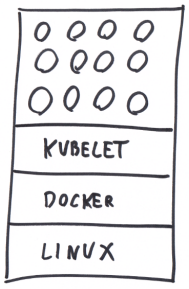
My Code is Containerized. Now What?

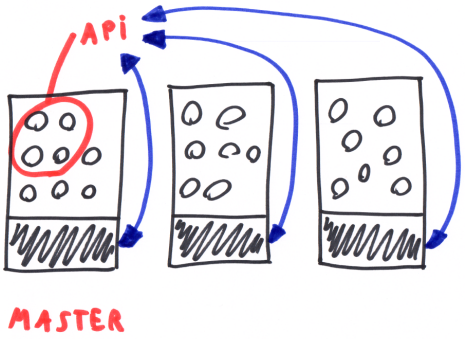
François Deppierraz - francois@ctrlaltdel.ch

September 14, 2018

1 What does a Kubernetes cluster looks like?



Node



Cluster

2 Kubernetes @ FIXME

https://fixme.ch/wiki/Kubernetes

Single-node cluster running an ancient version (1.8.0)
Deployed with kubeadm on a Debian stretch machine
Docker version 17.12.0-ce

Kubernetes Objects

```
In [2]: kubectl api-resources -o name | cut -d. -f1 | pr -2 -t
```

```
bindings                                tokenreviews
componentstatuses                        localsubjectaccessreviews
configmaps                               selfsubjectaccessreviews
endpoints                                selfsubjectrulesreviews
events                                        subjectaccessreviews
limitranges                              horizontalpodautoscalers
namespaces                               cronjobs
nodes                                       jobs
persistentvolumeclaims                       certificatesigningrequests
persistentvolumes                        events
pods                                        daemonsets
podtemplates                                deployments
replicationcontrollers                       ingresses
resourcequotas                               networkpolicies
secrets                                       podsecuritypolicies
serviceaccounts                               replicasets
services                                 networkpolicies
mutatingwebhookconfigurations                poddisruptionbudgets
validatingwebhookconfigurations               podsecuritypolicies
customresourcedefinitions                clusterrolebindings
apiservices                              clusterroles
controllerrevisions                      rolebindings
daemonsets                               roles
deployments                              priorityclasses
replicasets                              storageclasses
statefulsets                              volumeattachments
```

```
In [3]: kubectl api-resources -o name | wc -l
```

```
52
```

Namespaces

A single cluster can be shared between multiple users, projects or entities.

Namespaces is the basic construct behind multi-tenancy.

3 Stage 0: Prepare your tools

3.1 kubectl

- "As one of the author's, I say kyoob-cuddle. It's the laziest pronunciation. My alt favorite is kooby-cuttle"

- "Cube Control I guess"
- "kube see tee el"
- "I did an alias k=kubectl so i don't prononce it anymore :)"

Source: https://www.reddit.com/r/kubernetes/comments/5qthoc/how_should_i_pronounce_kubectl/

3.1.1 kubectl is the default (low-level) CLI client for the Kubernetes APIs

3.2 kubectl configuration

Config file is usually located in ~/.kube/config. In simple case, it was copied from /etc/kubernetes/admin.conf on the master node.

```
In [4]: kubectl config get-contexts

CURRENT   NAME            CLUSTER   AUTHINFO   NAMESPACE
*         admin@metafoo   metafoo   admin      demo


In [5]: kubectl cluster-info

Kubernetes master is running at https://62.220.135.205:6443
KubeDNS is running at https://62.220.135.205:6443/api/v1/namespaces/kube-system/services/kube-

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

3.3 A bunch of useful kubectl commands

3.3.1 kubectl get

Returns a list of Kubernetes resources.

```
In [6]: kubectl get nodes

NAME      STATUS    ROLES     AGE       VERSION
k8s       Ready     master    132d      v1.11.0


In [7]: kubectl get namespaces

NAME               STATUS    AGE
default            Active    132d
demo               Active    8d
kube-public        Active    132d
kube-system        Active    132d
metallb-system     Active    131d
power-monitoring   Active    132d
```

Pods

A pod is a **collection** of **containers** running on the same **node** sharing the same **kernel namespace** (semaphores, shared memory) and same **network namespace** (localhost).

