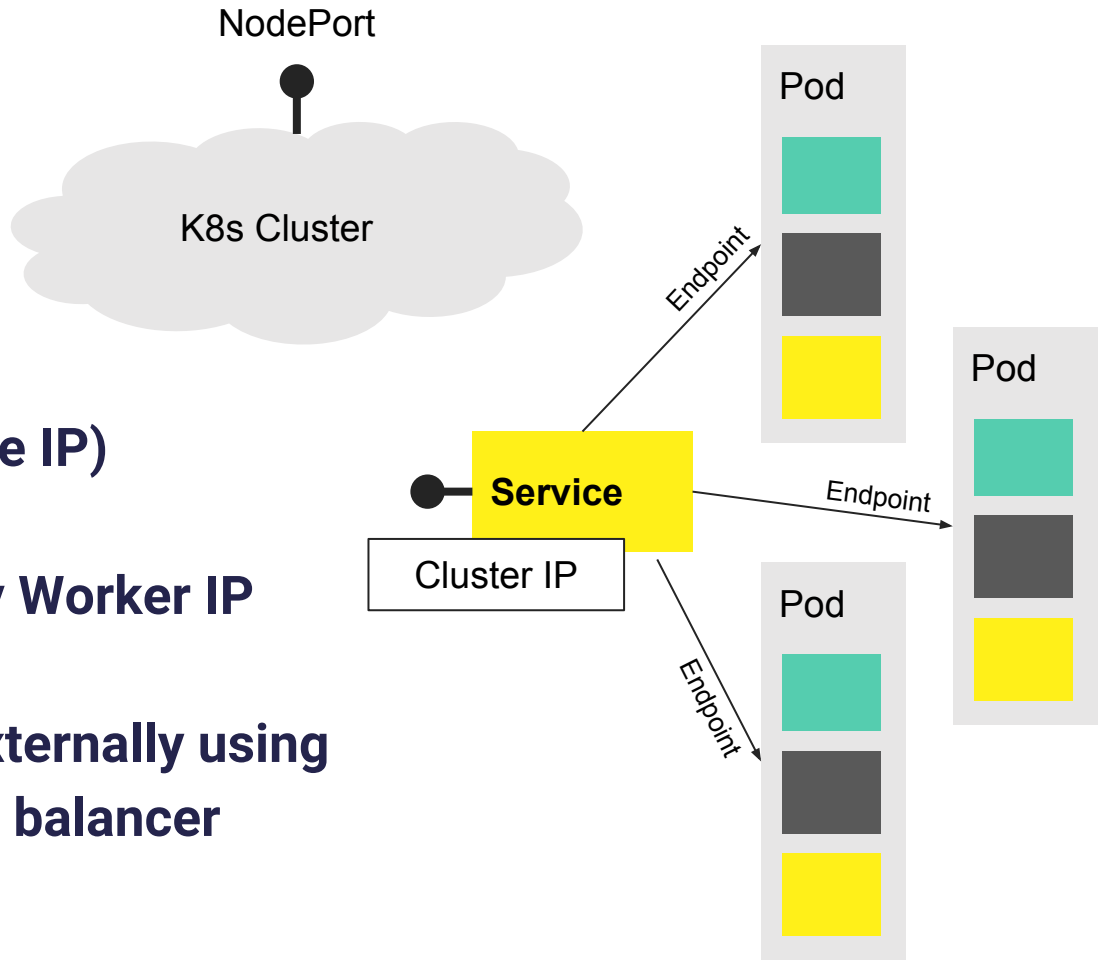
```
apiVersion: v1
kind: Service
metadata:
  name: www
spec:
  ports:
    - port: 80
      protocol: TCP
  selector:
    app: nginx
```

2. Delete the pod, what happens to the service?
3. Create another server2 pod, what happens to the service?
4. Execute: `kubectl exec busybox nslookup www`

