KUBERNETES-USER

CONCERNANT CES SUPPORTS DE COURS

SUPPORTS DE COURS RÉALISÉS PAR OSONES

https://osones.com

Logo Osones

- Sources : https://github.com/Osones/formations/
- HTML/PDF : https://osones.com/formations/

Licence Creative Commons BY-SA 4.0

KUBERNETES : PROJET, GOUVERNANCE ET COMMUNAUTÉ

KUBERNETES

- COE développé par Google, devenu open source en 2014
- Adapté à tout type d'environnement
- Devenu populaire en très peu de temps
- Premier projet de la C.N.C.F

CNCF

The Foundation's mission is to create and drive the adoption of a new computing paradigm that is optimized for modern distributed systems environments capable of scaling to tens of thousands of self healing multi-tenant nodes.

CNCF : PRÉREQUIS

- Distribuer sous forme de conteneurs
- Gestion dynamique de la configuration
- Orienté micro services

CNCF : LES RÔLES

- Intendance des projets
- Faire grossir et évoluer l'écosystème
- Rendre la technologie accessible
- Promouvoir la technologie

OCI

- Créé sous la Linux Fondation
- But : Créer un standard Open Source concernant la manière de "runner" et le format des conteneurs et images
- Non lié à des produits
- Non lié à des COE
- runC a été donné par Docker à l'OCI comme implémentions de base

KUBERNETES : PROJET

- Docs : https://kubernetes.io/docs/
- Slack : http://slack.k8s.io/
- Discuss : https://discuss.kubernetes.io
- Stack Overflow : https://stackoverflow.com/questions/tagged/kubernetes
- Serverfault https://stackoverflow.com/questions/tagged/kubernetes

KUBERNETES : PROJET

- Hébergé sur Github : https://github.com/kubernetes/kubernetes :
  - Issues : https://github.com/kubernetes/kubernetes/issues
  - Pull Requests https://github.com/kubernetes/kubernetes/pulls
  - Releases : https://github.com/kubernetes/kubernetes/releases
- Projets en incubation :
  - https://github.com/kubernetes-incubator/ (Deprécié)
  - https://github.com/kubernetes-sigs/

KUBERNETES : CYCLE DE DÉVELOPPEMENT

- Chaque *release* a son propre planning, pour exemple : https://github.com/kubernetes/sig-release/tree/master/releases/release-1.12#timeline
- Chaque cycle de développement dure 12 semaines et peut être étendu si nécessaire
- Features freeze
- Code Freeze
- Alpha Release
- Beta Releases
- Release Candidates

KUBERNETES : COMMUNAUTÉ

- Contributor and community guide : https://github.com/kubernetes/community/blob/master/READ community
- Décomposée en [Special Interest Groups] : https://github.com/kubernetes/community/blob/master/sig-lis
- Les SIG sont des projets, centres d'intérêts ou Working Group d
  - Network
  - Docs
  - AWS
  - etc
- Chaque SIG peut avoir des guidelines différentes.

KUBERNETES : ARCHITECTURE

KUBERNETES : COMPOSANTS

- Kubernetes est écrit en Go, compilé statiquement.
- Un ensemble de binaires sans dépendance
- Faciles à conteneuriser et à packager
- Peut se déployer uniquement avec des conteneurs sans dépendance d'OS

KUBERNETES : COMPOSANTS

- kube-apiserver : API server qui permet la configuration d'objet Kubernetes (Pod, Service, Deployment, etc.)
- kube-proxy : Permet le forwarding TCP/UDP et le load balancing entre les services et les backend (Pods)
- kube-scheduler : Implémente les fonctionnalités de scheduling
- kube-controller-manager : Responsable de l'état du cluster, boucle infinie qui régule l'état du cluster afin d'atteindre un état désiré

KUBERNETES : COMPOSANTS

- kubelet : Service "agent" fonctionnant sur tous les nœuds et assure le fonctionnement des autres services
- kubectl : Ligne de commande permettant de piloter un cluster Kubernetes

KUBERNETES : KUBELET

- Service principal de Kubernetes
- Permet à Kubernetes de s'auto configurer :
  - Surveille un dossier contenant les *manifests* (fichiers YAML des différents composant de Kubernetes).
  - Applique les modifications si besoin (upgrade, rollback).
- Surveille l'état des services du cluster via l'API server (*kube-apiserver*).