```
In [8]: kubectl get pod mypod
```

```
NAME      READY     STATUS    RESTARTS   AGE
mypod     1/1       Running   0          8s
```

```
In [9]: kubectl describe pod/mypod | grep -m1 -A13 Containers
```

```
Containers:
  busybox:
    Container ID:  docker://e85d59f3272f6c6d1fcbb637f1063d4456c5c91a3a3a570128ca614adac1a361
    Image:         busybox
    Image ID:      docker-pullable://busybox@sha256:cb63aa0641a885f54de20f61d152187419e8f6b159
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
      -c
      while :; do date; sleep 300; done
    State:          Running
      Started:      Thu, 13 Sep 2018 23:50:07 +0200
    Ready:          True
```

A **pod** is reachable on a single IP address inside the cluster network (think virtual machine).

```
In [10]: kubectl describe pod/mypod | grep -m1 IP
```

```
IP:           10.32.0.18
```

A **pod** can use volumes for persistent data.

```
In [11]: kubectl describe pod/mypod | grep -m1 Volume -A4
```

```
Volumes:
  default-token-wfrcr:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-wfrcr
    Optional:    false
```

A **pod** is scheduled to run on a single node.

```
In [12]: kubectl describe pod/mypod | grep -m1 Node
```

```
Node:         k8s/62.220.135.205
```

3.3.2 kubectl delete

Delete resources by filenames, stdin, resources and names, or by resources and label selector.

```
In [13]: kubectl delete pod/mypod

pod "mypod" deleted
```

3.3.3 kubectl create

Create a resource from a file or from stdin.

```
In [14]: cat mypod.yaml

apiVersion: v1
kind: Pod
metadata:
  name: mypod
  namespace: demo
spec:
  containers:
  - image: busybox
    name: busybox
    command: ['/bin/sh', '-c', 'while :; do date; sleep 300; done']


In [15]: kubectl create -f mypod.yaml

pod/mypod created
```

3.3.4 kubectl describe

Show details of a specific resource or group of resources.

```
In [16]: kubectl describe pod/mypod

Name:         mypod
Namespace:    demo
Node:         k8s/62.220.135.205
Start Time:   Thu, 13 Sep 2018 23:50:53 +0200
Labels:       <none>
Annotations:  <none>
Status:       Pending
IP:
Containers:
  busybox:
    Container ID:
    Image:         busybox
```

```
    Image ID:
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
      -c
      while :; do date; sleep 300; done
    State:          Waiting
      Reason:       ContainerCreating
    Ready:          False
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-wfrcr (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             False
  ContainersReady   False
  PodScheduled      True
Volumes:
  default-token-wfrcr:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-wfrcr
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  0s    default-scheduler  Successfully assigned demo/mypod to k8s
```

4 Stage 1: Deploy a basic nginx webserver

4.1 Deploy nginx

Create a **deployment** with a 2 **pods** which contains a single **container** based on the *nginx* image from docker hub.

```
In [17]: kubectl run webserver --image=nginx --replicas 2

deployment.apps/webserver created


In [18]: kubectl rollout status deployment/webserver
```

```
Waiting for deployment "webserver" rollout to finish: 0 of 2 updated replicas are available...
Waiting for deployment "webserver" rollout to finish: 1 of 2 updated replicas are available...
deployment "webserver" successfully rolled out
```

```
In [19]: kubectl get deployment,pods -l run=webserver
```

```
NAME                                DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/webserver   2         2         2            2           7s

NAME                                READY     STATUS    RESTARTS   AGE
pod/webserver-787d79b644-2p6wp   1/1       Running   0          7s
pod/webserver-787d79b644-t6t79   1/1       Running   0          7s
```

Labels

Almost every object in Kubernetes can get **labels** assigned to it.

Labels is the main grouping mechanism used in Kubernetes.

For example:

```
app=blah
env=prod, env=test, env=qa
project=foobar
```

Selectors

A **selector** is a label query, for example:

