Kubernetes pour les dev

version: 2022-10-28

-- Cyril zarak Duval, root CRI/ACU 2020

Présentation

Quelques généralités

Pourquoi écouter cette présentation ?

- ➔ Développer c'est bien, sans production ça sert à rien
- ➔ Notions de DevOps
- ➔ Connaître les bons outils = gain de temps
- ➔ Sujet difficile à aborder
- ➔ QCM à la fin

Qu'est-ce que Kubernetes ?

- Projet de Google
  - Adaptation de Borg en open source
- Sorti en 2015
- Orchestrateur de conteneurs
- Plateforme de déploiement, mise en production
- Gestion de la montée en charge
- Haute disponibilité

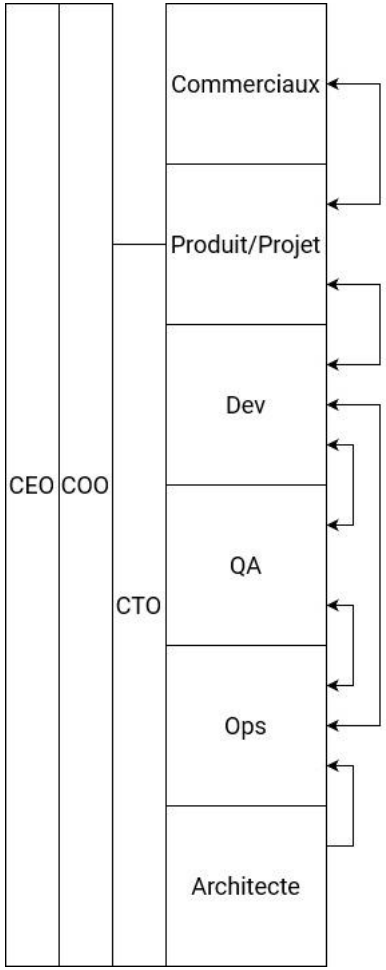
Quels problèmes cherche à résoudre Kubernetes ?

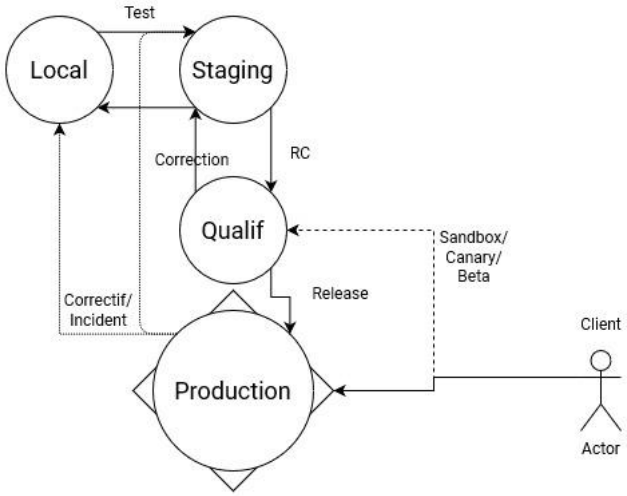
Sans surprise, ça ne sert pas à rien

Après le dev, l'ops

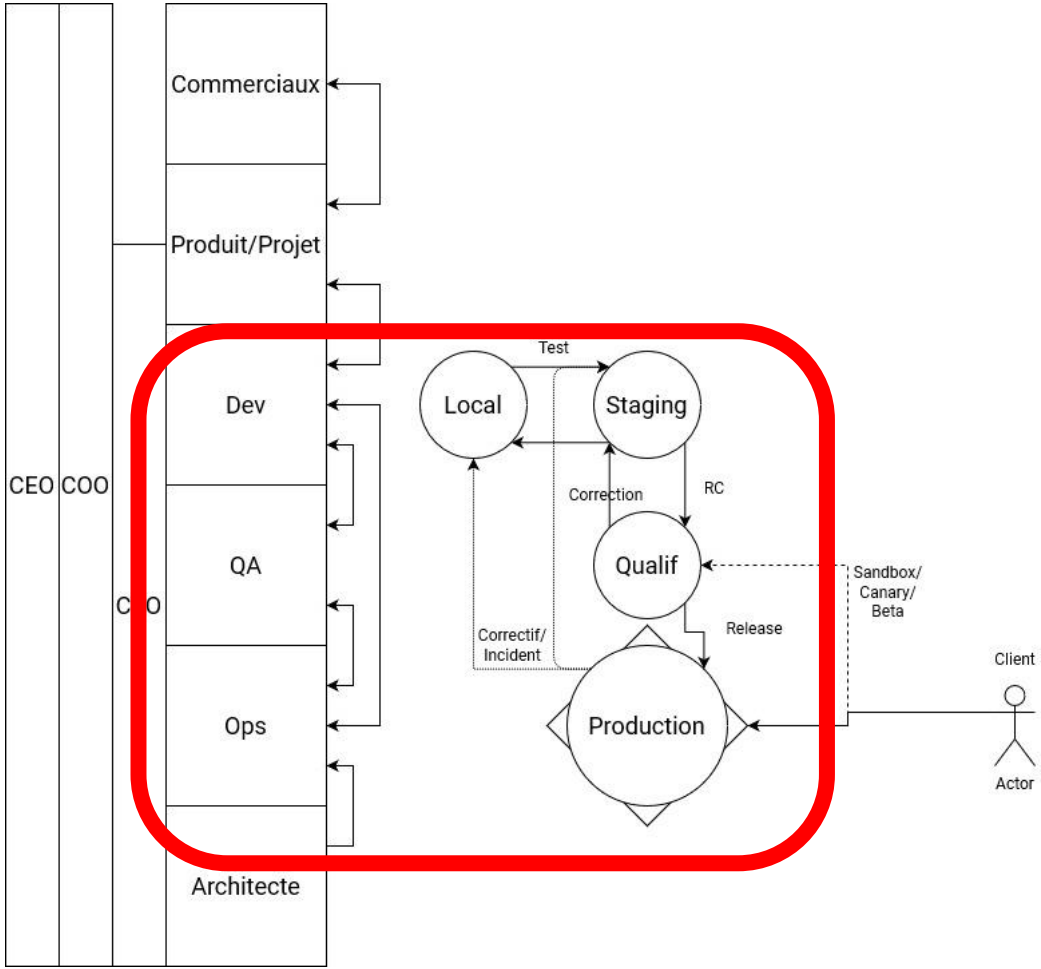
- Cycle de vie du développement en entreprise
  - Exemple simplifié
  - PME
  - ~SaaS

Après le dev, l'ops





Le devops


Commerciaux
Produit/Projet
Dev
QA
Ops
Architecte
CEO
COO
C O
Test
Local
Staging
Correction
RC
Qualif
Sandbox/
Canary/
Beta
Release
Correctif/
Incident
Production
Client
Actor

Le DevOps

- Le dev travaille sur son laptop
  - Son OS
  - Ses bibliothèques
  - Son environnement
  - Sa stack réseau
  - ...

Le DevOps

- Le dev travaille sur son laptop
  - Son OS
  - Ses bibliothèques
  - Son environnement
  - Sa stack réseau
  - …
- Envoyer sur la prod peut poser des problèmes

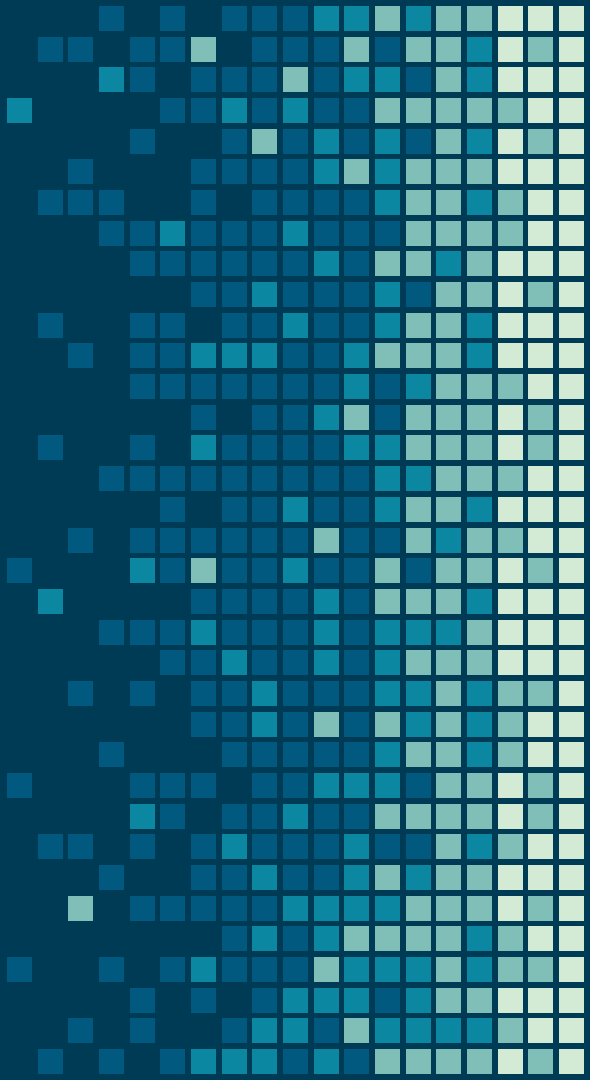
Le DevOps

- Le dev travaille sur son laptop
  - Son OS
  - Ses bibliothèques
  - Son environnement
  - Sa stack réseau
  - …
- Envoyer sur la prod peut poser des problèmes
- On va pas envoyer son laptop sur la prod

Le DevOps - docker

➔ Le dev travaille sur son laptop avec docker

Démystification de Docker par l'exemple



Le DevOps - docker

- ➔ Le dev travaille sur son laptop avec docker
- ➔ Image docker :
  - ◆ Contrôle de l'OS
    - `FROM debian:11`

Le DevOps - docker

- ➔ Le dev travaille sur son laptop avec docker
- ➔ Image docker :
  - ◆ Contrôle de l'OS
  - ◆ Contrôle du réseau

Le DevOps - docker

- ➔ Le dev travaille sur son laptop avec docker
- ➔ Image docker :
  - ◆ Contrôle de l'OS
  - ◆ Contrôle du réseau
  - ◆ Contrôle des dépendances
    - ●
```
FROM python:3-alpine
RUN pip install ansible
```

Le DevOps - docker

- ➔ Le dev travaille sur son laptop avec docker
- ➔ Image docker :
  - ◆ Contrôle de l'OS
  - ◆ Contrôle du réseau
  - ◆ Contrôle des dépendances
  - ◆ Contrôle des versions
    - ```
      FROM python:3-alpine
      RUN pip install ansible==2.10
      ```
      ```

Le DevOps - docker

- ➔ Le dev travaille sur son laptop avec docker
- ➔ Image docker :
  - ◆ Contrôle de l'OS
  - ◆ Contrôle du réseau
  - ◆ Contrôle des dépendances
  - ◆ Contrôle des versions
- ➔ Livrable = lien vers le docker registry

Le DevOps - docker

- Le dev travaille sur son laptop avec docker
- Image docker :
  - Contrôle de l'OS
  - Contrôle du réseau
  - Contrôle des dépendances
  - Contrôle des versions
- Livrable = lien vers le docker registry
  - 2-3 broutilles (variables env, ports, etc)