KUBERNETES : KUBE-APISERVER

- Les configurations d'objets (Pods, Service, RC, etc.) se font via l'API server
- Un point d'accès à l'état du cluster aux autres composants via une API REST
- Tous les composants sont reliés à l'API server

KUBERNETES : KUBE-SCHEDULER

- Planifie les ressources sur le cluster
- En fonction de règles implicites (CPU, RAM, stockage disponible, etc.)
- En fonction de règles explicites (règles d'affinité et anti-affinité, labels, etc.)

KUBERNETES : KUBE-PROXY

- Responsable de la publication de services
- Utilise *iptables*
- Route les paquets à destination des PODs et réalise le load balancing TCP/UDP

KUBERNETES : KUBE-CONTROLLER-MANAGER

- Boucle infinie qui contrôle l'état d'un cluster
- Effectue des opérations pour atteindre un état donné
- De base dans Kubernetes : replication controller, endpoints controller, namespace controller et serviceaccounts controller

KUBERNETES : KUBELET

```
root@ubuntu-xenial:~# ls -lh /etc/kubernetes/manifests/
total 16K
-rw------- 1 root root 2.0K Sep 23 20:04 etcd.yaml
-rw------- 1 root root 3.2K Sep 23 20:04 kube-apiserver.yaml
-rw------- 1 root root 2.5K Sep 23 20:04 kube-controller-manager.yaml
-rw------- 1 root root 1.1K Sep 23 20:04 kube-scheduler.yaml
```

KUBERNETES : NETWORK-POLICY-CONTROLLER

- Implémente l'objet NetworkPolicy
- Contrôle la communication entre les Pods
- Externe à Kubernetes et implémenté par la solution de Networking choisie :
  - Calico : https://projectcalico.org/
  - Flannel : https://coreos.com/flannel
  - Romana : https://romana.io/
  - Weave : https://www.weave.works/oss/net/
  - more : https://kubernetes.io/docs/concepts/cluster-administration/networking/#how-to-implement-the-kubernetes-networking-model

KUBERNETES : AUJOURD'HUI

- Version 1.11.x : stable en production
- Solution complète et une des plus utilisées
- Éprouvée par Google
- S'intègre parfaitement à d'autres *Container Runtime Interfaces (CRI)* comme containerd, cri-o, rktlet, frakti, etc...

KUBERNETES : CONCEPTS ET OBJETS

KUBERNETES : API RESOURCES

- Pods
- Deployments
- DaemonSets
- Services
- Namespaces
- Ingress
- NetworkPolicy
- Volumes

KUBERNETES : POD

- Ensemble logique composé de un ou plusieurs conteneurs
- Les conteneurs d'un pod fonctionnent ensemble (instanciation et destruction) et sont orchestrés sur un même hôte
- Les conteneurs partagent certaines spécifications du Pod :
  - La stack IP (network namespace)
  - Inter-process communication (PID namespace)
  - Volumes
- C'est la plus petite et la plus simple unité dans Kubernetes

KUBERNETES : POD

- Les Pods sont définis en YAML comme les fichiers docker-compose :

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

KUBERNETES : DEPLOYMENT

- Permet d'assurer le fonctionnement d'un ensemble de Pods
- Version, Update et Rollback
- Souvent combiné avec un objet de type *service*

