Serge NOEL

Version : 2.2

Votre formation – Présentation

Public :

- Développeurs, architectes techniques, administrateurs et responsables d'exploitation et de production, chefs de projet.

Objectifs :

- Comprendre le positionnement de Kubernetes et la notion

Votre formation – Présentation

Pré-requis :

- Connaissances systèmes Linux/Windows
- Notions sur les réseaux TCP/IP
- Utilisation de la ligne de commande et du script Shell en environnement Linux.

Votre formation – Formateur

Son Nom :

- Serge NOEL

Qualifications :

- Consultant / Chef de projet depuis 1990.

Votre formation – Vous

Votre Nom

Votre fonction

Vos expériences

Vos attentes

Votre formation - Logistique

Horaires de la formation

- 9h00-12h30
- 14h00-17h30

Pauses

Votre formation - Programme

Introduction à Kubernetes

Les solutions d’installation

Les fichiers descriptifs

Architecture de Kubernetes

Objets de base

Exploiter Kubernetes

Gestion avancée des conteneurs

Kubernetes en production

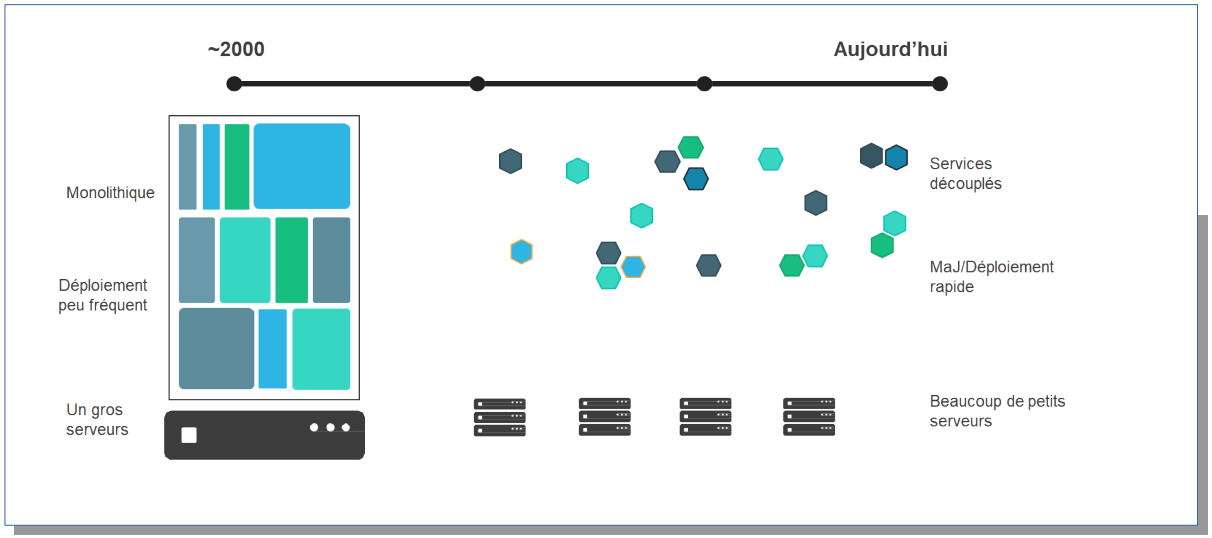
Introduction à Kubernetes

De la virtualisation à conteneurisation.

Le couple Docker/Kubernetes.

De la virtualisation à Docker

L’architecture des applications change rapidement



De la virtualisation à Docker

Analogie – origine du nom

De la virtualisation à Docker

Analogie – origine du nom

De la virtualisation à Docker

Solution: Des conteneurs Docker

- Package : une application et ses dépendances
- Isoler les applications les unes des autres
- Agnostique sur le contenu
- Création d'un standard pour le format des containers (OCI)

De la virtualisation à Docker

Du point de vue du développeur ...

- Build once… run anywhere
- un environnement portable pour l'exécution des apps
- Pas de risque d'oublier des dépendances, packages, … durant les déploiements
- Chaque application s'exécute dans son propre conteneur : avec ses propres versions des librairies
- Facilite l'automatisation des tests
- élimine les problèmes de compatibilité entres les plate-formes
- Coût ressource très bas pour lancer un container. On peut en lancer des dizaines sur un poste développeur (laptop)
- Permet de tester des technologies ou faire des prototypes rapidement et à très bas coût.

