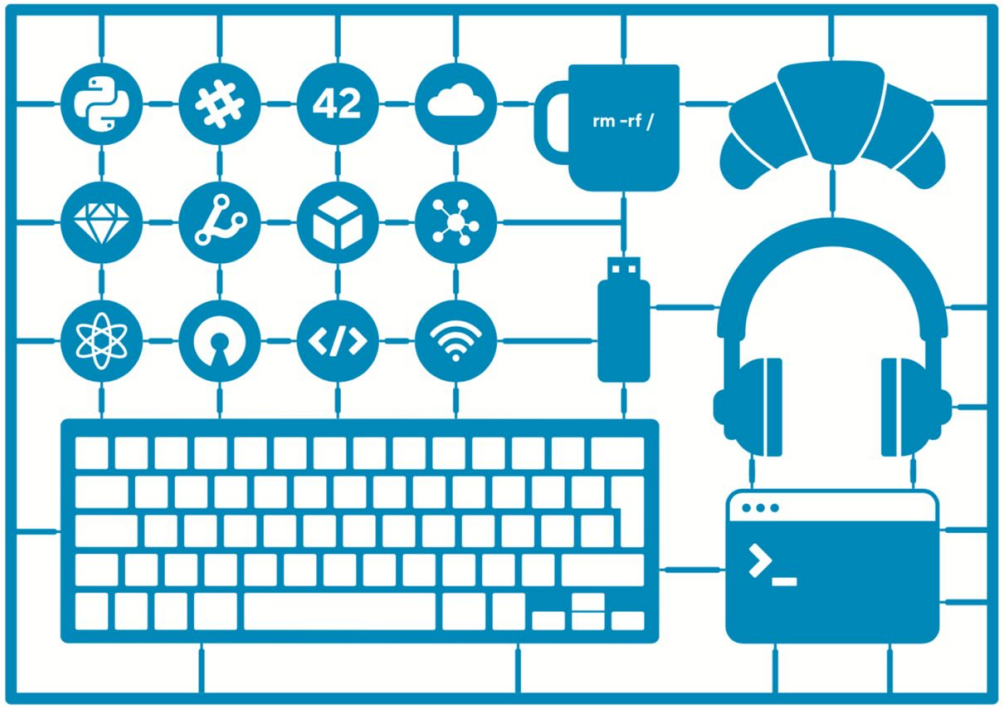
BBL by Orness : Introduction à Kubernetes

wifirst

ORNESS
Organisation, Network & Security Services

05 Mars 2019



Wifirst's Dev kit

1

Présentation

Qui suis-je ?

- Fan d'Internet et d'Open Source depuis 1995

- Ma stack du moment :
  - Ansible
  - OpenStack
  - Gitlab-CI and other testing buddies
  - Kubernetes

Aujourd'hui, Lead Architect & Lead DevOps chez Wifirst

- Parcours
  - 2000-2004 : LibertySurf / Tiscali (Telecom)
  - 2004-2007 : Orange with Orness (Telecom)
  - 2007-2015 : Agarik (Outsourcing)
  - 2015-2016 : Devoteam Open Solutions (Consulting)
  - 2017-Now : Wifirst (Telecom)

Wifirst ?

Chiffres clés

Wifirst est le leader européen du WiFi managé dans l'hôtellerie.

+2 M
de terminaux mobiles connectés chaque mois

150 K
Points d'accès WiFi supervisés

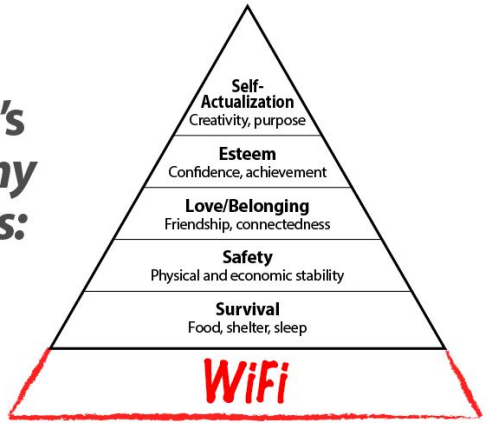
19
Pays couverts

140
Salariés

36M€
Chiffre d'affaires 2017

680 K
Chambres connectées

Maslow's *Hierarchy of Needs:*



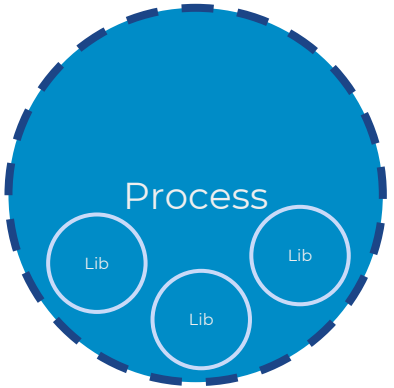
Wifirst ?