Exercise Service

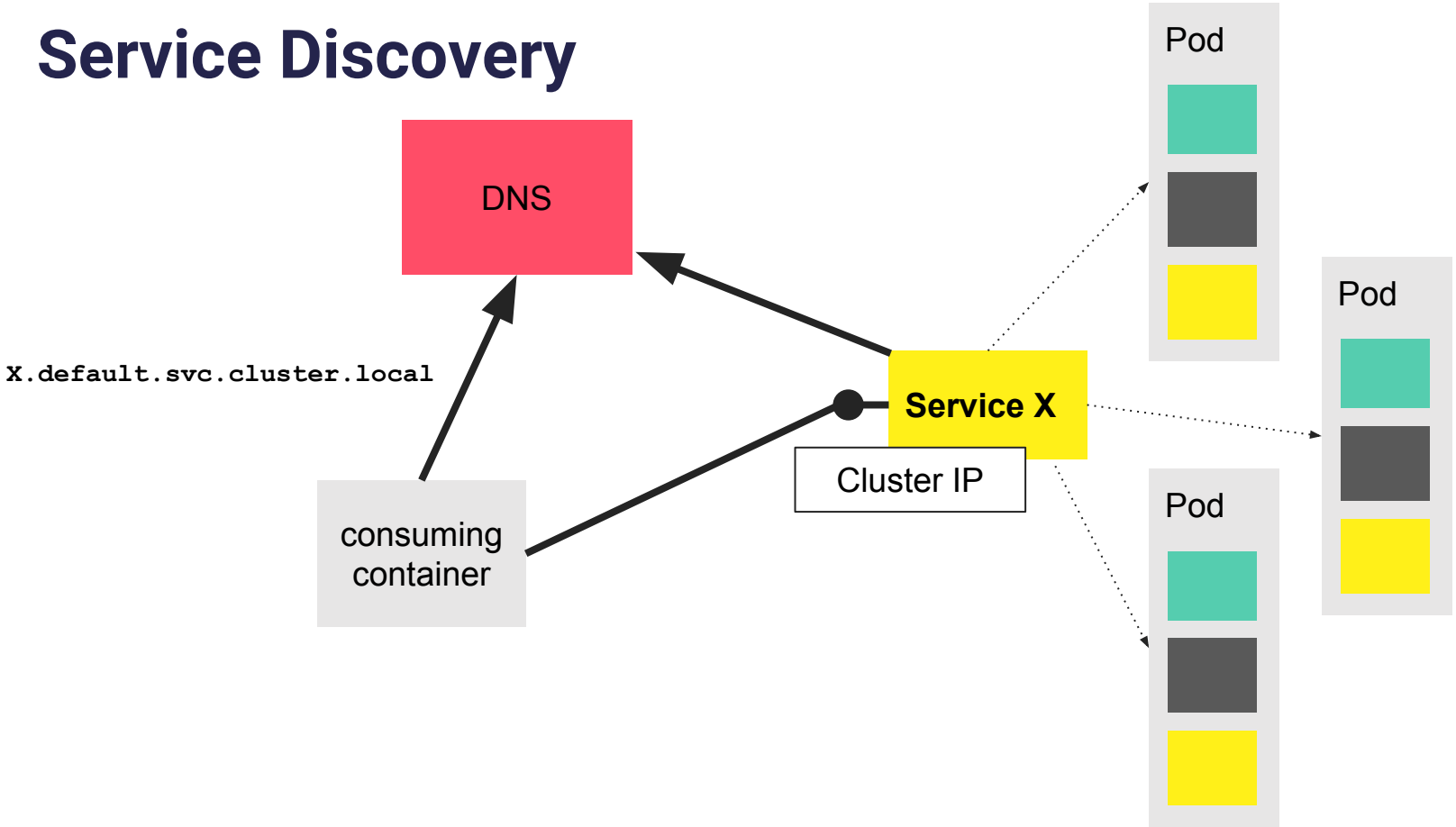
```
$ kubectl describe service www
Name:              www
Namespace:         chris
Labels:            <none>
Selector:          app=training
Type:              ClusterIP
IP:                10.0.195.251
Port:              <unset>  80/TCP
Endpoints:         10.244.80.9:80,10.244.98.8:80
Session Affinity:     None
No events.
$ kubectl exec busybox nslookup www
Server:    10.0.0.10
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local

Name:      www
Address 1: 10.0.195.251 www.chris.svc.cluster.local
```