KUBERNETES : DEPLOYMENT

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:stable
        ports:
        - containerPort: 80
```

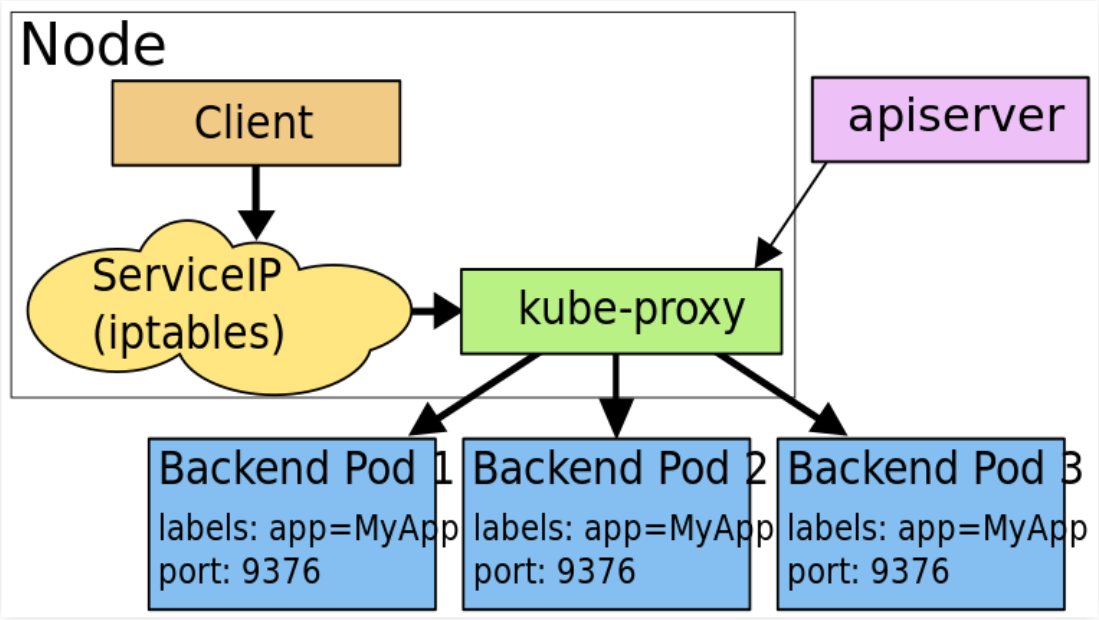
KUBERNETES : SERVICES

- Abstraction des Pods et Replication Controllers, sous forme d'une VIP de service
- Rendre un ensemble de Pods accessibles depuis l'extérieur
- Load Balancing entre les Pods d'un même service

KUBERNETES : SERVICES

- Load Balancing : intégration avec des cloud provider :
  - AWS ELB
  - GCP
  - Azure Kubernetes Service
  - OpenStack
- NodePort : chaque noeud du cluster ouvre un port statique et redirige le trafic vers le port indiqué
- ClusterIP : IP dans le réseau privé Kubernetes (VIP)
- LoadBalancer : expose le service à l'externe en utilisant le loadbalancer d'un cloud provider (AWS, Google, Azure)
- ExternalIP: le routage de l'IP publique vers le cluster est manuel

KUBERNETES : SERVICES



KUBERNETES : SERVICES

- Exemple de service (on remarque la sélection sur le label et le mode d'exposition):

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  type: NodePort
  ports:
  - port: 80
  selector:
    app: guestbook
    tier: frontend
```

KUBERNETES : SERVICES

Il est aussi possible de mapper un service avec un nom de domaine en spécifiant le paramètre spec.externalName.

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
  namespace: prod
spec:
  type: ExternalName
  externalName: my.database.example.com
```

KUBERNETES: INGRESS

- L'objet Ingress permet d'exposer un service à l'extérieur d'un cluster Kubernetes
- Il permet de fournir une URL visible permettant d'accéder un Service Kubernetes
- Il permet d'avoir des terminations TLS, de faire du *Load Balancing*, etc...

KUBERNETES : INGRESS

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: osones
spec:
  rules:
  - host: blog.osones.com
    http:
      paths:
      - path: /
        backend:
          serviceName: osones-nodeport
          servicePort: 80
```

KUBERNETES : INGRESS CONTROLLER

Pour utiliser un Ingress, il faut un Ingress Controller. Il existe plusieurs offres sur le marché :

- Traefik : https://github.com/containous/traefik
- Istio : https://github.com/istio/istio
- Linkerd : https://github.com/linkerd/linkerd
- Contour : https://www.github.com/heptio/contour/
- Nginx Controller : https://github.com/kubernetes/ingress-nginx