2

Un container ?

Principe du Container

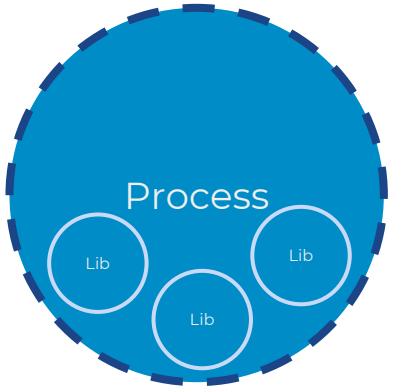
- Un container est un process qui tourne dans son propre environnement d'exécution
- Le container est isolé vis-à-vis des autres process qui tourne sur une machine et une limitation des ressources qu'il peut consommer est en place



Principe du Container

Une histoire d’isolation

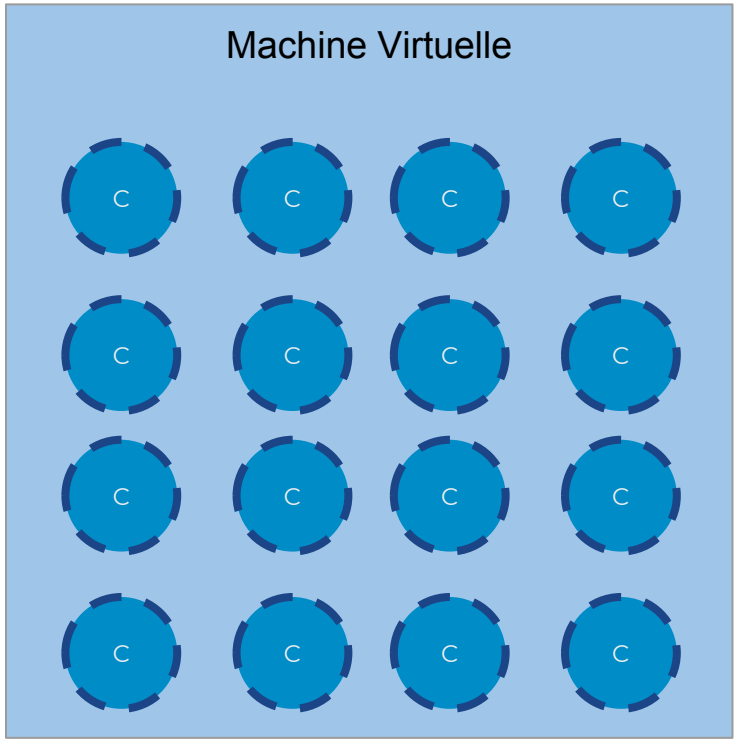
- 1979 Apparition des “chroot” permettant de limiter le système de fichier
- 2002 : Apparition des “namespace” permettant de limiter un process dans un sous-système du système (réseau, arbre de processus, montage de fichier,
- 2006 : Apparition des “Control Groups” permettant de limiter les ressources d’un process (CPU, mémoire, IO disk, IO network)
- 2008 : Les containers LXC sont officiellement nés
- 2013 : Première version Open Source de Docker
- 2014 : Première version Open Source de Kubernetes