Service Types

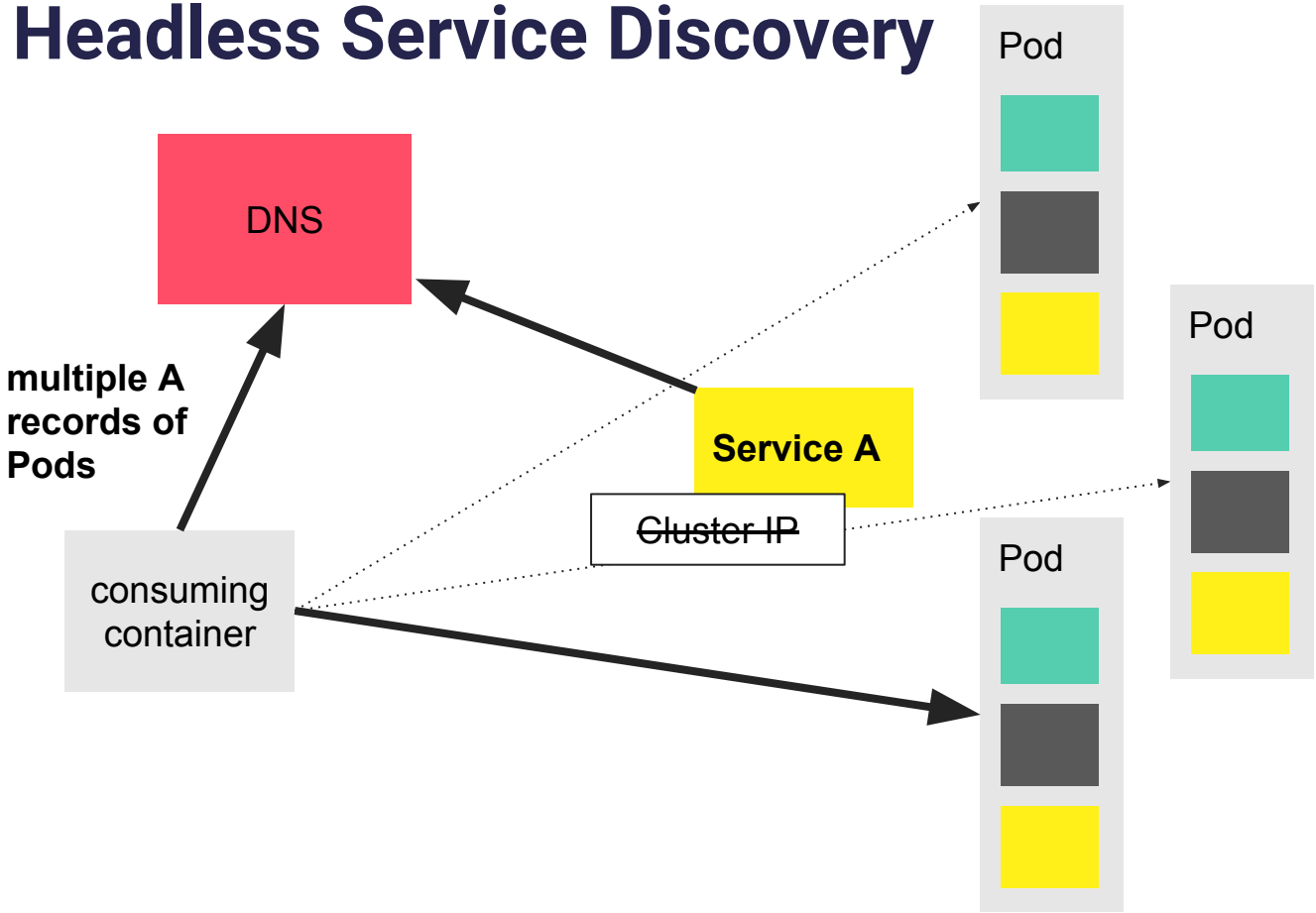


- **As a Cluster IP (Service IP)**
- **NodePort**
  Port available on every Worker IP
- **Load Balancer**
  Exposes the service externally using a cloud provider's load balancer