Le DevOps - docker

➔ Image docker = application, ~binaire

Le DevOps - docker

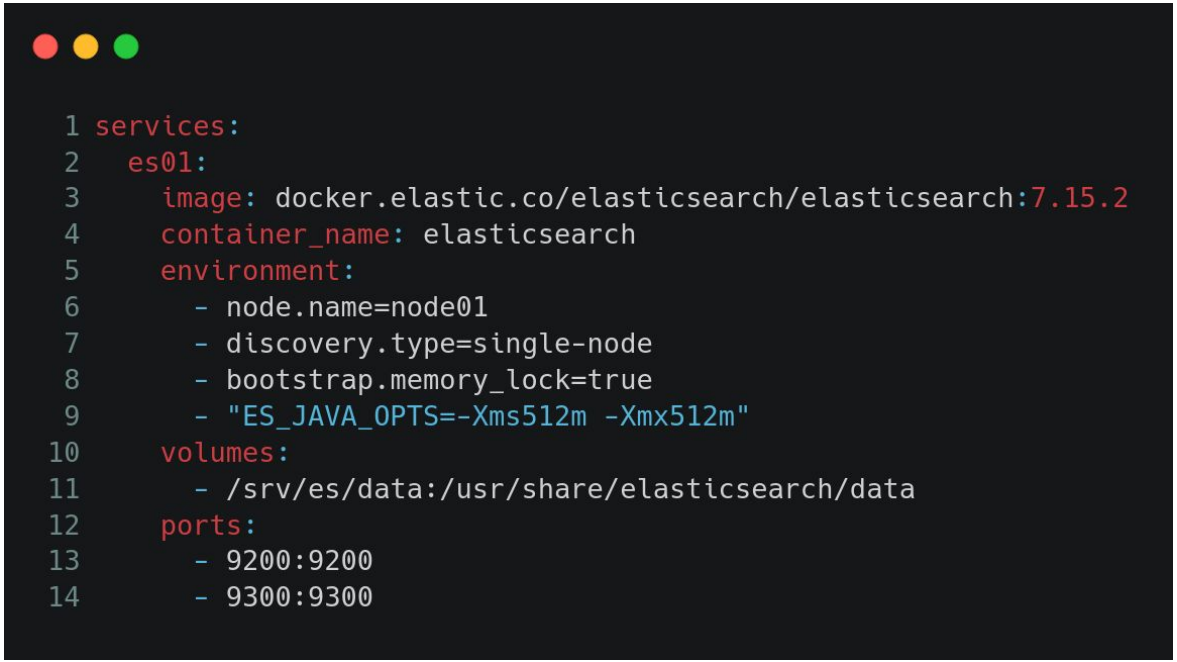
- ➔ Image docker = application, ~binaire
- ➔
```
docker run \
    -p 9200:9200 \
    -p 9300:9300 \
    -e "discovery.type=single-node" \
    -v /srv/es/data:/usr/share/elasticsearch/data \
    docker.elastic.co/elasticsearch/elasticsearch:7.15.2
```

Le DevOps - docker

- ➔ Image docker = application, ~binaire
- ➔
```
docker run \
    -p 9200:9200 \
    -p 9300:9300 \
    -e "discovery.type=single-node" \
    -v /srv/es/data:/usr/share/elasticsearch/data \
    docker.elastic.co/elasticsearch/elasticsearch:7.15.2
```
- ➔ Manque de tooling pour le déploiement, fichier de conf

Le DevOps - docker-compose

- ➔ docker-compose.yml = fichier de conf déclaratif de la CLI docker
- ➔



```yaml
services:
  es01:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.15.2
    container_name: elasticsearch
    environment:
      - node.name=node01
      - discovery.type=single-node
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    volumes:
      - /srv/es/data:/usr/share/elasticsearch/data
    ports:
      - 9200:9200
      - 9300:9300
```

Le DevOps - docker-compose

- ➔ docker-compose.yml = fichier de conf déclaratif de la CLI docker
- ➔ Pas de notions de :
  - Scaling horizontal
  - Haute disponibilité
  - Répartition de charge
  - Multi-noeuds
  - Déploiement (Rolling Release, rollback, ...)
  - ...

Le DevOps - Kubernetes

- Kubernetes :
  - Multi-noeuds
  - Déclaratif
  - Notions de services, secrets, configuration, réseau, …
- Production-ready

Le DevOps - Kubernetes

- ➔ Pour le dev :
  - Dev avec docker en local
    - Voire k3s/minikube/KinD
  - Intégration de l'image docker dans les ressources k8s
  - Environment staging vs production similaires :
    - Même ressources k8s
    - Différence scalabilité, ressources hardware, …
  - Workflow avec le tooling de k8s

Le DevOps - Kubernetes

- Ressources k8s déclaratives :
  - Mise en commun dev et ops

Le DevOps - Kubernetes

- ➔ Ressources k8s déclaratives :
  - Mise en commun dev et ops
  - Templating :

Le DevOps - Kubernetes

- ➔ Ressources k8s déclaratives :
  - Mise en commun dev et ops
  - Templating :
    - Templates identiques staging/production

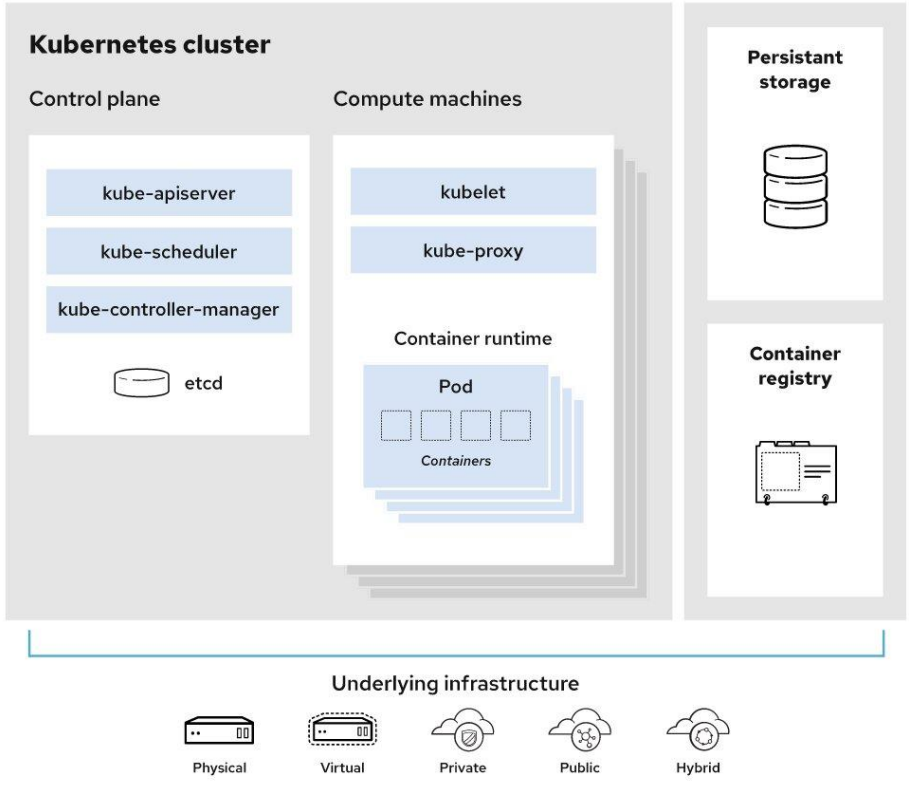
Le DevOps - Kubernetes

- ➔ Ressources k8s déclaratives :
  - ◆ Mise en commun dev et ops
  - ◆ Templating :
    - Templates identiques staging/production
    - Valeurs qui changent

A quoi ressemble Kubernetes ?

Une chimère à plusieurs visages

Kubernetes 101 – Architecture



Kubernetes 101 - Les ressources

- Interaction avec l'API de Kubernetes via des ressources
- Interaction avec k8s sur les ressources
  - Création, suppression, modification, ...
- Composants de k8s = ressources
  - nodes
  - namespaces
  - persistentvolumes
  - configmaps
  - pods
  - Pas de "container"

Kubernetes 101 - Le pod

- Plus petite unité d'exécution de Kubernetes
- Assimilable à un conteneur docker
  - 90% des cas pod = conteneur docker
- Spécificités :
  - Potentiellement plusieurs containers par pod
    - Principal + sidecar
    - Des init containers (migration DB)
  - Même network namespaces(7)
  - Même shared volumes

Kubernetes 101 - kubectl

```
$ kubectl get nodes
NAME                          STATUS   ROLES          AGE    VERSION
km01-d01-rdb.dev.dblc.io      Ready    controlplane   609d   v1.20.5
km02-d01-rdb.dev.dblc.io      Ready    controlplane   609d   v1.20.5
kw01-d01-rdb.dev.dblc.io      Ready    worker         609d   v1.20.5
kw02-d01-rdb.dev.dblc.io      Ready    worker         609d   v1.20.5
kw03-d01-rdb.dev.dblc.io      Ready    worker         609d   v1.20.5
kw04-d01-rdb.dev.dblc.io      Ready    worker         609d   v1.20.5
kw05-d01-rdb.dev.dblc.io      Ready    worker         498d   v1.20.5
```

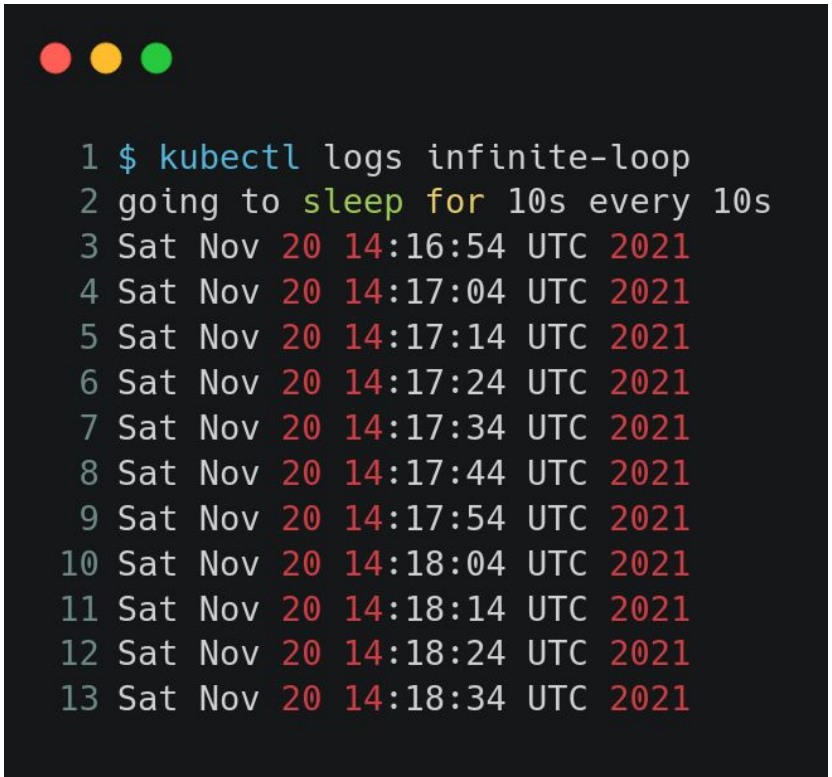
Kubernetes 101 - kubectl

```
$ k get node -o wide
NAME                      STATUS   ROLES          AGE    VERSION   INTERNAL-IP    EXTERNAL-IP   OS-IMAGE                    KERNEL-VERSION   CONTAINER-RUNTIME
km01-d01-rdb.dev.dblc.io  Ready    controlplane   609d   v1.20.5   10.3.196.50    <none>        Debian GNU/Linux 9 (stretch)  4.9.0-7-amd64    docker://19.3.7
km02-d01-rdb.dev.dblc.io  Ready    controlplane   609d   v1.20.5   10.3.196.51    <none>        Debian GNU/Linux 9 (stretch)  4.9.0-7-amd64    docker://19.3.7
kw01-d01-rdb.dev.dblc.io  Ready    worker         609d   v1.20.5   10.3.196.52    <none>        Debian GNU/Linux 9 (stretch)  4.9.0-7-amd64    docker://19.3.7
kw02-d01-rdb.dev.dblc.io  Ready    worker         609d   v1.20.5   10.3.196.53    <none>        Debian GNU/Linux 9 (stretch)  4.9.0-7-amd64    docker://19.3.7
kw03-d01-rdb.dev.dblc.io  Ready    worker         609d   v1.20.5   10.3.196.54    <none>        Debian GNU/Linux 9 (stretch)  4.9.0-7-amd64    docker://19.3.7
kw04-d01-rdb.dev.dblc.io  Ready    worker         609d   v1.20.5   10.3.196.55    <none>        Debian GNU/Linux 9 (stretch)  4.9.0-7-amd64    docker://19.3.7
kw05-d01-rdb.dev.dblc.io  Ready    worker         498d   v1.20.5   10.3.196.56    <none>        Debian GNU/Linux 9 (stretch)  4.9.0-7-amd64    docker://19.3.7
```