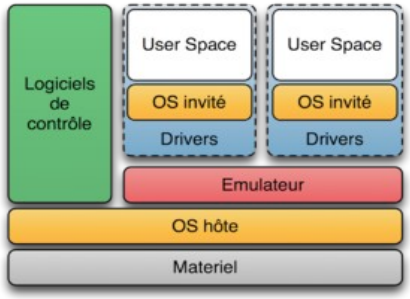
De la virtualisation à Docker

Du point de vue de l'admin sys ...

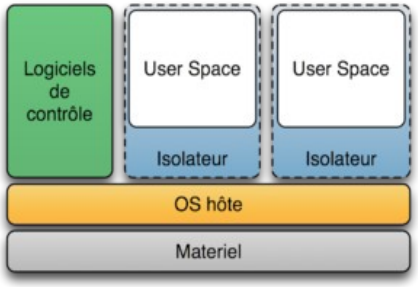
- Configure once…run anything
- Rend le workflow plus consistant, prédictible, répétable
- Élimine les inconsistances entre les environnements de dev/test/prod
- Améliore la rapidité et la fiabilité du déploiement continu (continuous

De la virtualisation à Docker

Comparaison



Virtualisation



Conteneurs

- Pas seulement Docker : terme générique désignant les technologies intégrées au noyau Linux depuis ~10 ans
- Les conteneurs sous Microsoft Windows Server depuis Windows 2016 serveur

De la virtualisation à Docker

~~Virtualisation~~ → Conteneurs

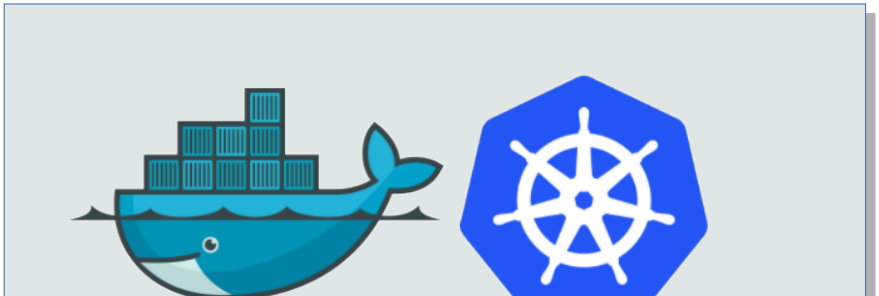
- Le système invité partage le noyau du système host
- Très performant : tout passe par le même noyau
- Peu coûteux : s'intègre au host, peut se partager

De la virtualisation à Docker

Qu'est-ce que Kubernetes ?

- Kubernetes est un système permettant d'exécuter et de coordonner des applications conteneurisées sur un cluster de machines.
- Il gère le cycle de vie des applications et services conteneurisés à l'aide de méthodes qui offrent prévisibilité, évolutivité et haute disponibilité.
- En tant qu'utilisateur de Kubernetes, vous pouvez :
  - définir comment vos applications doivent fonctionner
  - comment elles doivent pouvoir interagir avec d'autres applications ou avec le monde extérieur.
  - faire évoluer vos services vers le haut ou vers le bas
  - effectuer des mises à jour progressives
  - basculer le trafic entre les différentes versions de vos applications.

Le couple Docker/Kubernetes.



Le couple Docker/Kubernetes.

- Une des sociétés à avoir démocratisé l'utilisation des conteneurs Linux est **Docker**.
- **Docker** n'a pas créé la technologie : c'est un ensemble d'outils et d'API qui ont rendu les conteneurs beaucoup plus facilement gérables.
- Le succès de Docker est dû au fait qu'il est arrivé au bon moment, pile quand l'industrie cherchait un moyen de mieux gérer le Cloud et ses workloads Web.
- **Docker** est plus qu'une trousse à outils. La société a aussi fédéré tout un écosystème, foisonnant, qui a commencé à contribuer à un ensemble divers d'outils de gestion des cycles de vie des conteneurs.
- En juin 2014, la DockerCon, la première conférence jamais organisée sur le sujet, a par exemple pu se targuer de la présence d'acteurs majeurs comme Google, IBM, Microsoft, Amazon, Facebook, Twitter ou Rackspace, tous venus témoigner de leurs utilisations des conteneurs.

Le couple Docker/Kubernetes.

Comment Google fait tourner des workloads critiques dans des conteneurs ?