Service Discovery



Headless Service Discovery




```
apiVersion: v1
kind: Service
metadata:
  name: www
spec:
  clusterIP: "None"
  ports:
    - port: 80
      protocol: TCP
  selector:
    app: nginx
```

Exercise Service

```
$ kubectl exec busybox -i -t -- sh
/ # telnet wwwep 80
GET /
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>. . .</html>
Connection closed by foreign host
/ # nslookup wwwext
Server:    10.0.0.10
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local

Name:      wwwext
Address 1: 2a02:2e0:3fe:1001:302:: redirector.heise.de
Address 2: 193.99.144.80 redirector.heise.de
/ # nslookup www
Server:    10.0.0.10
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local

Name:      www
Address 1: 10.0.195.251 www.chris.svc.cluster.local
```

Namespaces

Namespaces

- **Are intended for use in environments with many users spread across multiple teams, or projects**
- **Provide a scope for names**
  - Names of resources need to be unique within a namespace, but not across namespaces
- **Are a way to divide cluster resources between multiple uses**
  - Resource quota
  - Namespace Isolation Policy (inbound traffic)
- **Service DNS entries have the form**

`<service-name>.<namespace-name>.svc.cluster.local`

Container Bootcamp

Kubernetes

5: Kubernetes Advanced Concepts

INNOQ

What to Learn?

- Ingress
- Secrets
- Config Maps

Ingress or How to Reach the Endpoints?

Ingress

Ingress is a collection of rules that allow inbound connections to reach the endpoints defined by a backend

Ingress can be configured to give services

- **externally-reachable urls and path**
- **load balance traffic**
- **terminate SSL**
- **offer name based virtual hosting**

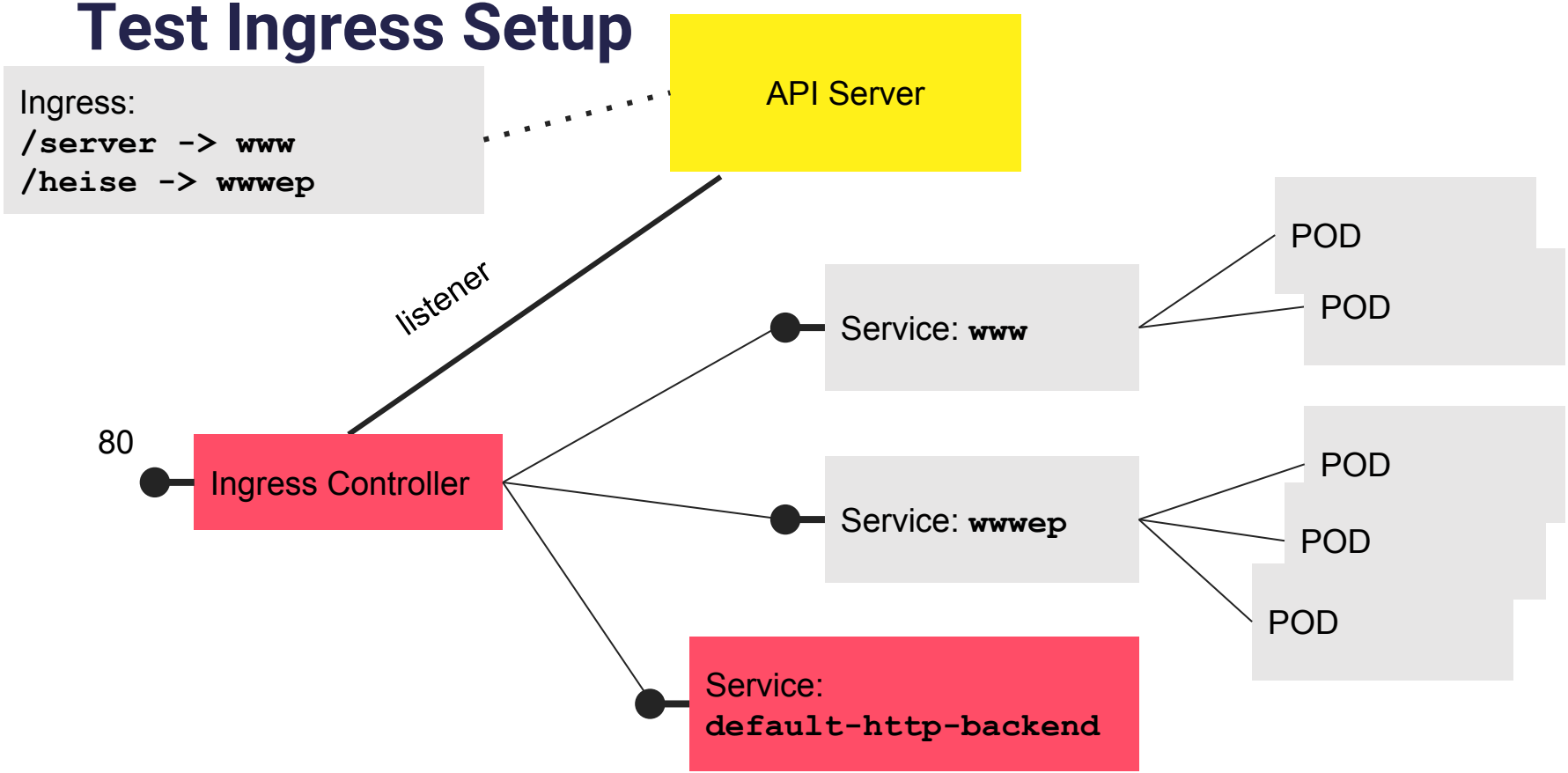
Exercise Ingress

```
$ curl bootcamp.ch.innoq.io/heise -H 'Host: innoq.training.com'
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="https://www.heise.de/">here</
</body></html>
$ curl 185.19.31.109/server -H 'Host: innoq.training.com'
<!doctype html>
<html lang="en">
  <head>
    <title>K8s Training</title>
  </head>
  <body>
    <h1>Hello Kubernetes Training</h1>
  </body>
</html>
```