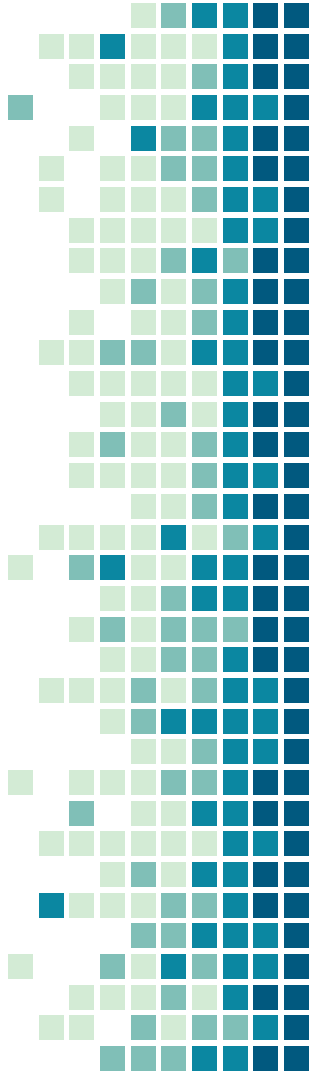
Kubernetes 101 - kubectl

```
$ kubectl get po
NAME                READY   STATUS    RESTARTS   AGE
infinite-2loops     2/2     Running   0          4m8s
infinite-loop       1/1     Running   0          4m3s
```

Kubernetes 101 - kubectl



```
$ kubectl describe po infinite-loop
Name:         infinite-loop
Namespace:    default
Priority:     0
Node:         chlorine/192.168.1.18
Start Time:   Sat, 20 Nov 2021 15:16:43 +0100
Labels:       <none>
Annotations:  <none>
Status:       Running
IP:           10.42.0.50
IPs:
  IP:  10.42.0.50
Containers:
  container1:
    Container ID:   docker://ba89414b1d829dd6f54777a87af5b37fe1f0395c68779762d0b2ecbcd266b0da
    Image:          pause
    Image ID:       docker://sha256:d18c91061232135b43592c31f7fd1ca73e4afee1e616ca8795f1dc9fa8de8c79
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Sat, 20 Nov 2021 15:16:44 +0100
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-bqvk2 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
```



Kubernetes 101 - kubectl

```
$ kubectl exec -it infinite-loop -- bash
bash-5.1$ hostname
infinite-loop
bash-5.1$ ps aux
PID   USER     TIME  COMMAND
    1 root      0:00 bash -c echo going to sleep for 10s every 10s; while true; do sleep 10; date; done
  547 root      0:00 sleep 10
  548 root      0:00 bash
  555 root      0:00 ps aux
bash-5.1$ env
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_SERVICE_PORT=443
HOSTNAME=infinite-loop
PWD=/
HOME=/root
KUBERNETES_PORT_443_TCP=tcp://10.43.0.1:443
TERM=xterm
SHLVL=1
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_ADDR=10.43.0.1
KUBERNETES_SERVICE_HOST=10.43.0.1
KUBERNETES_PORT=tcp://10.43.0.1:443
KUBERNETES_PORT_443_TCP_PORT=443
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
_=/usr/bin/env
```

Kubernetes 101 - kubectl

```
$ kubectl get po
NAME                READY   STATUS    RESTARTS   AGE
infinite-loop       1/1     Running   0          44m
infinite-2loops     2/2     Running   0          44m
$ kubectl delete po infinite-loop
pod "infinite-loop" deleted
$ kubectl get po
NAME                READY   STATUS    RESTARTS   AGE
infinite-2loops     2/2     Running   0          45m
```

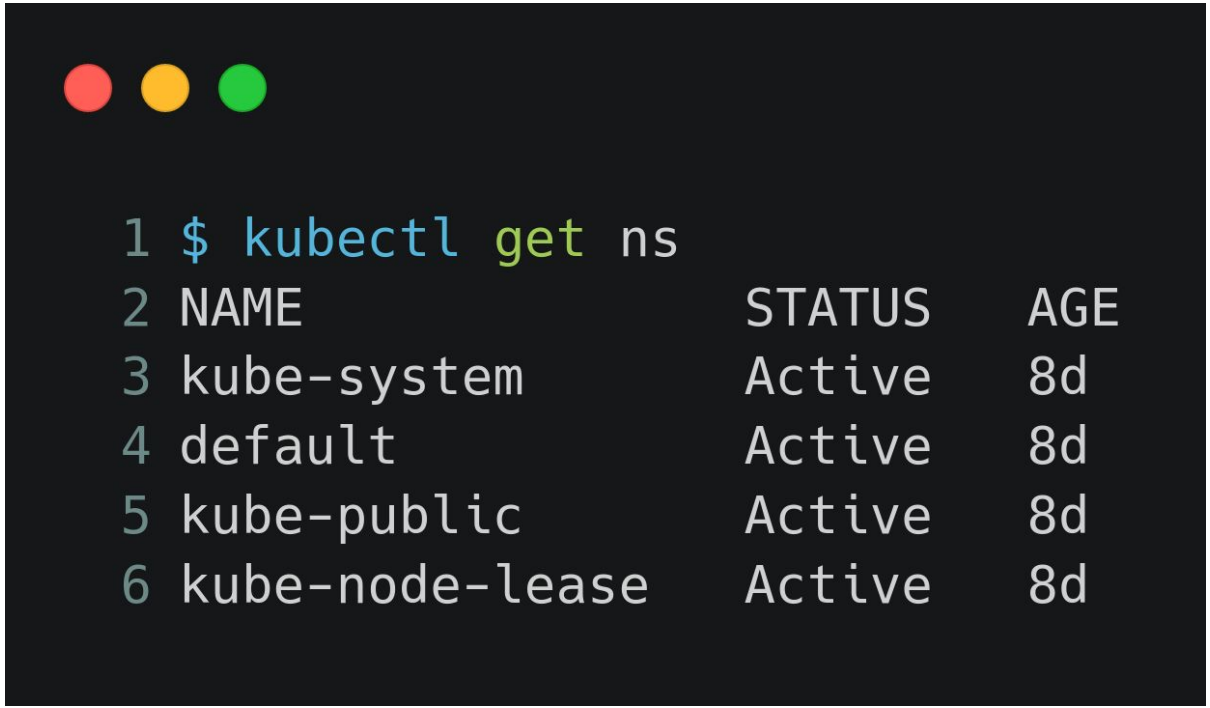
Kubernetes 101 - docker ps

```
$ docker ps --format 'table {{.Image}}\t{{.Command}}\t{{.RunningFor}}\t{{.Names}}'
IMAGE               COMMAND                  CREATED          NAMES
d18c91061232        "bash -c 'echo going…"   53 minutes ago   k8s_container2_infinite-2loops_default_8af0ba4c-e6c8-4831-9537-6192c896c72f_0
d18c91061232        "bash -c 'echo going…"   53 minutes ago   k8s_container1_infinite-2loops_default_8af0ba4c-e6c8-4831-9537-6192c896c72f_0
rancher/pause:3.1   "/pause"                 53 minutes ago   k8s_POD_infinite-2loops_default_8af0ba4c-e6c8-4831-9537-6192c896c72f_0
465db341a9e5        "entry"                  2 hours ago      k8s_lb-port-443_svclb-traefik-krzqs_kube-system_6bc33121-d8c8-459f-ba55-8529835ea77d_1
465db341a9e5        "entry"                  2 hours ago      k8s_lb-port-80_svclb-traefik-krzqs_kube-system_6bc33121-d8c8-459f-ba55-8529835ea77d_1
deaf4b1027ed        "/entrypoint.sh --gl…"   2 hours ago      k8s_traefik_traefik-97b44b794-jj86m_kube-system_bbef9cda-3ae8-41d9-b520-2f9cd318b811_1
rancher/pause:3.1   "/pause"                 2 hours ago      k8s_POD_svclb-traefik-krzqs_kube-system_6bc33121-d8c8-459f-ba55-8529835ea77d_2
3885a5b7f138        "/coredns -conf /etc…"   2 hours ago      k8s_coredns_coredns-7448499f4d-gpksg_kube-system_5202957b-d36f-44cc-a9b2-a5f941f8bfa8_1
rancher/pause:3.1   "/pause"                 2 hours ago      k8s_POD_traefik-97b44b794-jj86m_kube-system_bbef9cda-3ae8-41d9-b520-2f9cd318b811_1
rancher/pause:3.1   "/pause"                 2 hours ago      k8s_POD_coredns-7448499f4d-gpksg_kube-system_5202957b-d36f-44cc-a9b2-a5f941f8bfa8_2
9dd718864ce6        "/metrics-server"        2 hours ago      k8s_metrics-server_metrics-server-86cbb8457f-t54g4_kube-system_63f4afe1-3a25-47ad-a5d7-6cb9cefc4e90_2
148c19256271        "local-path-provisio…"   2 hours ago      k8s_local-path-provisioner_local-path-provisioner-5ff76fc89d-sbwvg_kube-system_8a0cb047-019b-459d-950a-
db33bd6e62b5_2
rancher/pause:3.1   "/pause"                 2 hours ago      k8s_POD_metrics-server-86cbb8457f-t54g4_kube-system_63f4afe1-3a25-47ad-a5d7-6cb9cefc4e90_2
rancher/pause:3.1   "/pause"                 2 hours ago      k8s_POD_local-path-provisioner-5ff76fc89d-sbwvg_kube-system_8a0cb047-019b-459d-950a-db33bd6e62b5_2
```

Kubernetes 101 - docker ps

```
$ kubectl get po --all-namespaces
NAMESPACE     NAME                                      READY   STATUS      RESTARTS   AGE
kube-system   helm-install-traefik-crd-7kcct            0/1     Completed   0          8d
kube-system   helm-install-traefik-cljb2                0/1     Completed   0          8d
kube-system   metrics-server-86cbb8457f-t54g4           1/1     Running     2          8d
kube-system   local-path-provisioner-5ff76fc89d-sbwvg   1/1     Running     2          8d
kube-system   svclb-traefik-krzqs                       2/2     Running     2          8d
kube-system   coredns-7448499f4d-gpksg                  1/1     Running     1          8d
kube-system   traefik-97b44b794-jj86m                   1/1     Running     1          8d
default       infinite-2loops                           2/2     Running     0          57m
```

Kubernetes 101 - namespaces



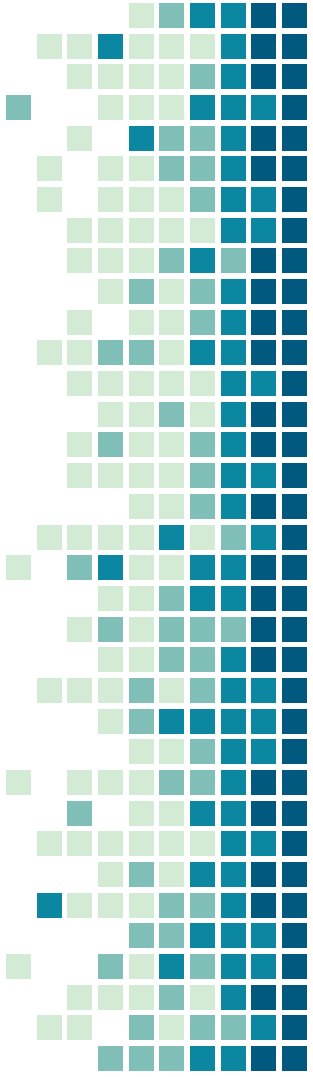
```
$ kubectl get ns
NAME                 STATUS   AGE
kube-system          Active   8d
default              Active   8d
kube-public          Active   8d
kube-node-lease      Active   8d
```

Comment créer des ressources ?