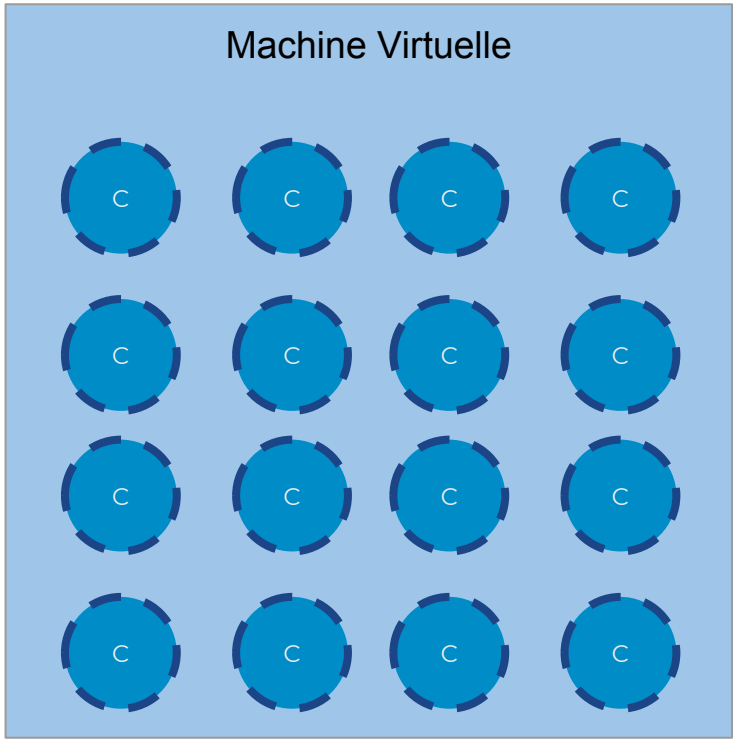


3

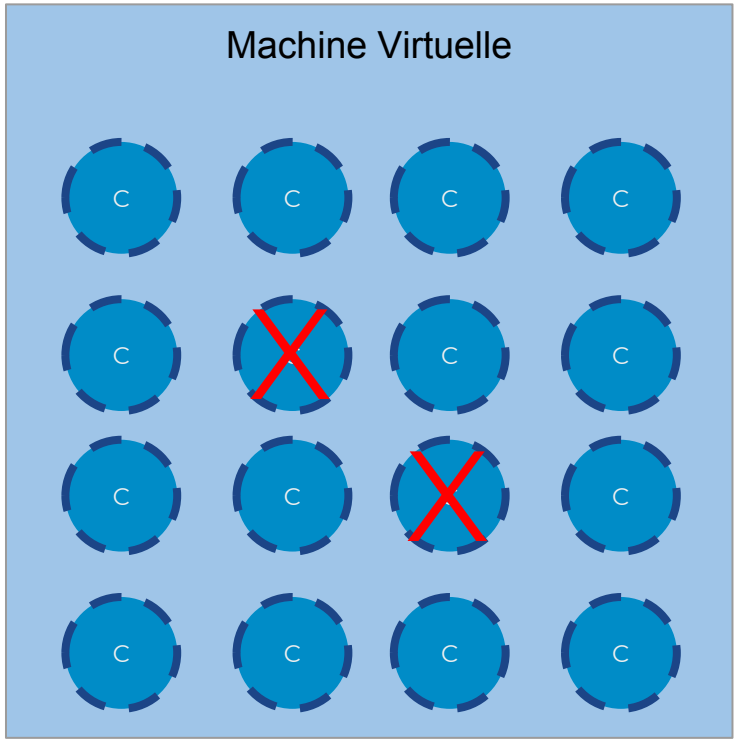
Pourquoi avoir besoin d’un orchestrateur ?

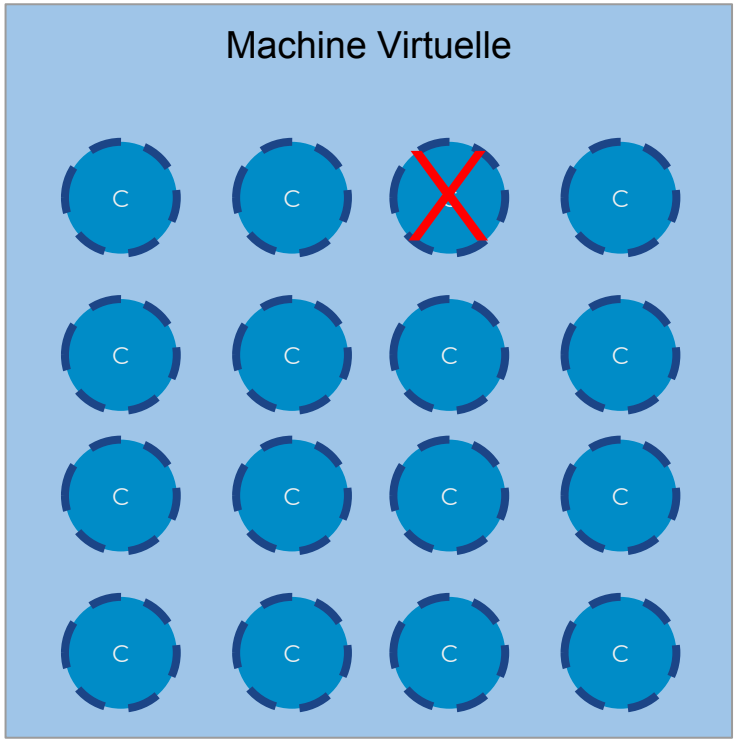
Pourquoi avoir besoin d'un Orchestrateur ?