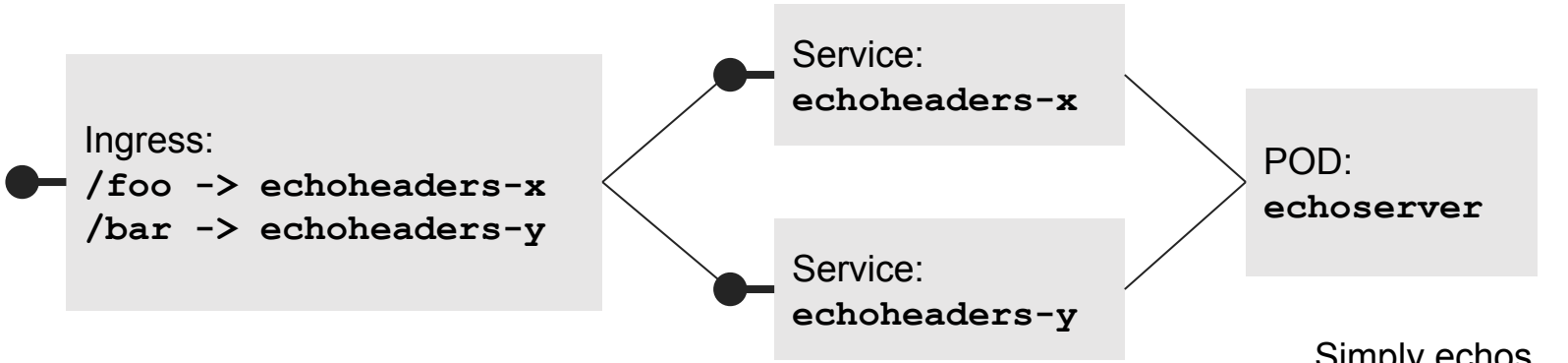
myingress.yaml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    ingress.kubernetes.io/rewrite-target: /
  name: myingress
spec:
  rules:
  - host: innoq.training.com
    http:
      paths:
      - path: /server
        backend:
          serviceName: www
          servicePort: 80
      - path: /heise
        backend:
          serviceName: wwwep
          servicePort: 80
```

Test Ingress Setup



Test Ingress Setup (logical)



Simply echos
HTTP requests
To console

Exercise Ingress