Créons des pods, et des créateurs de pods

Kubernetes

- ➔ Ressources k8s déclaratives
- ➔ YAML



Kubernetes

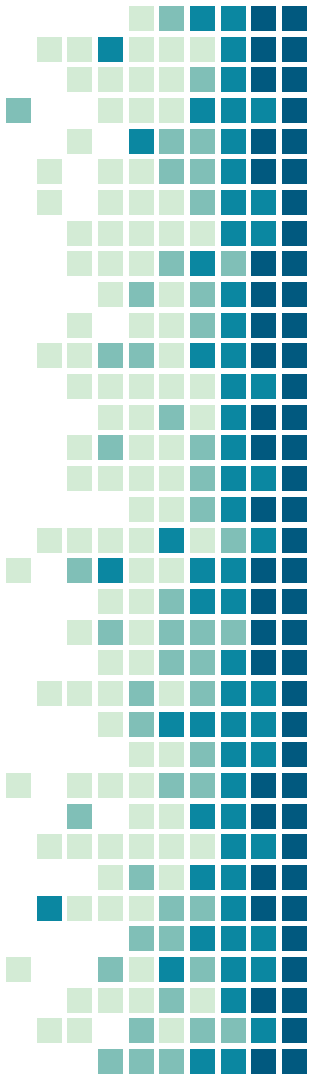
- ➔ Ressources k8s déclaratives
- ➔ YAML
- ➔ 3 informations obligatoires

Kubernetes

- Ressources k8s déclaratives
- YAML
- 3 informations obligatoires
  - apiVersion

Kubernetes

- Ressources k8s déclaratives
- YAML
- 3 informations obligatoires
  - apiVersion
    - v1
    - v1beta1
    - rbac.authorization.k8s.io/v1
    - k3s.cattle.io/v1



Kubernetes

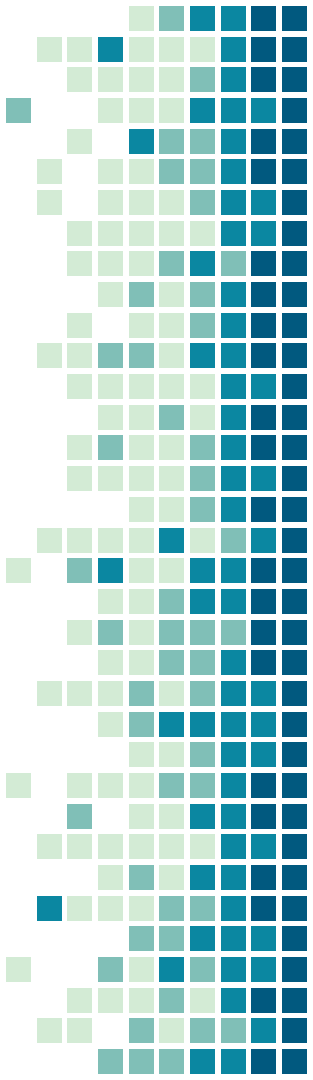
- Ressources k8s déclaratives
- YAML
- 3 informations obligatoires
  - apiVersion
  - kind

Kubernetes

- ➔ Ressources k8s déclaratives
- ➔ YAML
- ➔ 3 informations obligatoires
  - ◆ apiVersion
  - ◆ kind
  - ◆ metadata.name

Kubernetes

- ➔ Ressources k8s déclaratives
- ➔ YAML
- ➔ 3 informations obligatoires
  - ◆ apiVersion
  - ◆ kind
  - ◆ metadata.name
- ➔ 1 quasi-obligatoire



Kubernetes

- Ressources k8s déclaratives
- YAML
- 3 informations obligatoires
  - apiVersion
  - kind
  - metadata.name
- 1 quasi-obligatoire
  - spec

Kubernetes - kind: pod

➔ Créons un pod manuellement

Kubernetes - kind: pod

- Création manuelle
- Cas très basique
  - API complète
- Pas de scaling automatique
  - Pas de déploiement en HA
  - Pas de loadbalancing
  - ...

Kubernetes – kind: pod

➔ 2 containers

```
1 $ cat pod-2containers.yml
2 apiVersion: v1
3 kind: Pod
4 metadata:
5   name: infinite-2loops
6 spec:
7   containers:
8     - image: pause
9       imagePullPolicy: Never
10      name: container1
11    - image: pause
12      imagePullPolicy: Never
13      name: container2
```

Kubernetes - kind: pod

- ➔ 2 containers
- ➔ Besoin de préciser le container pour certaines opérations

```
$ cat pod-2containers.yml
apiVersion: v1
kind: Pod
metadata:
  name: infinite-2loops
spec:
  containers:
    - image: pause
      imagePullPolicy: Never
      name: container1
    - image: pause
      imagePullPolicy: Never
      name: container2
```

```
$ k logs infinite-2loops
error: a container name must be specified for pod infinite-2loops, choose one of: [container1 container2]
$ k logs -c container1 infinite-2loops
```

Kubernetes - kind: pod

➔ Prenons un exemple un peu plus complexe

```
┌─zarak@chlorine ~/misc/kubernetes/basic-service
└─$ cd ..
┌─zarak@chlorine ~/misc/kubernetes
└─$ ls
basic-refresh  basic-service  pause  pod-2containers.yml  pod-2services.yml  pod-simple.yml
┌─zarak@chlorine ~/misc/kubernetes
└─$ cat pod-2services.yml
apiVersion: v1
kind: Pod
metadata:
  name: basic-service
spec:
  volumes:
    - name: shared-workdir
      emptyDir: {}
  containers:
    - image: basic-refresh
      imagePullPolicy: Never
      name: refresh
      env:
        - name: WORKDIR
          value: /srv
      volumeMounts:
        - name: shared-workdir
          mountPath: /srv
    - image: basic-service
      imagePullPolicy: Never
      name: service
      env:
        - name: WORKDIR
          value: /mnt
      volumeMounts:
        - name: shared-workdir
          mountPath: /mnt
┌─zarak@chlorine ~/misc/kubernetes
└─$ k apply -f pod-2services.yml
pod/basic-service created
┌─zarak@chlorine ~/misc/kubernetes
└─$
```

Kubernetes - kind: pod

- 2 containers dans le pod
- Chacun leur propre FS
  - Volume précisé à côté des containers
  - Source = emptyDir, tmpfs vide
  - VolumeMounts dans chaque container pour préciser où le mount
- Ajout de variables d'environnement
- Beaucoup d'autres options
  - Lire la doc

On doit créer nos pods manuellement ?

Bien sûr que non, on est stylé, on fait du kube

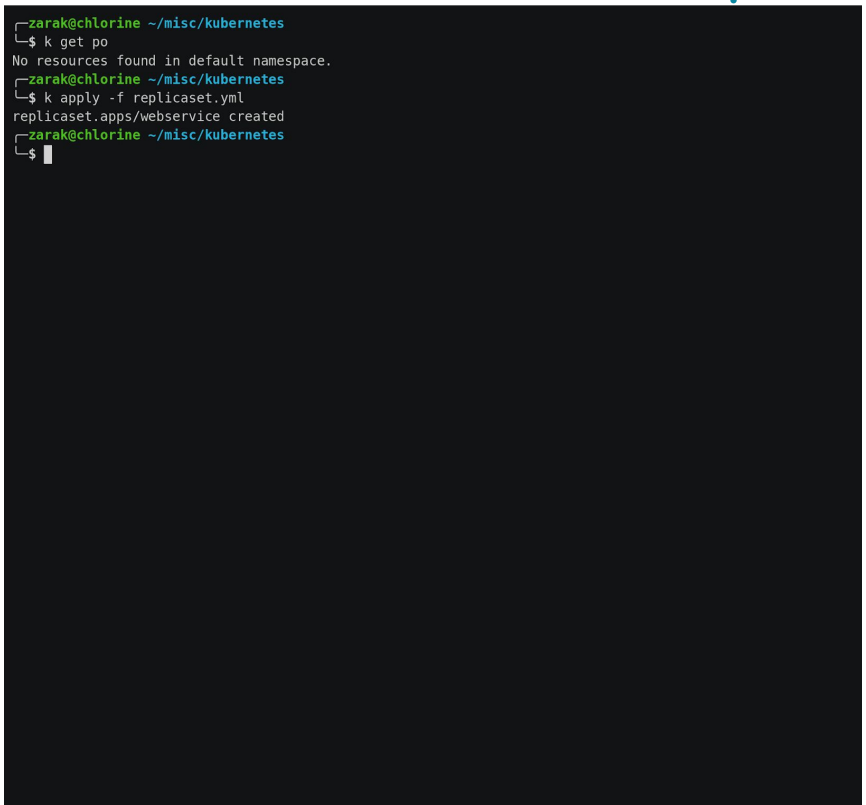
k8s – workload controllers

- Ressource haut-niveau de gestion des pods
- Précise quel template de pod on veut créer et gérer
- S'occupe de les créer et contrôler leur cycle de vie
- Plusieurs types, plusieurs utilisations :
  - ReplicaSet
  - Deployment
  - StatefulsSet
  - DaemonSet
  - Job/CronJob

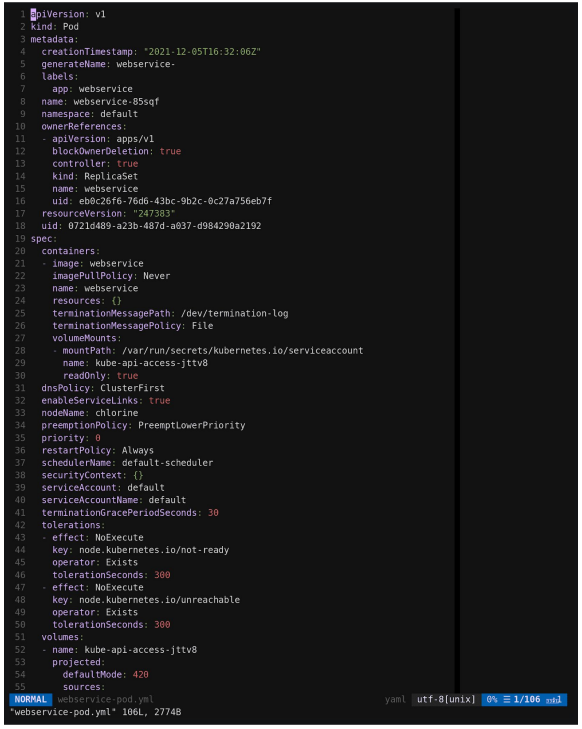
Kubernetes – kind: ReplicaSet

- Prends :
  - Un template de Pod
  - Un sélecteur de Pods
  - Un nombre x de répliques (replica)
- Créé à partir du template des Pods jusqu'à atteindre x répliques
- Nombre n de Pods actuels déterminé par le sélecteur
- Si n > x, termine des pods
- Si x > n, créé des pods

Kubernetes – kind: ReplicaSet



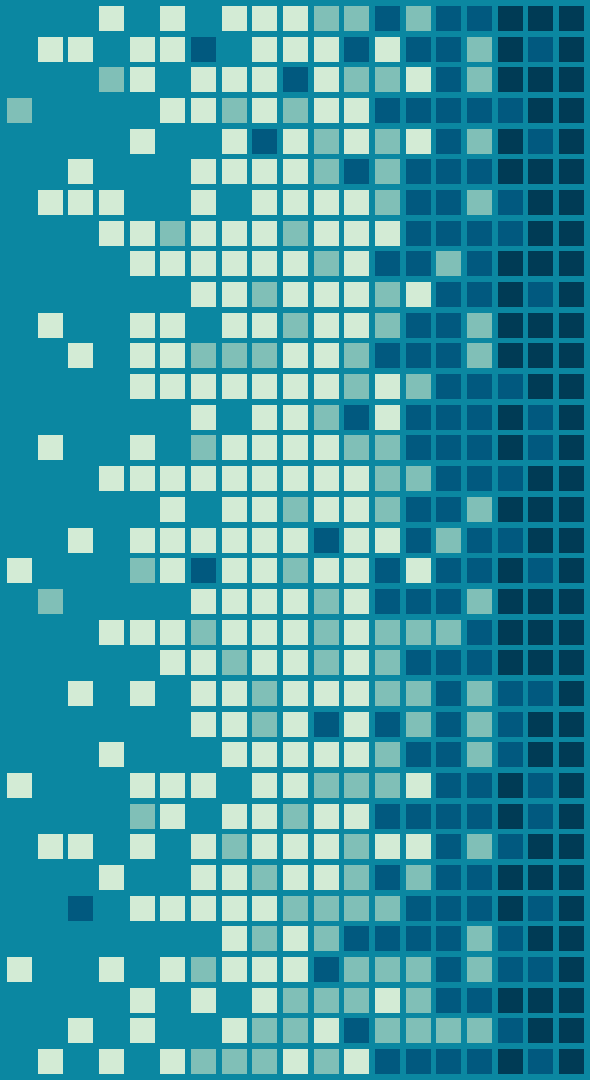
Kubernetes – kind: ReplicaSet



```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2021-12-05T16:32:06Z"
  generateName: webservice-
  labels:
    app: webservice
  name: webservice-85sqf
  namespace: default
  ownerReferences:
  - apiVersion: apps/v1
    blockOwnerDeletion: true
    controller: true
    kind: ReplicaSet
    name: webservice
    uid: eb0c26f6-76d6-43bc-9b2c-0c27a756eb7f
  resourceVersion: "247383"
  uid: 0721d489-a23b-487d-a037-d984290a2192
spec:
  containers:
  - image: webservice
    imagePullPolicy: Never
    name: webservice
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: kube-api-access-jttv8
      readOnly: true
  dnsPolicy: ClusterFirst
  enableServiceLinks: true
  nodeName: chlorine
  preemptionPolicy: PreemptLowerPriority
  priority: 0
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
  tolerations:
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  volumes:
  - name: kube-api-access-jttv8
    projected:
      defaultMode: 420
      sources:
```

“Comment accéder aux pods automatiquement ?