```
env = prod
app != blah
```

```
In [20]: kubectl get pods -l run=webserver
```

```
NAME                          READY     STATUS    RESTARTS   AGE
webserver-787d79b644-2p6wp   1/1       Running   0          8s
webserver-787d79b644-t6t79   1/1       Running   0          8s
```

Deployments

```
In [21]: kubectl get deployment -l run=webserver
```

```
NAME        DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
webserver   2         2         2            2           8s
```

```
In [22]: kubectl describe deployment/webserver
```

```
Name:                   webserver
Namespace:              demo
CreationTimestamp:      Thu, 13 Sep 2018 23:50:54 +0200
Labels:                 run=webserver
Annotations:            deployment.kubernetes.io/revision=1
Selector:               run=webserver
Replicas:               2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  run=webserver
  Containers:
   webserver:
    Image:        nginx
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   webserver-787d79b644 (2/2 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  6s    deployment-controller  Scaled up replica set webserver-787d
```

4.2 Expose our nginx to the Internet

Expose our **deployment** to the external world using a **service** of type LoadBalancer.

```
In [23]: kubectl expose deployment/webserver --port=80 --target-port=80 --type=LoadBalancer

service/webserver exposed
```

```
In [24]: kubectl describe service/webserver

Name:                     webserver
Namespace:                demo
Labels:                   run=webserver
Annotations:              <none>
Selector:                 run=webserver
```

```
Type:                     LoadBalancer
IP:                       10.103.111.186
LoadBalancer Ingress:     62.220.135.219
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30218/TCP
Endpoints:                10.32.0.19:80,10.32.0.20:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:
  Type    Reason       Age        From                Message
  ----    ------       ----       ----                -------
  Normal  IPAllocated  <invalid>  metallb-controller  Assigned IP "62.220.135.219"
```

```
In [25]: URL=http://$(kubectl get service/webserver -o=jsonpath='{.status.loadBalancer.ingress
         echo $URL
```

```
http://62.220.135.219/
```

```
In [26]: curl -s $URL | grep h1
```

```
<h1>Welcome to nginx!</h1>
```

5 Stage 2: Updates and Roll-backs

5.1 Switch to Apache without downtime

Update our existing **deployment** to use the *httpd* image from docker hub instead of *nginx*.

```
In [27]: kubectl set image deployment/webserver webserver=httpd
```

```
deployment.extensions/webserver image updated
```

```
In [28]: kubectl rollout status deployment/webserver
```

```
Waiting for deployment "webserver" rollout to finish: 1 out of 2 new replicas have been update
Waiting for deployment "webserver" rollout to finish: 1 out of 2 new replicas have been update
Waiting for deployment "webserver" rollout to finish: 1 old replicas are pending termination..
Waiting for deployment "webserver" rollout to finish: 1 old replicas are pending termination..
Waiting for deployment "webserver" rollout to finish: 1 old replicas are pending termination..
deployment "webserver" successfully rolled out
```

```
In [29]: curl -s $URL | grep h1
```

```
<html><body><h1>It works!</h1></body></html>
```

5.2 Roll-back

```
In [30]: kubectl rollout undo deployment/webserver

deployment.extensions/webserver


In [31]: kubectl rollout status deployment/webserver

Waiting for deployment "webserver" rollout to finish: 1 out of 2 new replicas have been updated
Waiting for deployment "webserver" rollout to finish: 1 out of 2 new replicas have been updated
Waiting for deployment "webserver" rollout to finish: 1 old replicas are pending termination..
Waiting for deployment "webserver" rollout to finish: 1 old replicas are pending termination..
Waiting for deployment "webserver" rollout to finish: 1 old replicas are pending termination..
deployment "webserver" successfully rolled out


In [32]: curl -s $URL | grep h1

<h1>Welcome to nginx!</h1>
```

6 Stage 3: Fat finger and Recovery

6.1 Update prod in a hurry

Update our existing **deployment** to use an inexistant image and see how it goes.