```
$ kubectl run echoheaders --image=gcr.io/google_containers/echoserver:1.4 --replicas=1 --port=8080
$ kubectl expose deployment echoheaders --port=80 --target-port=8080 --name=echoheaders-x
$ kubectl expose deployment echoheaders --port=80 --target-port=8080 --name=echoheaders-y

$ kubectl create -f testingress.yaml

$ kubectl describe ingress testingress
Name:             testingress
Namespace:        chris
Address:          185.19.31.109
Default backend:  default-http-backend:80 (<none>)
Rules:
  Host            Path  Backends
  ----            ----  --------
  innoq.training.com
                  /foo  echoheaders-x:80 (<none>)
                  /bar  echoheaders-y:80 (<none>)
```

testingress.yaml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: testingress
spec:
  rules:
  - host: bootcamp.ch.innoq.io
    http:
      paths:
      - path: /foo
        backend:
          serviceName: echoheaders-x
          servicePort: 80
      - path: /bar
        backend:
          serviceName: echoheaders-y
          servicePort: 80
```

Exercise Ingress

```
$ curl bootcamp.ch.innoq.io/bar -H 'Host: innoq.training.com'
CLIENT VALUES:
client_address=10.244.98.6
command=GET
real path=/bar
query=nil
request_version=1.1
request_uri=http://innoq.training.com:8080/bar

SERVER VALUES:
server_version=nginx: 1.10.0 - lua: 10001

HEADERS RECEIVED:
accept=*/*
connection=close
host=innoq.training.com
user-agent=curl/7.51.0
x-forwarded-for=10.244.98.1
x-forwarded-host=innoq.training.com
x-forwarded-port=80
x-forwarded-proto=http
x-real-ip=10.244.98.1
BODY:
-no body in request-
```

Secrets

Secrets

- intended to hold sensitive information, such as
  - passwords
  - OAuth tokens
  - ssh keys
- safer and more flexible than putting it verbatim in a pod definition or in a docker image
- reduces the risk of accidental exposure, during the workflow of creating, viewing, and editing pods.
- can be used in three ways:
  - as files in a volume mounted on one or more of its containers,
  - as Environment Variables
  - or used by kubelet when pulling images for the pod

But...

- **In the API server secret data is stored as plaintext in etcd; therefore:**
  - limit access to etcd to admins
  - wipe/shred disks used by etcd
- **Apps still need to protect the value of secret after reading it from the volume**
- **Users who can create a pod that uses a secret can also see the value of that secret**
- **No control which users of a Kubernetes cluster can access a secret**

-> Changes with RBAC and etcd encryption (K8s > 1.7.x)

Example:

```
$ 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 describe secret db-user-pass
Name:      db-user-pass
Namespace: chris
Labels:          <none>
Annotations:     <none>

Type: Opaque

Data
====
password.txt:    12 bytes
username.txt:    5 bytes
```

Example:

```
$ kubectl get secret db-user-pass -o yaml
apiVersion: v1
data:
  password.txt: MWYyZDFlMmU2N2Rm
  username.txt: YWRtaW4=
kind: Secret
metadata:
  creationTimestamp: 2017-01-30T12:57:21Z
  name: db-user-pass
  namespace: chris
  resourceVersion: "692297"
  selfLink: /api/v1/namespaces/chris/secrets/db-user-pass
  uid: a33a0e3f-e6eb-11e6-903c-06c92a0003b6
type: Opaque
```

Exercise Secrets

1. Create secret mysecret
2. Try to decode it with … `get` … `-o yaml` and … `base64 --decode`
3. Review / create `secretbusybox` Pod and view its log
4. `kubectl exec -it` **`secretbusybox`** `sh`
   And cat both files in `/etc/secrets`
5. BTW:

   What are all these Env Variables?
6. Does a secret get updated?

mysecret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

Config Maps

Config Map

holds key-value pairs of configuration data

similar to Secrets, but designed to more conveniently support working with strings that do not contain sensitive information

can be used to:

- Populate the values of environment variables
- Set command-line arguments in a container
- Populate config files in a volume

Config Map

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
data:
  special.how: very
  special.type: charm
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: env-config
data:
  log_level: INFO
  stage: INT
```

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.how
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.type
      envFrom:
        - configMapRef:
            name: env-config
  restartPolicy: Never
```