On va pas récupérer leur IP à chaque pour la filer au client ...



Notion de service

Une abstraction par dessus qui fait plaisir

k8s - Exposer un service

- Comment exposer un service avec docker ?
  - `docker run -p 0.0.0.0:80:9200`
  - `docker run -p 127.0.0.1:9200:9200 +`
    `reverse proxy`
  - `docker network --create toto +`
    `docker run --network toto <service> +`
    `docker run --network toto -p 0.0.0.0:80:80`
    `-e redirect_to=<service> <reverse proxy>`

k8s - Exposer un service

➔ Comment exposer un service avec docker qui a plusieurs backends ?

k8s - Exposer un service

- ➔ Comment exposer un service avec docker qui a plusieurs backends ?
  - ~~docker run -p 0.0.0.0:80:9200 +~~
    ~~docker run -p 0.0.0.0:80:9200~~

k8s - Exposer un service

- ➔ Comment exposer un service avec docker qui a plusieurs backends ?
  - ~~`docker run -p 0.0.0.0:80:9200 +`~~
    ~~`docker run -p 0.0.0.0:80:9200`~~
  - `docker run -p 127.0.0.1:9200:9200 +`
    `docker run -p 127.0.0.1:9201:9200 +`
    `reverse proxy`

k8s - Exposer un service

- ➔ Comment exposer un service avec docker qui a plusieurs backends ?
  - ~~`docker run -p 0.0.0.0:80:9200 +`~~
    ~~`docker run -p 0.0.0.0:80:9200`~~
  - `docker run -p 127.0.0.1:9200:9200 +`
    `docker run -p 127.0.0.1:9201:9200 +`
    `reverse proxy`
- ➔ Comment gérer la scalabilité ?

k8s - Exposer un service

- ➔ Comment exposer un service avec docker qui a plusieurs backends ?
  - ~~`docker run -p 0.0.0.0:80:9200 +`~~
    ~~`docker run -p 0.0.0.0:80:9200`~~
  - `docker run -p 127.0.0.1:9200:9200 +`
    `docker run -p 127.0.0.1:9201:9200 +`
    `reverse proxy`
- ➔ Comment gérer la scalabilité ?
  - ¯_(ツ)_/¯

k8s - Exposer un service

- Avec k8s ?
- Objet `service`
- Abstraction du/des backend(s) pour offrir un entrypoint unique
  - 1, 3, 1000 backends ? 1 endpoint
- "Équivalent" d'un meilleur port binding de docker

k8s - Exposer un service

- kind: Service
- spec.selector
- ports:
  - source->destination
- Loadbalance entre les pods matchés
  - Loadbalancing = random entre Pods disponibles
- Résultat exposé ?
  - Dépend du type
  - Par défaut clusterIP

k8s - Exposer un service

```
┌─zarak@chlorine ~/misc/kubernetes
└─$ ls
basic-refresh  pause                pod-2services.yml  replicaset.yml  webservice
basic-service  pod-2containers.yml  pod-simple.yml     service.yml     webservice-pod.yml
┌─zarak@chlorine ~/misc/kubernetes
└─$ vim service.yml
┌─zarak@chlorine ~/misc/kubernetes
└─$ k apply -f service.yml
service/webservice created
┌─zarak@chlorine ~/misc/kubernetes
└─$ k get service
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.43.0.1      <none>        443/TCP   24d
webservice   ClusterIP   10.43.69.232   <none>        80/TCP    2s
┌─zarak@chlorine ~/misc/kubernetes
└─$ curl 10.43.69.232
root%
┌─zarak@chlorine ~/misc/kubernetes
└─$ curl 10.43.69.232/whoam
```

k8s - Exposer un service

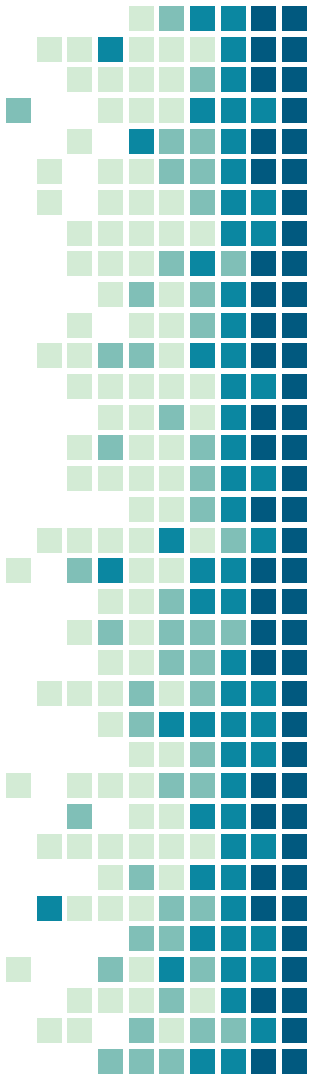
- kind: Service
- spec.type: NodePort
- Port mappé sur tous les noeuds vers le service
- exemple

k8s - Exposer un service

- ➔ kind: Service
- ➔ spec.type: LoadBalancer
- ➔ Demande au loadbalancer externe (du cloud provider) de fournir un entrypoint
- ➔ Pour exposer publiquement un service

k8s - Exposer un service

- ➔ kind: Service
- ➔ spec.type: LoadBalancer
- ➔ Demande au loadbalancer externe (du cloud provider) de fournir un entrypoint
- ➔ Pour exposer publiquement un service
- ➔ ClusterIP = dans le cluster, local

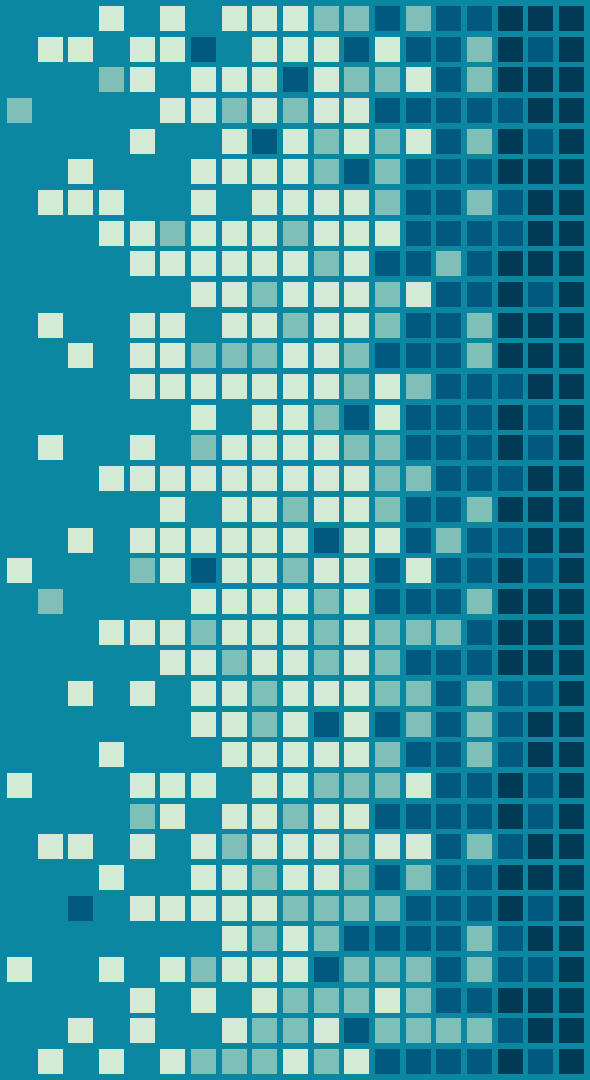


k8s - Exposer un service

- ➔ kind: Service
- ➔ spec.type: LoadBalancer
- ➔ Demande au loadbalancer externe (du cloud provider) de fournir un entrypoint
- ➔ Pour exposer publiquement un service
- ➔ ClusterIP = dans le cluster, local
- ➔ LoadBalancer = accessible hors cluster, public

“Oui mais comment accéder aux pods automatiquement via le service?

On va pas récupérer la clusterIP à chaque fois ...



k8s - Exposer un service

- kind: Service
- LoadBalancer = accessible hors cluster, public
  - Dépend du cloudprovider
  - Possibilité d'avoir une IP fixe
  - … et des DNS
- clusterIP = dans le cluster, local
  - Si architecture micro-services, comment trouver un autre micro service ?
  - Dans le cluster, local = DNS interne

k8s – DNS

- ➔ Kube-dns
- ➔ Allocation automatique d'IP pour les pods et services

k8s – DNS

- Kube-dns
- Allocation automatique d'IP pour les pods et services
  - Pourquoi pas d'enregistrements DNS ?

k8s - DNS

- Kube-dns
- Allocation automatique d'IP pour les pods et services
  - Pourquoi pas d'enregistrements DNS ?

```
$ k get service --all-namespaces
NAMESPACE     NAME             TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)                      AGE
default       kubernetes       ClusterIP      10.43.0.1       <none>          443/TCP                      24d
kube-system   kube-dns         ClusterIP      10.43.0.10      <none>          53/UDP,53/TCP,9153/TCP       24d
kube-system   metrics-server   ClusterIP      10.43.99.213    <none>          443/TCP                      24d
kube-system   traefik          LoadBalancer   10.43.168.43    192.168.1.18    80:31123/TCP,443:31573/TCP   24d
default       webservice       LoadBalancer   10.43.69.232    <pending>       80:30000/TCP                 60m
```

k8s – DNS

```
$ k get svc -o wide
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE   SELECTOR
kubernetes   ClusterIP   10.43.0.1      <none>        443/TCP   29d   <none>
webservice   ClusterIP   10.43.128.18   <none>        80/TCP    9s    app=webservice
$ dig webservice.default.svc.cluster.local @10.43.0.10

; <<>> DiG 9.16.23 <<>> webservice.default.svc.cluster.local @10.43.0.10
;; global options: +cmd
;; Got answer:
;; WARNING: .local is reserved for Multicast DNS
;; You are currently testing what happens when an mDNS query is leaked to DNS
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10832
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 96045e0b4a294191 (echoed)
;; QUESTION SECTION:
;webservice.default.svc.cluster.local. IN A

;; ANSWER SECTION:
webservice.default.svc.cluster.local. 5 IN A    10.43.128.18

;; Query time: 3 msec
;; SERVER: 10.43.0.10#53(10.43.0.10)
;; WHEN: Sat Dec 11 17:34:59 CET 2021
;; MSG SIZE  rcvd: 129