KUBERNETES : DAEMONSET

- Assure que tous les noeuds exécutent une copie du pod sur tous les noeuds du cluster
- Ne connaît pas la notion de replicas.
- Utilisé pour des besoins particuliers comme :
  - l'exécution d'agents de collection de logs comme fluentd ou logstash
  - l'exécution de pilotes pour du matériel comme nvidia-plugin
  - l'exécution d'agents de supervision comme NewRelic agent, Prometheus node exporter

NB : kubectl ne peut pas créer de DaemonSet

KUBERNETES : DAEMONSET

```
apiVersion: apps/v1beta2
kind: DaemonSet
metadata:
  name: ssd-monitor
  spec:
  selector:
    matchLabels:
      app: ssd-monitor
  template:
    metadata:
      labels:
        app: ssd-monitor
  spec:
    nodeSelector:
    disk: ssd
    containers:
    - name: main
      image: luksa/ssd-monitor
```

KUBERNETES : STATEFULSET

- Similaire au Deployment
- Les pods possèdent des identifiants uniques.
- Chaque replica de pod est créé par ordre d'index
- Nécessite un Persistent Volume et un Storage Class.
- Supprimer un StatefulSet ne supprime pas le PV associé

KUBERNETES : STATEFULSET

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
```

KUBERNETES : LABELS

- Système de clé/valeur
- Organiser les différents objets de Kubernetes (Pods, RC, Services, etc.) d'une manière cohérente qui reflète la structure de l'application
- Corréler des éléments de Kubernetes : par exemple un service vers des Pods

KUBERNETES : LABELS

- Exemple de label :

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

KUBERNETES : LABELS

- La commande kubectl get pods, par défaut, ne liste pas les labels. Il est possible de les voir en utilisant --show-labels:

```
$ kubectl get pods --show-labels
NAME     READY    STATUS    RESTARTS   AGE      LABELS
nginx    1/1      Running   0          31s      app=nginx,env=prod
```

KUBERNETES : NAMESPACES

- Fournissent une séparation logique des ressources par exemple :
  - Par utilisateurs
  - Par projet / applications
  - Autres...
- Les objets existent uniquement au sein d'un namespace donné
- Évitent la collision de nom d'objets

KUBERNETES : CRÉATION D'OBJETS KUBERNETES

- Les objets Kubernetes sont créés sous la forme de fichiers JSON ou YAML et envoyés à l'APIServer
- Possible d'utiliser la commande `kubectl run`, mais limitée aux `Deployments` et aux `Jobs`
- L'utilisation de fichiers YAML permet de les stocker dans un système de contrôle de version comme git, mercurial, etc...
- La documentation de référence pour l'API Kubernetes https://kubernetes.io/docs/reference/#api-reference

KUBERNETES : CRÉATION D'OBJETS KUBERNETES

- Pour créer un object Kubernetes depuis votre fichier YAML, utilisez la commande kubectl create :

```
kubectl create -f object.yaml
```

- Il est possible de créer des objets Kubernetes à partir d'une URL :

```
kubectl create -f https://raw.githubusercontent.com/kubernetes/examples/master/gue
```

KUBERNETES : CRÉATION D'OBJETS KUBERNETES

- Pour les supprimer exécuter simplement :

```
kubectl delete -f object.yaml
```

- Mettre à jour un objet Kubernetes en écrasant la configuration existante:

```
kubectl replace -f object.yaml
```

KUBERNETES : STOCKAGE

KUBERNETES : VOLUMES

- Fournir du stockage persistent aux PODs
- Fonctionnent de la même façon que les volumes Docker pour les volumes hôte :
  - EmptyDir ~= volumes docker
  - HostPath ~= volumes hôte
- Support de multiples backend de stockage :
  - GCE : PD
  - AWS : EBS
  - GlusterFS / NFS
  - Ceph
  - iSCSI

KUBERNETES : VOLUMES

- On déclare d'abord le volume et on l'affecte à un service :

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: redis
spec:
  containers:
  - name: redis
    image: redis
    volumeMounts:
    - name: redis-persistent-storage
      mountPath: /data/redis
  volumes:
  - name: redis-persistent-storage
    emptyDir: {}
```

KUBERNETES : STORAGE CLASS

- Permet de définir les différents types de stockage disponibles
- Utilisé par les Persistent Volumes pour solliciter un espace de stockage au travers des Persistent Volume Claims