Exercise Config Maps

1. *Try to create ConfigMaps (see README in Creating-ConfigMaps)*
2. *Review and try the Config Map examples (3 Use Cases)*
3. *What happens if a config map does not exist? Or a Key/Value?*

Hints:

`kubectl get po --show-all` ***(shows finished PODs)***

`kubectl get configmap game-config -o yaml` ***(shows CM content)***

Container Bootcamp

Kubernetes

10: Custom Resource Definitions (CRD)

INNOQ

Custom Resource Definition

Kubernetes comes with many built-in API objects

CRD extend Kubernetes with their own API object type in order to do custom automation (API Server creates REST Path)

Each CRD has the following:

- metadata - Standard Kubernetes object metadata.
- kind - The kind of the resources described by this third party resource.
- description - A free text description of the resource.
- versions - A list of the versions of the resource.

Example innoq-cron-tab

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: innoqcrontabs.innoq.com
spec:
  group: innoq.com
  names:
    kind: InnoqCronTab
    listKind: InnoqCronTabList
    plural: innoqcrontabs
    singular: innoqcrontab
    shortNames:
    - ict
  scope: Namespaced
  version: v1
description: "A specification of a Pod to run on a cron
style schedule"
```

Creates a new RESTful API endpoint at

`/apis/stable.innoq.com/v1/namespaces/<namespace>/innoqcrontabs/…`

-> API endpoint URL and support for CRUD operations, and watch API

You can then manage our InnoqCronTab objects using kubectl

Example innoq-cron-tab