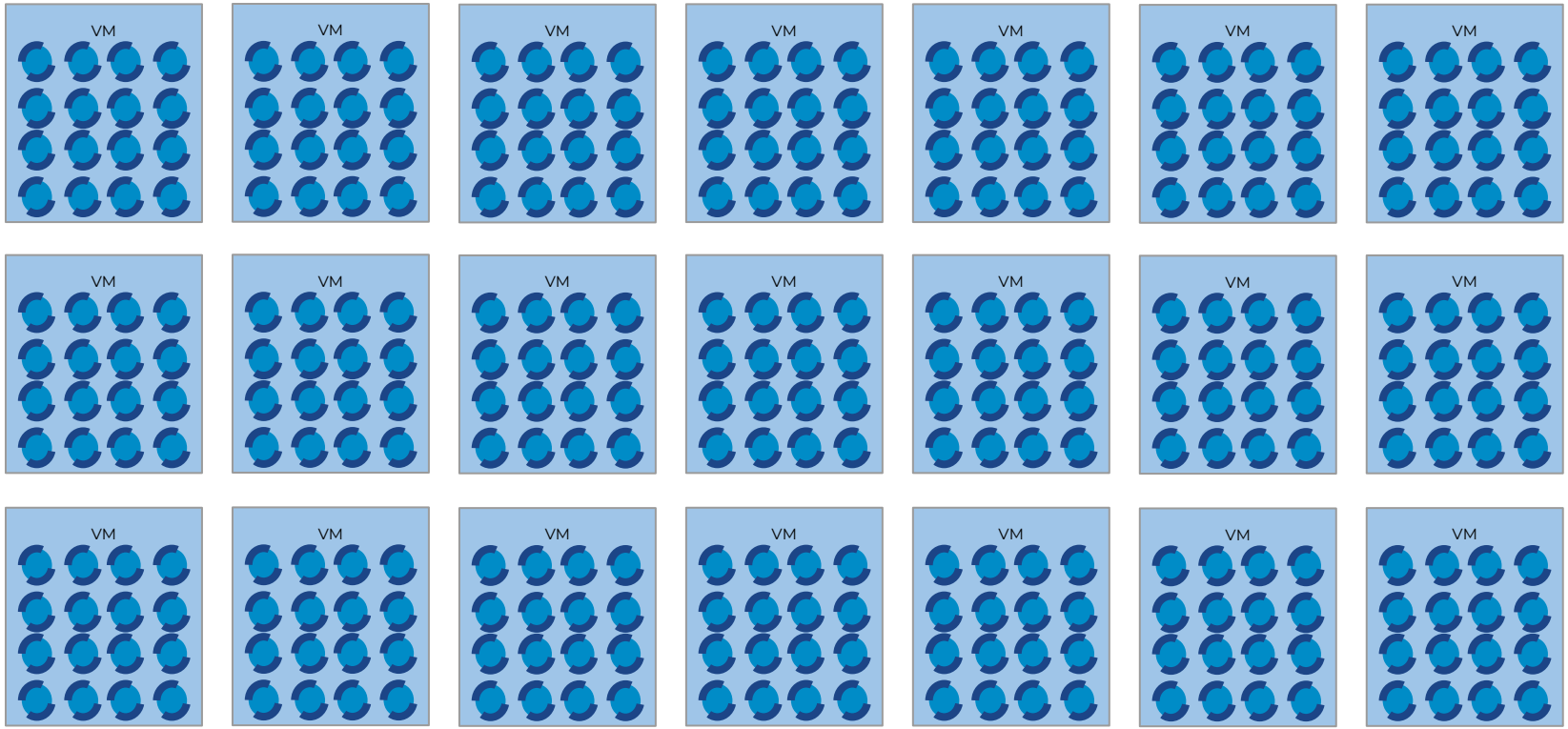


Pourquoi avoir besoin d'un Orchestrateur ?