KUBERNETES : STORAGE CLASS

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  zones: us-east-1d, us-east-1c
  iopsPerGB: "10"
```

KUBERNETES : PERSISTENTVOLUMECLAIMS

- Ressource utilisée et vue comme une requête pour solliciter du stockage persistant
- Offre aux PV une variété d'options en fonction du cas d'utilisation
- Utilisé par les StatefulSets pour solliciter du stockage (Utilisaltion du champ volumeClaimTemplates)

KUBERNETES : PERSISTENTVOLUMECLAIMS

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
    name: storage-claim
spec:
    accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: "slowl"
  selector:
    matchLabels:
      release: "stable"
    matchExpressions:
      - {key: capacity, operator: In, values: [10Gi, 20Gi]}
```

KUBERNETES : PERSISTENTVOLUME

- Composant de stockage dans le cluster kubernetes
- Stockage externe aux noeuds du cluster
- Cycle de vie d'indépendant du pod qui le consomme
- Peut être provisionné manuellement par un administrateur ou dynamiquement grâce un StorageClass

KUBERNETES : PERSISTENTVOLUME

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: persistent-volume-1
spec:
  storageClassName: slow
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/tmp/data"
```

KUBERNETES : JOBS

KUBERNETES : JOB

- Crée des pods et s'assurent qu'un certain nombre d'entre eux se terminent avec succès.
- Peut éxécuter plusieurs pods en parallèle
- Si un noeud du cluster est en panne, les pods sont reschedulés vers un autre noeud.

KUBERNETES : JOB

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  parallelism: 1
  completions: 1
  template:
    metadata:
      name: pi
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: OnFailure
```

KUBERNETES: CRON JOB

- Un CronJob permet de lancer des Jobs de manière planifiée.
- la programmation des Jobs se définit au format Cron
- le champ jobTemplate contient la définition de l'application à lancer comme Job.

KUBERNETES : CRONJOB

```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: batch-job-every-fifteen-minutes
spec:
  schedule: "0,15,30,45 * * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
            app: periodic-batch-job
        spec:
          restartPolicy: OnFailure
          containers:
          - name: pi
            image: perl
            command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
```

KUBERNETES : MINIKUBE

- Outil permettant de démarrer rapidement un cluster mono-node Kubernetes localement
- Execute Kubernetes dans une machine virtuelle
- Nécessite des outils de virtualisation (VirtualBox, VMware Fusion, KVM, etc...)
- Supporte plusieurs systèmes d'exploitation : Linux, Mac OS, Windows
- Installation : https://github.com/kubernetes/minikube#Installation

KUBERNETES : MINIKUBE

- Création d'un cluster Kubernetes

```
$ minikube get-k8s-versions
The following Kubernetes versions are available when using the localkube bootstrapp
  - v1.10.0
  - v1.9.4
  - v1.9.0
  - v1.8.0
  - v1.7.5
  - v1.7.4
```

KUBERNETES : MINIKUBE

```
$minikube start --kubernetes-version="v1.11.1"
Starting local Kubernetes v1.11.1 cluster...
Starting VM...
Getting VM IP address...
[...]
Connecting to cluster...
Setting up kubeconfig...
Starting cluster components...
Kubectl is now configured to use the cluster.
Loading cached images from config file.
```

KUBERNETES : MINIKUBE

- Effectuer un diagnostic basique du cluster

```
$ kubectl version
Client Version: v1.11.1
Server Version: v1.11.1
```

KUBERNETES : MINIKUBE

```
$ kubectl get componentstatuses
NAME                 STATUS    MESSAGE             ERROR
controller-manager   Healthy     ok
scheduler            Healthy     ok
etcd-0               Healthy     {"health": "true"}
```

KUBERNETES : KUBECTL

- Le seul (ou presque) outil pour interagir avec des clusters Kubernetes
- Utilise un fichier de configuration pour communiquer avec l'API de Kubernetes
- Le(s) fichier(s) se trouve(nt) par défaut dans `~/.kube/config`
- Le fichier de config. contient :
  - L'adresse(URI) de l'APIServer
  - Les chemins des certificats TLS utilisés pour l'authentification
- Fichier `kubeconfig` peut être passé en paramètre de kubectl avec le *flag* `--kubeconfig`