- Si Docker est dans la lumière des projecteurs, une autre entreprise – qui maîtrise l'art du dimensionnement – est passé maître dans l'art d'utiliser les conteneurs pour ses workloads de production. Cette société s'appelle **Google**.
- Google gère en effet chaque semaine plus de **deux milliards de conteneurs**. Des services comme Gmail, Search, Apps ou Maps tournent tous dans des conteneurs.
- Demandez à n'importe quel administrateur qui a eu à gérer plusieurs centaines de

Le couple Docker/Kubernetes.

Kubernetes : une nouvelle ère pour les containeurs

- Un des principaux outils que Google a rendu open-source s'appelle **Kubernetes** - qui signifie en grec *pilote* ou *barreur*.
- **Kubernetes fonctionne en complément de Docker**.
  - Docker permet de gérer le cycle de vie des conteneurs,
  - Kubernetes apporte l'orchestration et la gestion de clusters de conteneurs.
- Les clients qui provisionnent des VMs sur AWS, Azure ou sur n'importe quel Cloud Public, n'ont que faire de ce qui se passe du côté du hardware. Du moment qu'ils ont les VMs qu'ils souhaitent, avec les bonnes performances qui vont avec, ils ne se préoccupent pas de savoir comment cela marche « en dessous ». Savoir si les serveurs physiques d'AWS sont de chez HP, Dell ou IBM est sans importance.

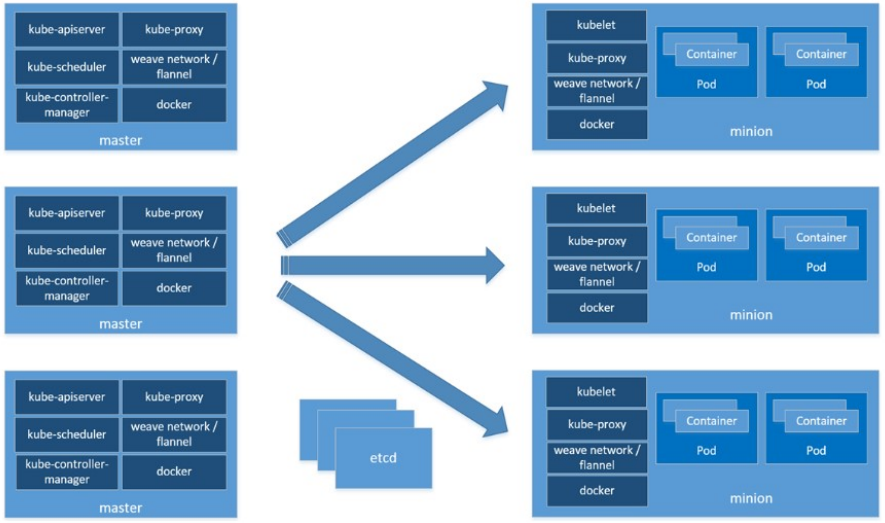
Solutions d'installation

minikube

- Un bon moyen de commencer à travailler avec Kubernetes, c'est d'utiliser Minikube.
- C'est un script qui va récupérer une machine virtuelle préconfigurée avec tous les composants nécessaires à Kubernetes, et l'installer sur votre machine.

Solutions d'installation

On-Premise



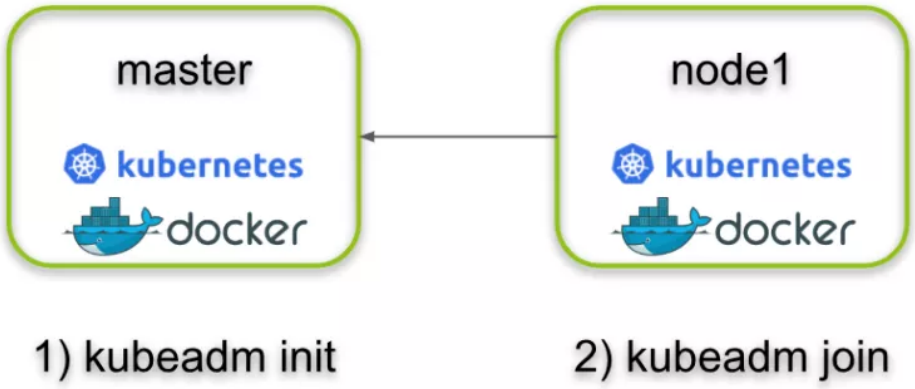
Solutions d'installation

Kubespray à la rescousse

- Depuis quelques mois, il existe une nouvelle manière

Solutions d'installation

Déploiement et publication manuelle.