```

k8s – DNS

```
$ k get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.43.0.1      <none>        443/TCP   29d
webservice   ClusterIP   10.43.128.18   <none>        80/TCP    2m29s
$ k get po
NAME                READY   STATUS    RESTARTS   AGE
webservice-85sqf    1/1     Running   3          6d
webservice-bwb6d    1/1     Running   3          5d22h
webservice-24w2f    1/1     Running   3          6d
infinite-loop       1/1     Running   0          75s
$ k exec -it infinite-loop -- bash
bash-5.1$ curl webservice/whoami && echo
webservice-24w2f
bash-5.1$ curl webservice/whoami && echo
webservice-85sqf
bash-5.1$ exit
$ dig webservice @10.43.0.10

; <<>> DiG 9.16.23 <<>> webservice @10.43.0.10
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 2660
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 2ff146690e27900c (echoed)
;; QUESTION SECTION:
;webservice.            IN  A

;; AUTHORITY SECTION:
.           30  IN  SOA a.root-servers.net. nstld.verisign-grs.com. 2021121100 1800
900 604800 86400

;; Query time: 23 msec
;; SERVER: 10.43.0.10#53(10.43.0.10)
;; WHEN: Sat Dec 11 17:37:44 CET 2021
;; MSG SIZE  rcvd: 126
```

k8s – Exposer un service HTTP

➔ Service = port plus IP

k8s – Exposer un service HTTP

- ➔ Service = port plus IP
- ➔ Exposition publique = une IP externe

k8s – Exposer un service HTTP

- ➔ Service = port plus IP
- ➔ Exposition publique = une IP externe
- ➔ Si plusieurs services HTTP (port 80/443), plusieurs IP externes

k8s – Exposer un service HTTP

- ➔ Service = port plus IP
- ➔ Exposition publique = une IP externe
- ➔ Si plusieurs services HTTP (port 80/443), plusieurs IP externes
- ➔ Solutions possibles :

k8s – Exposer un service HTTP

- ➔ Service = port plus IP
- ➔ Exposition publique = une IP externe
- ➔ Si plusieurs services HTTP (port 80/443), plusieurs IP externes
- ➔ Solutions possibles :
  - ◆ Gros reverse proxy interne

k8s – Exposer un service HTTP

- ➔ Service = port plus IP
- ➔ Exposition publique = une IP externe
- ➔ Si plusieurs services HTTP (port 80/443), plusieurs IP externes
- ➔ Solutions possibles :
  - ◆ Gros reverse proxy interne
    - Replicaset + service + bricoles

k8s – Exposer un service HTTP

- ➔ Service = port plus IP
- ➔ Exposition publique = une IP externe
- ➔ Si plusieurs services HTTP (port 80/443), plusieurs IP externes
- ➔ Solutions possibles :
  - ◆ Gros reverse proxy interne
    - Replicaset + service + bricoles
    - Reload de la config générale à chaque modif

k8s - Exposer un service HTTP

- Service = port plus IP
- Exposition publique = une IP externe
- Si plusieurs services HTTP (port 80/443), plusieurs IP externes
- Solutions possibles :
  - Gros reverse proxy interne
    - Replicaset + service + bricoles
    - Reload de la config générale à chaque modif
  - La même chose mais intégré Kubernetes : l'ingress

k8s – HTTP – Ingress

- ➔ Pas l'ancêtre de Pokémon GO
- ➔ Objet k8s

k8s – HTTP – Ingress

- Pas l'ancêtre de Pokémon GO
- Objet k8s
- Signifie entrée (opposition à egress)
- Reverse proxy pour gérer la répartition du trafic entrant
  - Le service `LoadBalancer` géré par l'ingress controller
  - Création de ressources de type "Ingress" par service à exposer
  - Pas de contrôle direct du reverse proxy, abstraction
  - Répartition du trafic via le contenu
    - HTTP : header Host + request line
    - HTTPS : SNI (+ request line si terminaison SSL/TLS)

k8s - HTTP - Ingress

- ➔ Déployé par l'ops dans le cluster = pas la responsabilité du dev de gérer l'ingress controller
- ➔ Le dev gère les ingress
- ➔ Pas à se soucier du controller sous-jacent = abstraction
- ➔ Pas besoin d'écrire une configuration spécifique nginx/haproxy/traefik/...
- ➔ Ingress controller déployé de base dans k3s/minikube
- ➔ Exposons proprement notre webservice :
  - ◆ Création d'ingress

k8s – HTTP - Ingress

- ➔ Possibilité de controler :
  - L'hostname
  - Les routes
  - Les ports
  - …

k8s – HTTP – Ingress

- Possibilité de controler :
  - L'hostname
  - Les routes
  - Les ports
  - ...

```
$ k get ingress --all-namespaces
NAMESPACE   NAME         CLASS    HOSTS   ADDRESS         PORTS   AGE
default     webservice   <none>   *       192.168.1.18    80      112s
```

Comment savoir à qui dispatch ?

Les probes, savoir qui est prêt

k8s - Probes - problématique

- Un service load balance entre plusieurs pods
- Un pod peut prendre beaucoup de temps à être prêt
  - Lancement du pod, chargement en mémoire de données, connexions aux DB à établir, ...
- Comment ne pas envoyer du trafic à un pod en train de boot ?
- Savoir quand il est prêt
- Le signaler
- Comment savoir ?
  - En testant

k8s - Probes

- 3 types de Probes :
  - StartupProbe
  - LivenessProbe
  - ReadinessProbe
- Définie dans la spec d'un Pod
- Effectuent un test à interval régulier
- Gérées par Kubernetes
  - Gestion du trafic voire restart d'une app deadlock

k8s – Probes

- ➔ Exemple de StartupProbe
- ➔ Exemple de ReadinessProbe
- ➔ Exemple de LivenessProbe

Les autres workload controllers

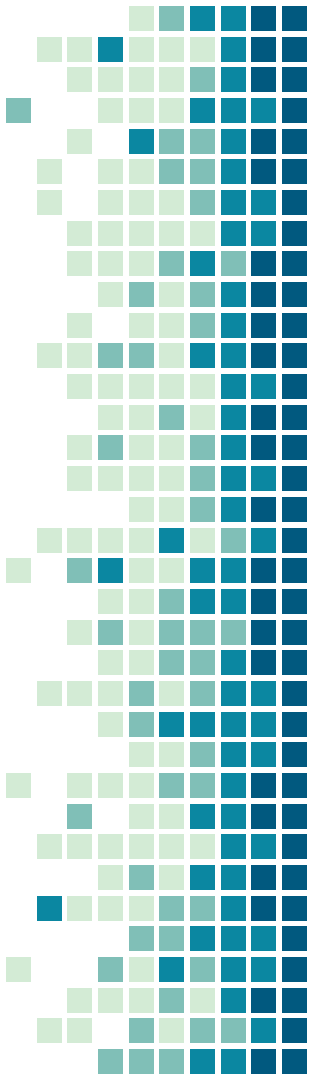
Soyons plus intelligents que le replicaset

k8s – workload controllers

- ➔ Plusieurs types, plusieurs utilisations :
  - ReplicaSet
  - Deployment
  - StatefulsSet
  - DaemonSet
  - Job/CronJob

k8s – job

- Comme un Pod
  - kind: Job évidemment
- Le container ne doit pas être un daemon
- C'est un job : il fait son job et se casse
- Kube relance le job si le container exitcode != 0



k8s - CronJob

➔ Un controller qui lance des Job selon des critères de périodicité

k8s – CronJob

- ➔ Un controller qui lance des Job selon des critères de périodicité
- ➔ … comme cron

k8s - CronJob

- ➔ Un controller qui lance des Job selon des critères de périodicité
- ➔ … comme cron
- ➔ Rien de plus à dire

k8s - DaemonSet

- ➔ Un controller qui lance un Pod sur chacun des noeuds du cluster k8s
- ➔ On rajoute un noeud : run d'un Pod
- ➔ On supprime un noeud : stop le Pod
- ➔ Utile pour :
  - ◆ Le monitoring
  - ◆ Le stockage local
  - ◆ La collection de logs

k8s - Deployment

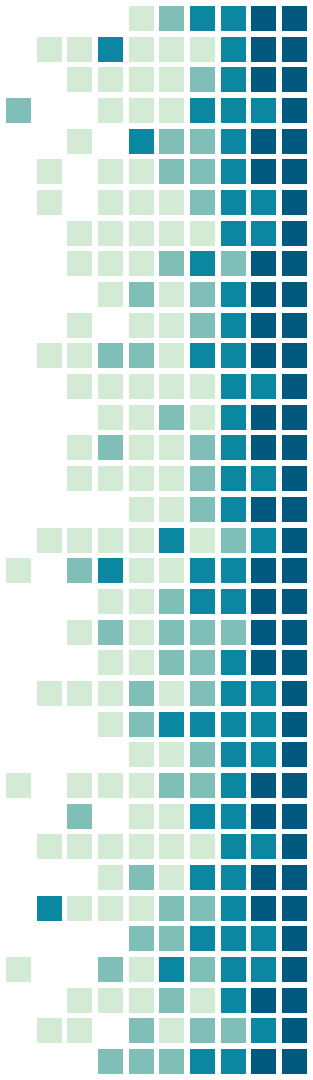
- Un controller plus intelligent que ReplicaSet
- Utilise des Replicaset pour générer les pods
- Modifie le(s)s Replicaset(s) pour modifier les pods
- Exemple
- Gère les rolling release par défaut :
  - Modification des Pods étape par étape
  - Démarrage d'un nouveau Pod
  - Arrêt d'un ancien
  - Etc
  - Exemple

k8s - Deployment

- L'ancien rs non supprimé
  - Scale à 0 Pods
  - Toujours accessible
- `kubectl rollout`
  - Voir l'avancement d'un déploiement
  - Voir l'historique des déploiements
  - Rollback vers une ancienne version
  - Pause/Reprendre un déploiement
- Ne pas utiliser de ReplicaSet directement, mais utiliser un Deployment

k8s - StatefulSets

- ➔ Un genre de Deployment, pour des données stateful
- ➔ Garantie sur l'unicité et l'ordre de déploiement des Pods
- ➔ Pas de fongibilité des Pods
- ➔ Utile pour assurer l'association entre Pod et stockage persistent
- ➔ Bon exemple : elasticsearch



k8s – StatefulSets

```
$ k get po
NAME                  READY   STATUS    RESTARTS   AGE
infinite-loop         1/1     Running   2          2d23h
complex-webservice-0  0/1     Running   0          23s
$ k get po
NAME                  READY   STATUS    RESTARTS   AGE
infinite-loop         1/1     Running   2          2d23h
complex-webservice-0  1/1     Running   0          37s
complex-webservice-1  0/1     Running   0          12s
$ k get po
NAME                  READY   STATUS    RESTARTS   AGE
infinite-loop         1/1     Running   2          2d23h
complex-webservice-0  1/1     Running   0          66s
complex-webservice-1  1/1     Running   0          41s
complex-webservice-2  0/1     Running   0          16s
```

Et après ?