KUBERNETES : KUBECTL

- Afficher la liste des ressources API supportées par le serveur:

```
$ kubectl api-resources
NAME                              SHORTNAMES   APIGROUP                       NAMESPACED   KIND
bindings                                                                      true         Binding
componentstatuses                 cs                                          false        ComponentStatus
configmaps                        cm                                          true         ConfigMap
endpoints                         ep                                          true         Endpoints
events                            ev                                          true         Event
limitranges                       limits                                      true         LimitRange
namespaces                        ns                                          false        Namespace
nodes                             no                                          false        Node
persistentvolumeclaims            pvc                                         true         PersistentVolumeClaim
persistentvolumes                 pv                                          false        PersistentVolume
pods                              po                                          true         Pod
```

KUBERNETES : KUBECTL

- Afficher les noeuds du cluster :

```
kubectl get nodes
```

- Ces commandes sont équivalentes:

```
kubectl get no
kubectl get nodes
```

KUBERNETES : KUBECTL

- Afficher les *namespaces*

```
kubectl get ns
kubectl get namespaces
```

- Par défaut, kubectl utilise le *namespace* default
- Il est possible de sélectionner un *namespace* avec l'option -n ou --namespace

```
kubectl -n kube-system get pods
```

KUBERNETES : KUBECTL

- Afficher les pods (pour le namespace *default*)

```
kubectl get pods
kubectl get pod
```

KUBERNETES : KUBECTL

- Afficher les services (pour le *namespace* default):

```
kubectl get services
kubectl get svc
```

KUBERNETES : CONFIGMAPS

- Objet Kubernetes permettant stocker séparer les fichiers de configuration
- Il peut être créé d'un ensemble de valeurs ou d'un fichier resource Kubernetes (YAML ou JSON)
- Un ConfigMap peut sollicité par plusieurs pods

KUBERNETES : CONFIGMAPS

```
apiVersion: v1
data:
    redis-config: |
      maxmemory 2mb
      maxmemory-policy allkeys-lru
kind: ConfigMap
metadata:
  name: redis-config
  namespace: default
```

KUBERNETES : SECRETS

- Objet Kubernetes de type `secret` utilisé pour stocker des informations sensibles comme les mots de passe, les *tokens*, les clés SSH...
- Similaire à un `ConfigMap`, à la seule différence que le contenu des entrées présentes dans le champ `data` sont encodés en base64.
- Il est possible de directement créer un `Secret` spécifique à l'authentification sur un registre Docker privé.
- Il est possible de directement créer un `Secret` à partir d'un compte utilisateur et d'un mot de passe.

KUBERNETES : SECRETS

```
$ kubectl create secret docker-registry mydockerhubsecret \
--docker-username="employeeusername" --docker-password="employeepassword" \
--docker-email="employee.email@organization.com"
```