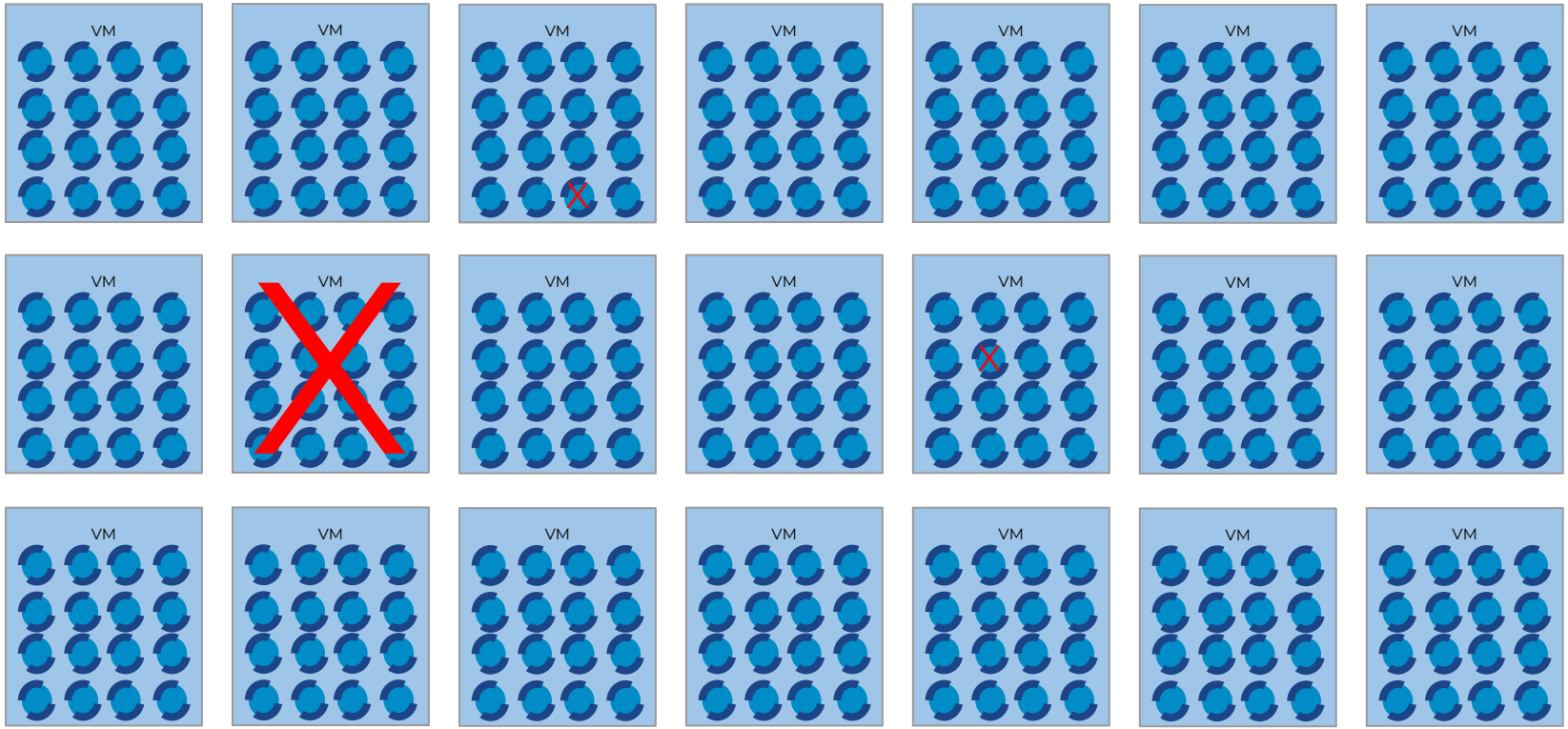




Pourquoi avoir besoin d'un Orchestrateur ?



Pourquoi avoir besoin d'un Orchestrateur ?