Aller plus loin

k8s – Controller / Operator

- Un controller surveille des ressources et prends des actions
  - Création d'une ressource particulière
  - Modification/suppression
  - Événement sur la cible
  - Exemple : Ingress controller
  - Exemple : workload controller (Deployment, ReplicaSet, ..)
- Un Operator est un Controller qui opère avec des CRD
  - Custom Resource Definition

k8s – Operator par l'exemple

➔ k8s gère une ressource de type secret

k8s – Operator par l'exemple

- k8s gère une ressource de type secret
  - Comme une ConfigMap, mais base64 encoded

k8s – Operator par l'exemple

- ➔ k8s gère une ressource de type secret
  - Comme une ConfigMap, mais base64 encoded
  - Sémantique

k8s – Operator par l'exemple

- k8s gère une ressource de type secret
  - Comme une ConfigMap, mais base64 encoded
  - Sémantique
  - Possible de créer une var d'env dans un Pod à partir d'un Secret

k8

```
$ cat secret.yml
---

apiVersion: v1
kind: Secret
metadata:
  name: not-so-super-secret
type: Opaque
data:
  username: YWRtaW4=
  password: cGFzc3dvcmQ=
$ k apply -f secret.yml
secret/not-so-super-secret created
$ k get secrets
NAME                  TYPE                                  DATA   AGE
default-token-k8w67   kubernetes.io/service-account-token   3      5m35s
not-so-super-secret   Opaque                                2      7s
$ k get secrets not-so-super-secret -o YAML
apiVersion: v1
data:
  password: cGFzc3dvcmQ=
  username: YWRtaW4=
kind: Secret
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":{"password":"cGFzc3dvcmQ=","username":"YWRtaW4="},"kind":"Secret","metadata":
{"annotations":{},"name":"not-so-super-secret","namespace":"default"},"type":"Opaque"}
  creationTimestamp: "2021-12-15T10:51:54Z"
  name: not-so-super-secret
  namespace: default
  resourceVersion: "986"
  uid: a629883c-5c02-4dec-ab5d-a1e674ddd757
type: Opaque
$ k get secrets not-so-super-secret -o JSON | jq -r .data.password | base64 -d
password%
```

k8s – Operator par l'exemple

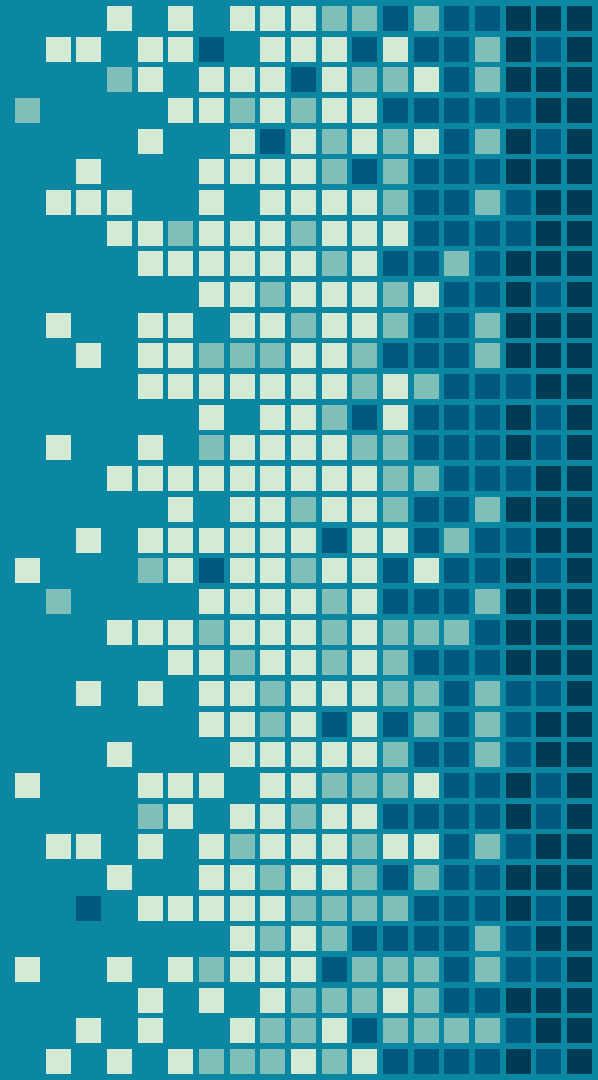
- k8s gère une ressource de type secret
  - Comme une ConfigMap, mais base64 encoded
  - Sémantique
  - Possible de créer une var d'env dans un Pod à partir d'un Secret
  - Possible de mount un secret comme un fichier dans un Pod

k8s – Operator par l'exemple

- k8s gère une ressource de type secret
  - Comme une ConfigMap, mais base64 encoded
  - Sémantique
  - Possible de créer une var d'env dans un Pod à partir d'un Secret
  - Possible de mount un secret comme un fichier dans un Pod
- Problème : comment stocker le secret dans git ?

k8s - Operator par l'exemple

- k8s gère une ressource de type secret
  - Comme une ConfigMap, mais base64 encoded
  - Sémantique
  - Possible de créer une var d'env dans un Pod à partir d'un Secret
  - Possible de mount un secret comme un fichier dans un Pod
- Problème : comment stocker le secret dans git ?
  - Une solution possible : sealed-secrets

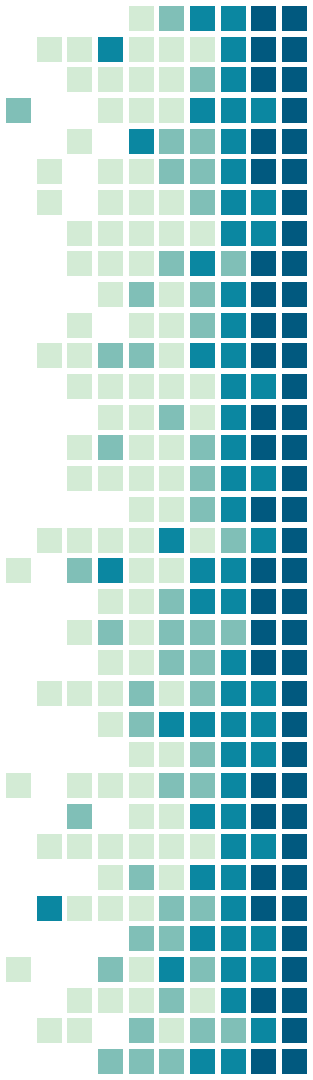


Problem: "I can manage all my K8s config in git, except Secrets."

Solution: Encrypt your Secret into a SealedSecret, which is safe to store – even to a public repository. The SealedSecret can be decrypted only by the controller running in the target cluster and nobody else (not even the original author) is able to obtain the original Secret from the SealedSecret.

k8s – Operator par l'exemple

- sealed-secrets projet de bitnami
- Fonctionne avec une CRD et un operator
- Secret chiffré avec kubeseal
- Mis dans k8s en tant que `SealedSecret`
  - Utilise l'API `bitnami.com/v1alpha1`



k8s -

```
$ kubeseal --controller-name sealed-secrets --format yaml < secret.yml > sealed-secret.yml
$ cat sealed-secret.yml
apiVersion: bitnami.com/v1alpha1
kind: SealedSecret
metadata:
  creationTimestamp: null
  name: not-so-super-secret
  namespace: default
spec:
  encryptedData:
    password: AgCMD7Wwo7C5UjEO6AfQGx5Iy1xXVFtZwr8ESyb8XxCHQMZ/xPTfRZTefSIKOQ1gBWeT2
/HUKAc+52yFBmOFIZqlHWEWCSUOA0XHyX7moWwYWWXOwgIq66
/TDUoNXpsLuziwHSyhEUDbpOaecl45wthpBX2+KqAU4jyo+mHHjfPpSf1HhBjvTp+HYrowbCFpSzMG+Q1iubsqj+OVV8jK7IwaCioRv9uPPdm+BTVQLdApYefi
HcMChd8Pv2zxEUVxEvgVpRA61oh3yTIYRUoASjvXCLDx6gyIeIgyz1qNelUV1a67MpbegJyMU/c8dZV2
/yRlwKLHs1wq03rP2AXNDNy8IQDXzFMRNKtk7guqSYmCYqWbyO6dzgHYccvf4jwUu
/dRKqm+Hsc1TP64kjOsVJoA3mztwicg27jWunSJfyhHMrqumSvF4YzPM+YK00U4kqvt5C3SFIFNqCkizplA7LViF4wD8l
/Jm66qkhZDJZlAQAIDfJlZ4rZrevdJq1xxI+ogoDyFgkA0zNzHkQUZcQAiDcKZ04wd3zPXHd7EyK
/qXggJdeVwT8915gM2aTTzl5xGu8V33qfCty4IumJ6NtVBys8EfvP6dSa70PesNL7ZyGRuoV6o91lFUwURW8rf08zZH8ICdDDpokISVAgVDfdg240niDf8zLlP
XE93ybRkOaT4cGXukU5NWxpX9Ns7P4SlByMs0673nA==
    username: AgBgSE++7+FrIpngjGim31EKhSVbjw2h7ioo9FvPzFIfpQAzthUrTa3NqqzY+l0NA0urCAQtwRvm47tRbQhyH/u3KzvDSr0HVrrJ7aBS
/V3uM/0T9sTfLI13zvKYoQGDvZAA4XAsX+QucoPUplmP5qw3teNjAx7s0CLl8enNQQm9JVWYEz4Qoquk3G2RHoBlJqcqsrOs+OfITFXahy4U3orlNP
/ccT4B+ovh2gPtsuLvYJ4LRV7FnPvVEKB8i3IYF4qJDc7/mJ7T0WDDeQmKbe7Pc5q/w2r4To2UXegJxQMJd8Ar32tE/IO2kTlvh9Qn1obKaOWfDbttRQ+8fxTt
/LMLHUGmLHIFjYMiaRQqqgTnA94K+f5YfIBa5w6kVYmlRP1ZqBv1fjZQXAUWDceHHgbGwvydeofqj3HbWq+PGNbK81hG7GgKnaofLRXcIID+f9SuVYuNNuxNLh
M82SRL/qkgz0pPJiOoJRAWzon7kHwPheVeTLqAj0p
/k2rbSEnA6TxWUatR0dOGoL20hw85u6BnEUvsvdEuIKX9dEz9Yv1M+29tBdIV1j8wMPSMgRoroNQz54yRUcihy+WqjHrnbwTTloPt76eyzNrzy6p9DSwmRVt3
/h5fsLAHIT3e6dH/3jahstzU8QRMljvO70z34wocndhpVDLMMhAXD14X4SeQgdpCIO/wktm+Jf4ov0aA7VZsKu7DoQ==
  template:
    data: null
    metadata:
      creationTimestamp: null
      name: not-so-super-secret
      namespace: default
    type: Opaque
$ k get secrets
NAME                  TYPE                                  DATA   AGE
default-token-k8w67   kubernetes.io/service-account-token   3      20m
$ k apply -f sealed-secret.yml
sealedsecret.bitnami.com/not-so-super-secret created
$ k get sealedsecrets.bitnami.com
NAME                  AGE
not-so-super-secret   5s
$ k get secrets
NAME                  TYPE                                  DATA   AGE
default-token-k8w67   kubernetes.io/service-account-token   3      20m
not-so-super-secret   Opaque                                2      8s
$ k -n kube-system logs sealed-secrets-7569f57679-w2f84
2021/12/15 11:06:58 Updating default/not-so-super-secret
2021/12/15 11:06:58 Event(v1.ObjectReference{Kind:"SealedSecret", Namespace:"default", Name:"not-so-super-secret",
UID:"e1f94994-c7a1-45c5-b463-0282f1b8cbfd", APIVersion:"bitnami.com/v1alpha1", ResourceVersion:"1628", FieldPath:""}):
type: 'Normal' reason: 'Unsealed' SealedSecret unsealed successfully
```

k8s - Operator par l'exemple

- ➔ sealed-secrets est un operator
- ➔ Il fourni sa CRD : SealedSecret
- ➔ L'operator tourne utilise l'API de kubernetes pour surveiller les modifications de SealedSecrets
- ➔ Si création d'un SealedSecret, il le déchiffre et créé un Secret
- ➔ Si suppression d'un SealedSecret, il supprime le Secret associé

Kubernetes et templating

Nous n'avons pas les mêmes valeurs

Templating

- Plusieurs micro-services
- Globalement identiques à quelques exceptions près
  - Image docker
  - Certaines variables d'environnement peut-être ?
  - …
- Nécessaire de copier x fois Deployment/Service/Ingress/… ?
- Template = squelette
- Dérivation du template avec les valeurs
- Utilisation d'Helm

Helm

- Plusieurs solutions de templating avec k8s
- Helm plus répandue
  - Facile d'utilisation également
- Helm registry
  - Beaucoup de templates déjà proposés
  - Ex: bitnami's sealed-secrets
- Version 3 actuelle : pas de config dans le cluster

Essayons helm

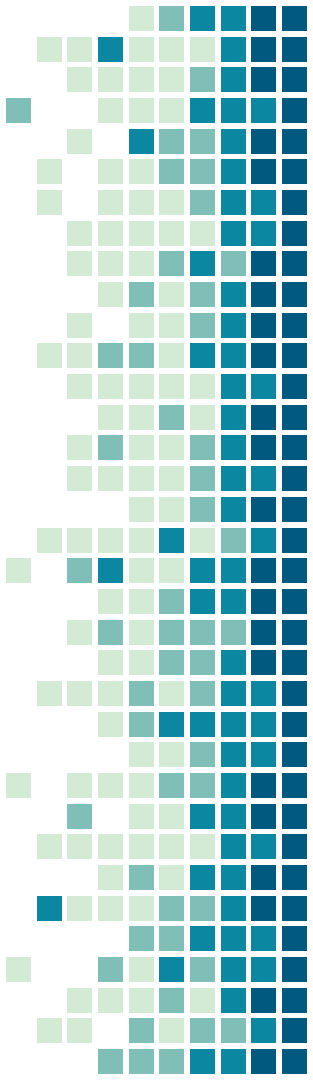
En tant que lecteur, puis auteur

Helm - chart

- ➔ Helm propose des charts
- ➔ Un chart est une ressource helm pour installer une application
- ➔ Un chart peut comporter:
  - ◆ Un ou des templates
  - ◆ Des éventuels sous-charts en dépendances
    - ● Par exemple une base de données, un message broker, etc

Helm - chart & releases

- ➔ Helm fait la différence entre un chart et une release
  - ◆ Une release est une application d'un chart
  - ◆ La release est dans un namespace
  - ◆ La release a un nom, et une version
  - ◆ La release est liée à un template
- ➔ On peut télécharger des charts via la CLI de helm
  - ◆ Mais pas de release donc



Helm – Anatomie d'un chart

- Un chart helm s'oriente autour de 4 fichiers
- Chart.yaml
  - Contient plusieurs information sur le Chart
    - Nom
    - Version
    - Version d'API de helm à utiliser
    - La liste des éventuelles dépendances
    - Des metadata
    - ….