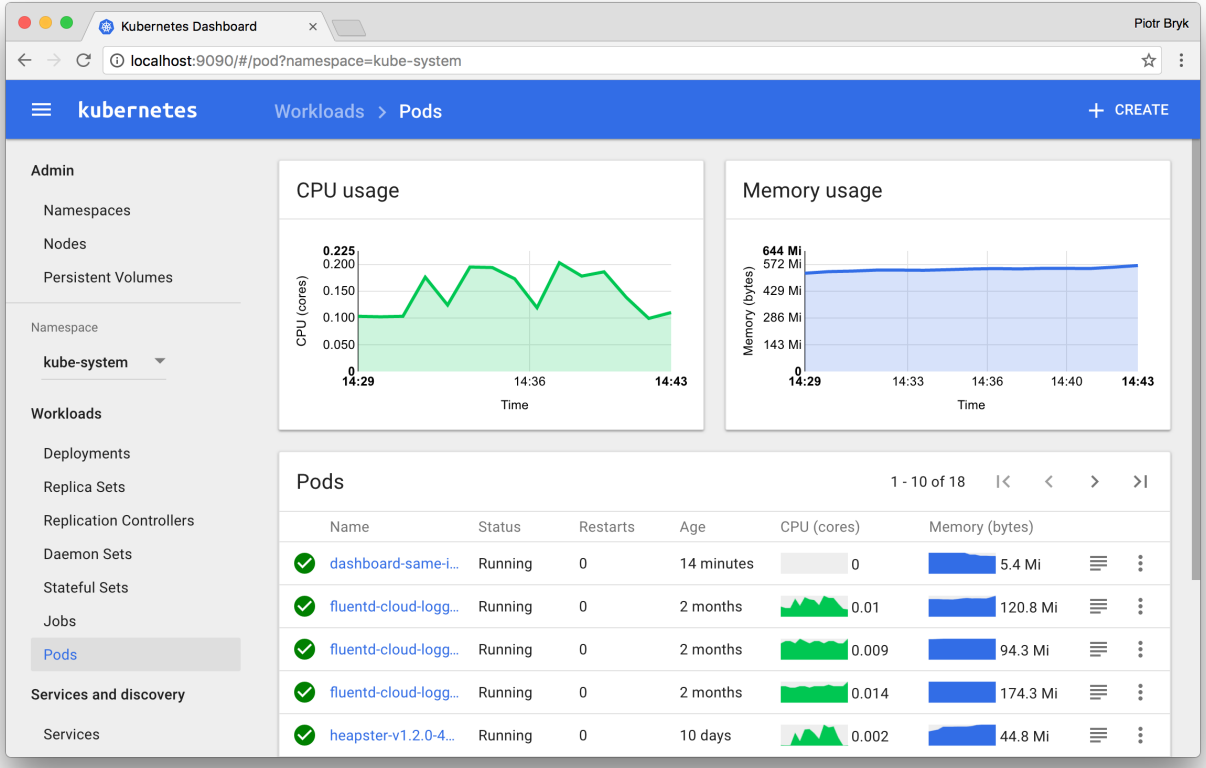
KUBERNETS : SECRETS

```
apiVersion: v1
kind: Pod
metadata:
  name: private-pod
  spec:
  imagePullSecrets:
  - name: mydockersecret
  containers:
  - image: privateregistry/privateimage:tag
    name: main
```

KUBERNETES : KUBERNETES DASHBOARD

- Interface graphique web pour les clusters Kubernetes
- Permet de gérer les différents objets Kubernetes créés dans le(s) cluster(s).
- Installé par défaut dans minikube

KUBERNETES : KUBERNETES DASHBOARD



KUBERNETES : KUBERNETES DASHBOARD

- Pour déployer le Dashboard, éxecuter la commande suivante:

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/master/sı
```

- Pour accéder au Dashboard, il faut établir une communication entre votre poste et le cluster Kubernetes :

```
$ kubectl proxy
```

- Now access Dashboard at:

http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/

KUBERNETES : INTRODUCTION TO HELM

- Une application conçue pour faciliter l'installation et la gestion des applications sur Kubernetes.
- Il utilise un format de paquetage appelé Charts.
- Il est comparable à apt/yum/homebrew.
- Plusieurs charts existent déjà sur le répertoire officiel : https://github.com/kubernetes/charts.

KUBERNETES : INTRODUCTION TO HELM

- Tiller : Serveur Helm. Il interagit avec l'APIServer de Kubernetes pour installer, mettre à jour et supprimer les ressources Kubernetes.
- Chart : Contient toutes les ressources définies et nécessaires pour éxecuter une application ou un service à l'intérieur de cluster Kubernetes. Un chart est pratiquement un regroupement de ressources Kubernetes pré-configurées.
- Release : Une instance d'un chart helm s'éxécutant dans un cluster Kubernetes.
- Repository : répertoire ou espace (public ou privé) où sont regroupés les charts.

KUBERNETES : INTRODUCTION TO HELM

- Installer Helm (sur une distribution Linux):
- Deployer tiller : `helm init`
- Voir la liste des `charts` disponibles sur les répertoire officiel : `helm search`
- Afficher la liste des `charts` disponibles pour *prometheus* : `helm search prometheus`
- Afficher les options disponibles dans un `chart` Helm: `helm inspect stable/prometheus`

CONCLUSION