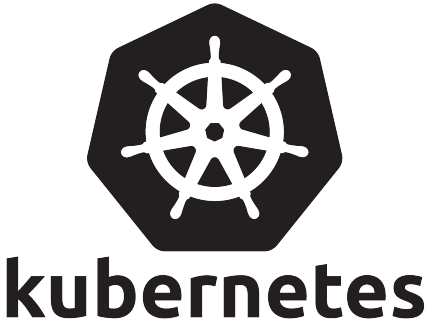
wifirst

4

Kubernetes

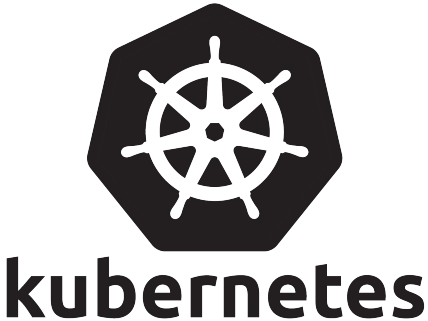
Kubernetes

- Kubernetes est un orchestrateur de container open source pour automatiser les déploiements, le scaling et le management d'applications conteneurisées.
- Inspiré par le système **BORG**. La version 1.0 est sortie en Juillet 2015
- Le code a été donnée à la Cloud Native Computing Foundation par Google



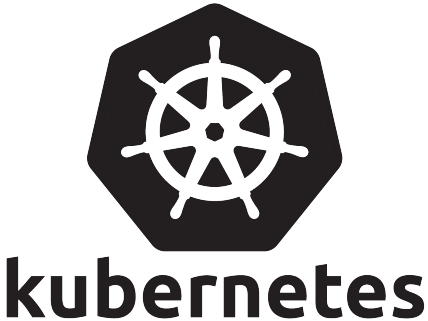
Lorsque vous voulez créer une ressource (ou primitive) dans un cluster Kubernetes, il est possible d'écrire un fichier de description en YAML permettant de décrire la ressource que **le cluster doit maintenir**

```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```



CLOUD NATIVE
COMPUTING FOUNDATION

- Kubernetes ne s'occupe pas du tout de la construction des images des containers
  - Les outils de CI doivent s'occuper de créer les images
- Kubernetes orchestre les éléments d'infrastructure mais il ne va pas fournir lui-même
  - Les machines ou machines virtuelles
  - Le stockage des containers
  - Le réseau des containers
  - L'environnement d'exécution
- Les administrateurs Kubernetes doivent donc fournir ces éléments avant que le cluster fonctionne correctement
  - Voir vos sysadmins si vous travaillez on-premise
  - Voir les offres Cloud comme GCP, Azure, AWS, OVH



CLOUD NATIVE
COMPUTING FOUNDATION

5

Les primitives

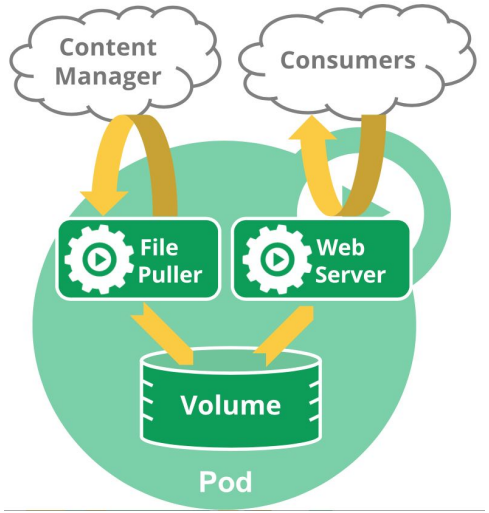
Namespace

- Un namespace est un espace de travail dans un cluster Kubernetes
- Il permet de créer un sorte de cluster Kubernetes virtuel dans un un cluster physique
- Dans un namespace, vous ne voyez que les containers qui sont dans le namespace
- Exemple de namespace :
    - Production
    - Preproduction
    - Staging
    - Dev

- ATTENTION: Par défaut, il n'y a pas d'isolation réseau entre les namespaces. En fonction de votre driver réseau, vous pouvez faire du filtrage réseau entre namespace.

Pods

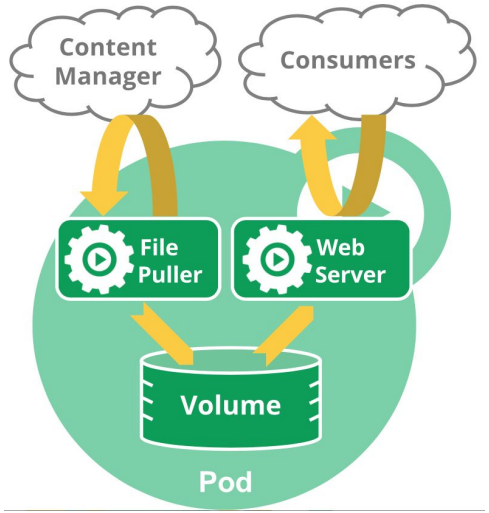
- Un pod est la plus petite unité géré par Kubernetes
- Curieusement, un pod n'est pas un container
- Dans le cas d'un pod multi-container, chaque container va partager les ressources fournit au pod, à savoir :
    - La stack réseau
    - La CPU et la mémoire



Pods

- Exemple de Pod Mono-container

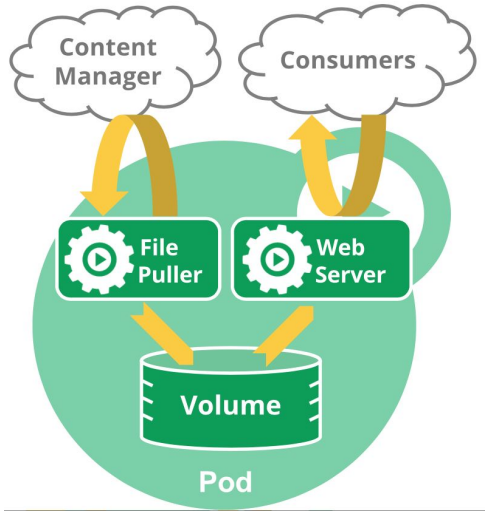
```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-mono
  labels:
    app: nginx_pod
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
      - containerPort: 80
```



Pods

- Exemple de Pod Multi-container

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-multi
  labels:
    app: nginx_mysql
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
      - containerPort: 80
  - name: mysql
    image: mysql:5.6
    env:
        -
          name: MYSQL_ROOT_PASSWORD
          value: password
    ports:
      - containerPort: 3306
```