```
In [33]: kubectl set image deployment/webserver webserver=hjfdsfdsjfhds

deployment.extensions/webserver image updated


In [34]: timeout 5 kubectl rollout status deployment/webserver || echo failure

Waiting for deployment "webserver" rollout to finish: 1 out of 2 new replicas have been updated
failure


In [35]: kubectl get pods -l run=webserver

NAME                         READY     STATUS         RESTARTS   AGE
webserver-5bb89f7759-szxjw   0/1       ErrImagePull   0          5s
webserver-787d79b644-ltwr4   1/1       Running        0          15s
webserver-787d79b644-mtd6d   1/1       Running        0          10s
webserver-7dd8ddf4d-pfz47    0/1       Terminating    0          24s


In [36]: curl -s $URL | grep h1
```

```
<h1>Welcome to nginx!</h1>
```

```
In [37]: kubectl rollout undo deployment/webserver

deployment.extensions/webserver
```

7 Stage 4: Configuration

ConfigMaps

A **ConfigMap** is an object which usually contains configuration parameters.

The size of a **ConfigMap** is limited to 1 MB.

A **ConfigMap** can be exposed to **Pods** using environment variables or regular files (via volumeMounts).

```
In [38]: kubectl create configmap httpdocs --from-file=index.html

configmap/httpdocs created
```

```
In [39]: kubectl delete deployment/webserver

deployment.extensions "webserver" deleted
```

```
In [40]: cat webserver-with-cm.deployment.yaml

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    run: webserver
  name: webserver
  namespace: demo
spec:
  replicas: 2
  selector:
    matchLabels:
      run: webserver
  template:
    metadata:
      labels:
        run: webserver
    spec:
      containers:
      - image: nginx
```

```
        name: webserver
        volumeMounts:
        - mountPath: /usr/share/nginx/html
          name: html
      volumes:
      - configMap:
          name: httpdocs
        name: html
```

```
In [41]: kubectl create -f webserver-with-cm.deployment.yaml
```

```
deployment.extensions/webserver created
```

```
In [42]: curl -s $URL | grep h1
```

```
<h1>Hello World!</h1>
```

8 Stage -1: Troubleshooting

8.1 Getting logs from pods

```
In [43]: kubectl logs mypod
```

```
Thu Sep 13 21:50:57 UTC 2018
```

8.2 Executing commands inside pods

```
In [44]: kubectl exec mypod -- /bin/ps ax
```

```
PID   USER     TIME  COMMAND
    1 root      0:00 /bin/sh -c while :; do date; sleep 300; done
    9 root      0:00 sleep 300
   10 root      0:00 /bin/ps ax
```

8.3 Get cluster events

```
In [45]: kubectl get events | tail -10
```

```
9s          9s          1         webserver.1554141dbab9d9b4                    Deployment
9s          9s          1         webserver-5cd47d669-87jpt.1554141dbe718b93    Pod
7s          7s          1         webserver-5cd47d669-87jpt.1554141e3601740e    Pod          s
6s          6s          1         webserver-5cd47d669-4sscl.1554141e4287b8e4    Pod          s
5s          5s          1         webserver-5cd47d669-87jpt.1554141eb45eebf9    Pod          s
4s          4s          1         webserver-5cd47d669-87jpt.1554141ebbf4fae8    Pod          s
```

```
4s          4s          1         webserver-5cd47d669-87jpt.1554141ec9c89b81    Pod          s
3s          3s          1         webserver-5cd47d669-4sscl.1554141f29c6017b    Pod          s
2s          2s          1         webserver-5cd47d669-4sscl.1554141f332a613b    Pod          s
2s          2s          1         webserver-5cd47d669-4sscl.1554141f441281ec    Pod          s
```

Recap: Kubernetes Objects

- Nodes
- Namespaces
- Pods
- Deployments
- Services
- ConfigMaps

8.4 Where should I go from there?

8.4.1 How could pods keep data when they get deleted during updates?

8.4.2 Use PersistentVolumeClaims

8.4.3 A service of type LoadBalancer works at layer 4 (TCP), I need layer 7 routing (HTTP)!

8.4.4 Use Ingress

8.4.5 How does self healing actually works?

8.4.6 It works using control loops (just like elevators, quadcopters and power plants!)