```
apiVersion: innoq.com/v1
kind: InnoqCronTab
metadata:
  name: my-new-cron-object
cronSpec: "* * * *"
image: innoqs-awesome-cron-image
```

`InnoqCronTab` instance `my-new-cron-object`

Exercise CRD

1. Create the resource `innoq-cron-tab.stable.innoq.com` with resource.yaml

2. Create an instance of InnoqCronTab

3. Use get, describe to review the crd / InnoqCronTab

Introducing Operators

Operator

... is an application-specific controller that extends the Kubernetes API to

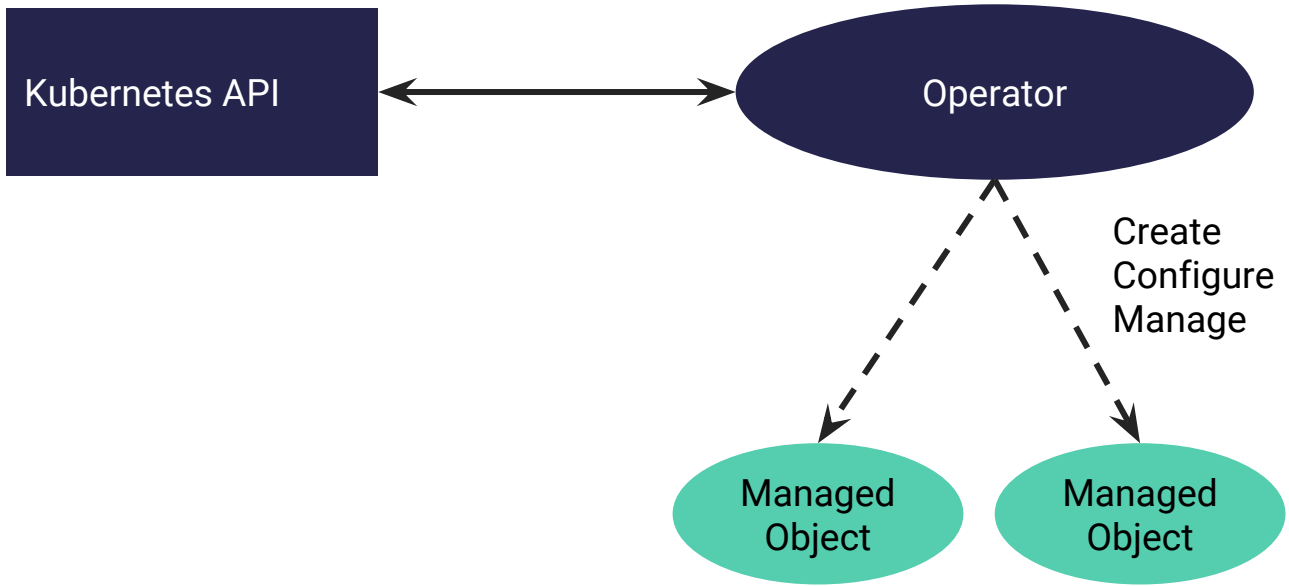
- create
- configure
- manage

instances of complex stateful applications

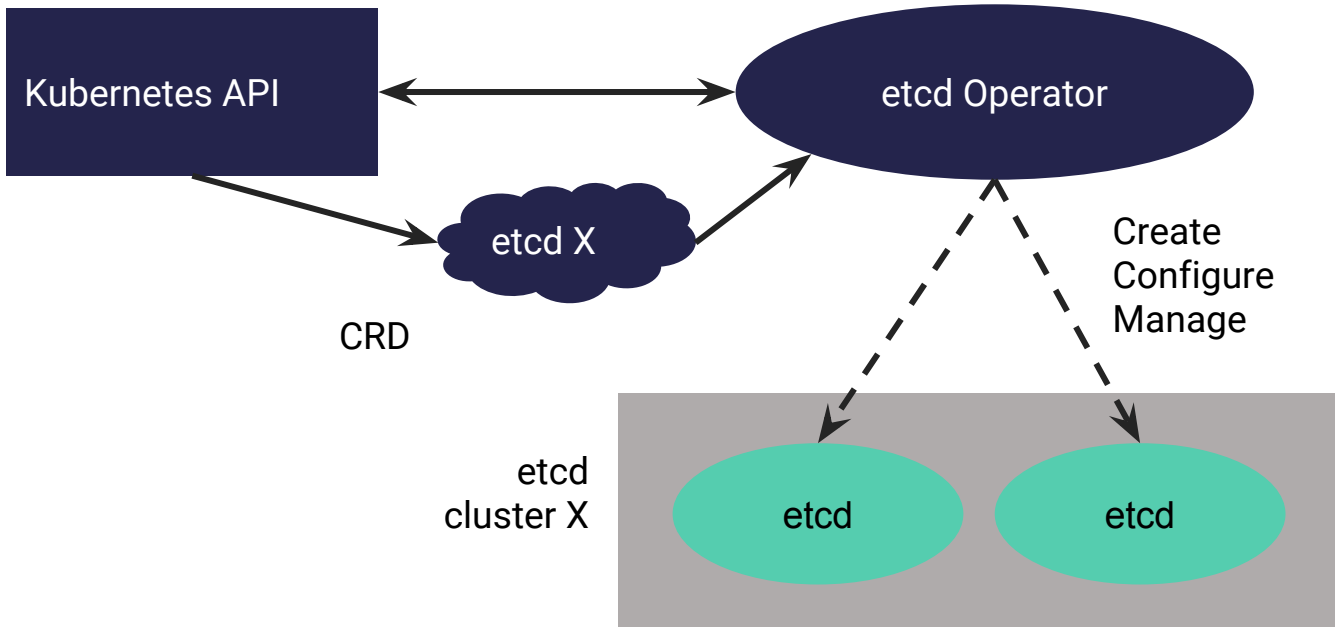
It builds upon the basic Kubernetes resource and controller concepts

Includes domain or application-specific knowledge to automate common tasks

Operator / Controller



Example: etcd Operator



Exercise/Demo CoreOS etcd Operator

1. Review and create the Operator (wait for startup)

2. Review and create a etcd-Cluster

Use `watch kubectl get po` to view cluster instantiation. Services?

3. Put some data in it

```
$ kubectl run --rm -i --tty fun --image quay.io/coreos/etcd --restart=Never -- /bin/sh
/ # export ETCDCTL_API=3
/ # etcdctl --endpoints http://example-etcd-cluster-client.etcd-operator:2379 put foo bar
OK
/ # etcdctl --endpoints http://example-etcd-cluster-client.etcd-operator:2379 get foo
foo
bar
(ctrl-D to exit)
```

4. Delete Cluster my-etcd-cluster