Label et Selector

- Une bonne pratique est de mettre en place des labels dans les metadatas de chaque primitive
- Les labels vont permettre de sélectionner des ressources en fonction des labels

```
# kubectl get pods -l "app=nginx_pod"
NAME         READY   STATUS    RESTARTS   AGE
nginx-mono   1/1     Running   0          12h

# kubectl get pods -l "app=nginx_mysql"
NAME          READY   STATUS    RESTARTS   AGE
nginx-multi   2/2     Running   0          12h
```

ReplicaSets

- Un ReplicaSet est responsable de maintenir un nombre de pods identique sur l'ensemble du cluster et de le répartir sur les différentes machines du cluster
- En cas de crash d'un pod, le RS va le recréer directement

Deployment

- Un Deployment va permettre d'organiser un ensemble de pods et de replicaset afin de créer une application dans Kubernetes

- Grâce au Deployment, Kubernetes va pouvoir :
  - Maintenir l'état de votre Deployment en cas de crash des pods
  - Mettre à jour le nombre de replicas de votre deployment
  - Faire les rollings upgrade de vos applications (et les rollbacks)

Deployment

- Créer le deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Deployment

- Scale up - Scale Down d'un deployment

```
# kubectl scale --replicas=5 deployment/nginx-deployment

# kubectl scale --replicas=3 deployment/nginx-deployment
```

Deployment

- Rolling upgrade du deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.9.1
        ports:
        - containerPort: 80
```

Deployment

- Rollback du deployment

```
# kubectl rollout history deployment/nginx-deployment
deployment.extensions/nginx-deployment
REVISION  CHANGE-CAUSE
1         <none>
2         <none>

#kubectl describe deployment/nginx-deployment
[…]
Annotations:            deployment.kubernetes.io/revision: 2
[…]

#kubectl rollout undo deployment/nginx-deployment
deployment.extensions/nginx-deployment rolled back
```

Service

- Création d’un service

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 8081
    targetPort: 80
```

Storage

- Il est possible de créer des volumes de stockage persistent
  - En fonction de votre environnement vous avez accès à différents types de stockage (NFS,iSCSI,AWS ELB, Azure, GCP,....)