Helm – Anatomie d'un chart

- templates/
- Ce dossier contient les fichiers de template
  - Les ressources Kubernetes avec éventuellement des emplacements de templating
    - En *.yaml
  - Des éventuelles fonctions de templating
    - En .tpl

Helm - Anatomie d'un chart

- values.yml
- Ce fichier contient les valeurs par défaut de notre chart
- Il est courant de créer des charts qui fonctionnent directement sans changer les valeurs
  - Evidemment les valeurs ont pour but d'être changées si besoin
- Le but est de séparer configuration et "code" (templates)
- Fournir un descriptif immédiat et simple de ce qui peut changer
- Un utilisateur ne "doit" pas changer le template

Helm – Anatomie d'un chart

- ➔ charts/
- ➔ Ce dossier peut contenir d'autres charts helm utilisé en dépendances
- ➔ Peut être rempli automatiquement à partir de Chart.yaml

Helm - Anatomie d'un chart

- ➔ Il existe d'autres subtilités et fichiers pour un chart helm complet ou avancé
- ➔ Peu pertinent de les évoquer pour le moment

Helm – obtenir des charts

- ➔ Helm permet de télécharger et publier des charts directement
- ➔ Il est courant de:
  - ◆ Ajouter le repo d'une organization fournissant un chart nous intéressant:
    - helm repo add jaegertracing https://jaegertracing.github.io/helm-charts
  - ◆ Télécharger un chart en local pour l'examiner
    - helm pull jaegertracing/jaeger
  - ◆ Créer une release à partir d'un chart
    - Local: helm install my-jaeger .
    - Directement d'un repo: helm install my-jaeger jaegertracing/jaeger

Helm - les values

- Un chart téléchargé sur un repo aura des valeurs par défaut
- Le plus souvent on va vouloir changer ces valeurs
- helm show value <chart name>
- On peut alors :
  - Changer une ou deux valeurs via la CLI et --set
  - Donner un fichier avec les valeurs modifiées via --values
- Pas besoin de tout redéfinir
- Les valeurs précisées override les existantes
- Attention à la syntax

Ecrire un chart helm

Une activité familiale

Helm – écrire un chart

- Regardons rapidement comment écrire un chart basique
- Pour commencer, il faut un Chart.yaml
- Dedans, il suffit de mettre:
  - apiVersion: v2
  - appVersion: 0.1.0
  - version: 0.1.0
  - name: my-chart
- Si l'on souhaite y mettre plus d'informations, se référer à la documentation ou à des charts existants

Helm – écrire un chart

- La partie complexe se passe dans le dossier templates/
- La division classique dans templates/:
  - séparer son chart en plusieurs fichiers .yaml
  - Chaque fichier correspond à un type de ressource
  - Chaque fichier est nommé par le type de ressource créé
  - Si grosse application divisable en composant, on fait la séparation
- Exemple:
  - composant-A-service.yaml
  - composant-A-deployment.yaml
  - composant-B-deployment.yaml

Helm – good practices

- ➔ Au fur et à mesure que l'on écrit des ressources kube, on remplace les valeurs hardcodées par du template
- ➔ On essaye de créer des valeurs hiérarchiques
- ➔ Exemple:
- ➔

```
composantA:
  image:
    name: registry.com/composantA
    tag: v0.1.0
    pullPolicy: Always
```

Helm – good practices

- ➔ Les choix “arbitraires” ou “hardcodés” doivent être modifiables via les variables
- ➔ Les variables doivent permettre de rajouter des annotations libres
- ➔ Les variables doivent permettent de contrôler des choix comme:
  - Type de Service
  - Deployment d’un ingress ou non
  - Port exposé
  - Nom d’un secret préexistant à utiliser vs créer le sien
  - …

Helm - templating

- ➔ Comment utiliser les values dans le template ?
- ➔ Helm utilise un mélange de go templating et de spring templating
  - ◆ Avec des fonctions personnalisées
- ➔ Faire appel à une valeur se fait avec {{ <key> }}
- ➔ La key va permettre de choisir ce qui remplacera le template
- ➔ La key est préfixée par . qui désigne le scope de haut niveau
- ➔ Il faut ensuite choisir dans l'arborescence la valeur

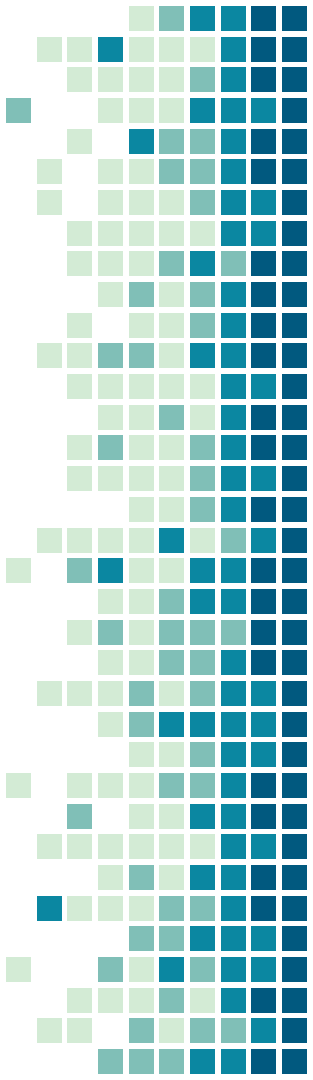
Helm - top-level objects

- Helm met à disposition plusieurs objet de haut niveau dans le scope de haut niveau par défaut
  - Les scopes ne seront pas abordés plus en détails ici cependant
- .Values contient les valeurs
  - Par défaut + par celles précisées par l'utilisateur
    - Override partiel
- .Release contient les valeurs propres à la release à installer
  - Le namespace de destination par exemple
  - Le nom de la release
  - …

Helm - top-level objects

- ➔ .Files permet d'interagir avec des fichiers
- ➔ .Chart contient le contenu de Chart.yaml

- ➔ .Capabilities contient des infos sur les capabilities du cluster
- ➔ .Template des infos sur le template actuel



Helm – variables

- ➔ Mettons en pratique l'utilisation du template pour le nom de l'image à utiliser
- ➔ Dans composant-A-deployment.yaml par exemple:
- ➔ image: {{ .Values.composantA.image.name }}:{{ .Values.composantA.image.tag }}

Helm - fonctions

- ➔ Il est possible d'utiliser des fonctions
- ➔ Par exemple vous voulez remplacer les . de la version par des tirets
- ➔ {{ .Chart.appVersion | replace "." "-" }}
- ➔ Vous voulez être sûr que l'argument est une string
- ➔ {{ .Values.my-value | toString }}

Helm - flow control

- ➔ Il est possible d'utiliser des if/then/else et des loops
- ➔ {{ if eq .Values.myvalue "toto" }}it says toto{{ else }}it says something else{{ end }}
- ➔ Plus d'information sur la doc de helm "flow control"
- ➔ Lire "Chart template guide"
- ➔ Et "helm best practices"

ArgoCD

Deployment et kube

Deployment

- ➔ Les templates helm regroupent des ressources kube en “app”
- ➔ Mais comment déployer ces apps automatiquement ?
- ➔ Les maintenir à jour ?
- ➔ Avoir de la visibilité sur ce qui est fait ?
- ➔ Déployer à la main avec helm install est possible
  - ◆ Mais c’est pénible, et pas automatique
  - ◆ Et on aime bien les trucs automatiques
  - ◆ On pourrait utiliser des CI ?

Gitops

- Avec Kubernetes, c'est l'occasion d'utiliser une approche GitOps
- Le GitOps c'est utiliser git comme référentiel de vérité
- Approche classique :
  - Une feature est merge sur master dans git
  - Une CI/CD test et deploy la feature automatiquement
    - Ou un ops le fait manuellement
  - Que se passe-t-il si des modifications sont faites manuellement ?
  - Comment s'assurer de l'état des plateformes ?

Gitops

- Le GitOps introduit un concept de pull-based deployment contrairement au push-based
- Le mécanisme de deployment est intégré à la plateforme
- Ce mécanisme surveille en continue son état et l'état demandé par le repo git, et réconcilie les deux si besoin
- Avec Kubernetes, l'état est simple à comparer :
  - Il suffit de comparer le manifest dans le repo vs dans le cluster

ArgoCD

- ➔ L'approche GitOps est très répandu avec k8s
- ➔ L'outil le plus populaire est ArgoCD
- ➔ ArgoCD s'installe dans un cluster k8s
- ➔ On deploy des Applications (argoproj.io/v1alpha1)
- ➔ Une Application consiste en un application (souvent une release helm) à deployer via ArgoCD
- ➔ ArgoCD surveille la source de l'Application (repo git) et l'état actuel dans le cluster
- ➔ En fonction de sa configuration, deployment automatique ou pas

ArgoCD

- Une Application ArgoCD contient :
  - Des metadatas
  - Une source (repo git, chart helm)
    - Une version à déployer (tag, branche, …)
  - Des options de sync policy (automatic, manual, self heal, prune, …)
  - Des valeurs à donner au template helm pour la release
  - Une destination

ArgoCD - pourquoi ?

- ➔ Pourquoi ArgoCD est intéressant ?
- ➔ Deployment automatique
  - ◆ Pas besoin de s'en préoccuper
  - ◆ Garantie que le repo git représente la prod
  - ◆ Intégration directe avec le git workflow
  - ◆ Pas besoin d'écrire la logique de déploiement dans une CI/CD
  - ◆ Déploiement versionnable avec git

ArgoCD - comment ?

- ➔ Comment ArgoCD s'utilise ?
- ➔ On évoquait 3 "plateformes" de déploiement
  - dev/staging
  - qualif/ppr
  - prod
- ➔ On peut imaginer:
  - 3 Applications ArgoCD pour déployer l'application
  - Pour la prod: valeurs de prod, sur branche master, protégée
  - Pour la qualif: valeurs de qualif/prod, sur branche "qualif", semi-protégée
  - Pour le dev: valeurs de dev, branche "development", open-bar

ArgoCD & helm

- ➔ Une application ArgoCD peut déployer un chart helm
- ➔ Un chart helm peut déployer des Application ArgoCD
- ➔ On peut imaginer une structure complexe pour une application avec plusieurs micro-services comme:
  a. Une Application deploy un chart helm pour notre "application"
  b. Ce chart deploy plusieurs Applications pour chacun des micro-services
  c. Chacune des Applications deploy un chart helm pour le micro-service

Merci !

Des questions ?

Disponible sur zarak.fr/

Contact: cyril@cri.epita.fr

zarak production#5492