- Création d'un volume de stockage local

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

StatefulSet

- Grâce à la notion de stockage, il va être possible de créer des containers contenant des applications stateful comme une base de données

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  serviceName: "nginx"
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: task-pv-volume
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: task-pv-volume
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi
      storageClassName: manual
```

Jobs et Cronjob

- Il est possible de créer des jobs qui seront exécutés dans un container

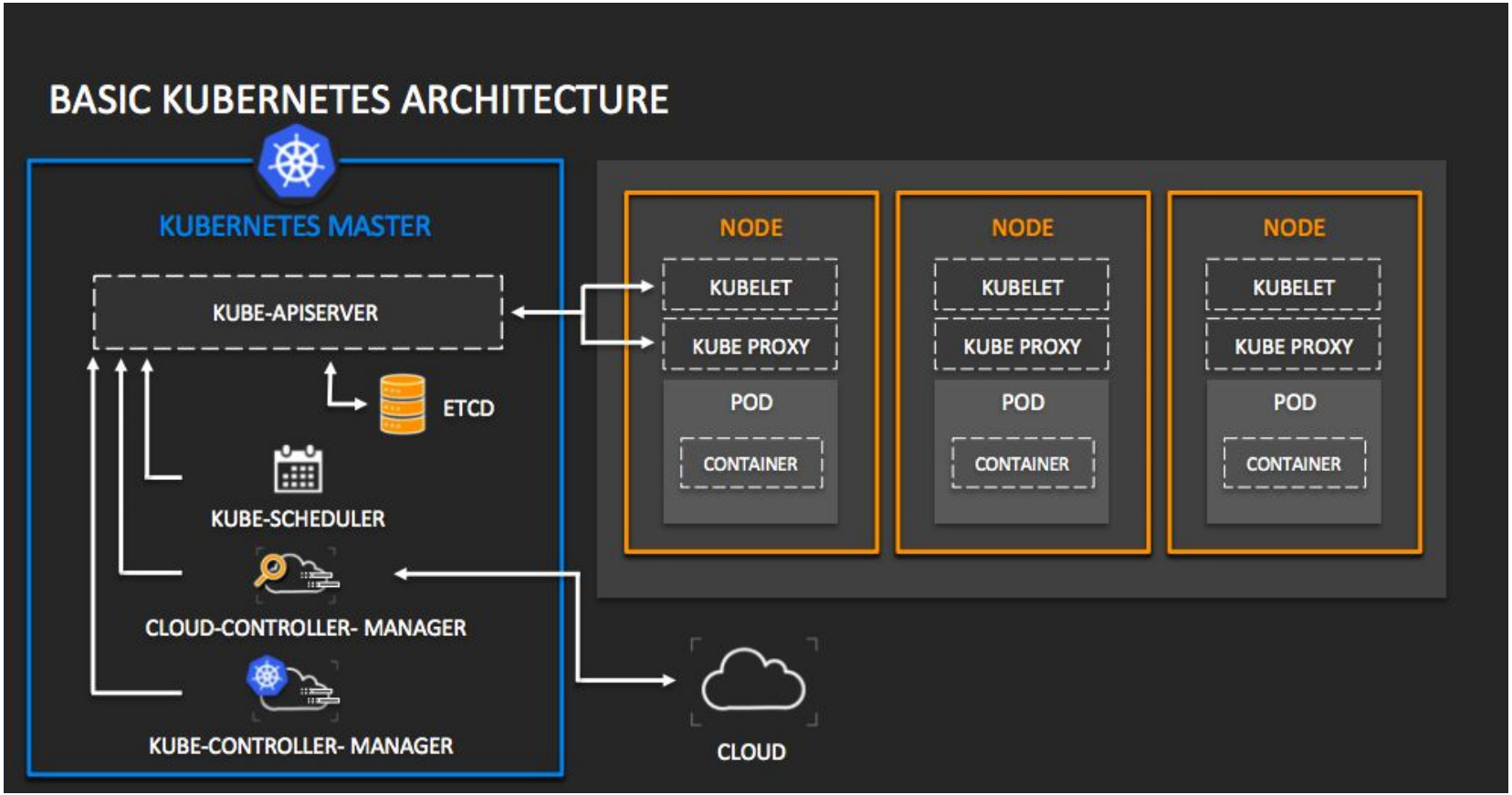
```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(20000)"]
      restartPolicy: Never
  backoffLimit: 4
```

- De la même manière, il est possible de définir des jobs qui vont tourner à intervalle régulier comme un crontab

6

Architecture de Kubernetes

Architecture Kubernetes



7

Pour aller plus loin ?

Ressources

- Doc officel : https://kubernetes.io/docs/home/
- Online Training: https://kubernetes.io/docs/tutorials/online-training/overview/

Merci

www.wifirst.fr blog.wifirst.fr www.facebook.com/wifirst www.twitter.com/wifirst