Solutions d'installation

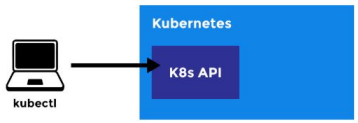
Installation et configuration de docker.

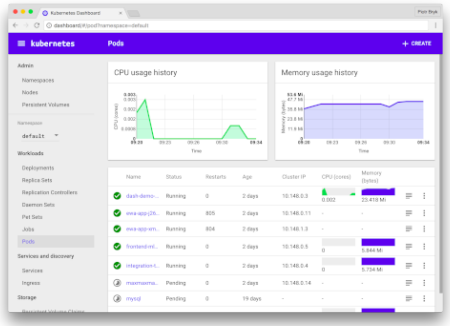
- Il convient d'installer en premier lieu Docker car ***K8s*** (Kubernetes) utilise Docker

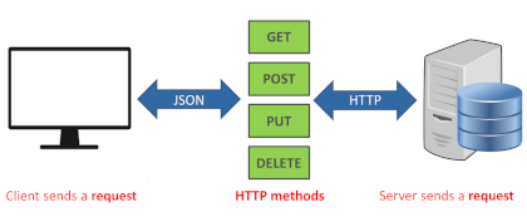
Accéder à Kubernetes

Accéder au cluster Kubernetes :

- CLI (kubectl),
- GUI (dashboard)
- APIs.

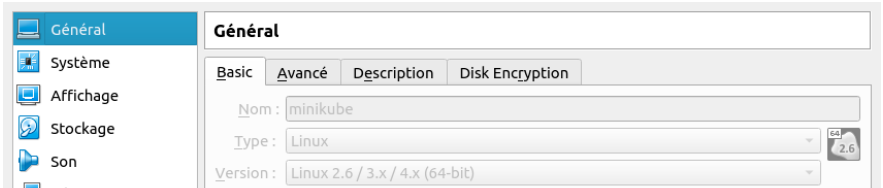






Solutions d'installation

Détail et introspection de minikube.



Premiers essais

Bonjour le monde en version Minikube :

- Une fois que votre cluster Kubernetes est en cours d'exécution et que **kubectl** est configuré, vous pouvez exécuter votre première application en quelques étapes. Cela peut être fait en utilisant *les commandes impératives* qui n'ont pas besoin de fichiers de configuration.
- Pour exécuter une application, vous devez fournir un nom de déploiement (nginx), l'emplacement de l'image du conteneur (easylinux/kubernetes:nginx) et le port (80)

Premiers essais

Exemple :

Si vous utilisez un cluster personnalisé Kubernetes (Minikube, kubeadm ou autre). Il n'y a pas de LoadBalancer integré (contrairement à AWS ou Google Cloud). Dans ce cas vous devez utiliser NodePort ou un controlleur Ingress.

Premiers essais

```
$ kubectl get services
NAME         TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
Nginx        LoadBalancer   10.97.48.179   <none>       80:31497/TCP      39s
kubernetes   ClusterIP      10.96.0.1      <none>        443/TCP          6d22h

$ kubectl describe service nginx
Name:                     nginx
Namespace:                default
Labels:                   app=nginx
Annotations:              <none>
Selector:                 app=nginx
Type:                     NodePort
IP:                       10.109.57.235
Port:                     <unset>  80/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  32209/TCP
Endpoints:                172.17.0.6:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>

$ minikube service nginx
Opening kubernetes service default/nginx in default browser...
```

Premiers essais

Scale Up

Une fois que cela a fonctionné, vous pouvez augmenter votre application avec:

```
$ kubectl scale deployments/nginx --replicas=4
```

Et vérifiez le résultat avec:

```
$ kubectl get deployments
NAME                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
```

Premiers essais

Suppression d'un déploiement.

Si une application n’est plus nécessaire, il suffit de la supprimer.

```
$ kubectl delete service backend
$ kubectl delete deployment backend
```

NB : pas d’équivalent à docker stop car il n’y a pas de sens en production à ‘suspendre’ une application

Travaux pratiques

Déploiement d'une plateforme de test.

- Validation du poste de travail
- Mise en œuvre Minikube
- Tests de fonctionnement

Les fichiers descriptifs

Syntaxe.

- **YAML**, acronyme de **Y**et **A**nother **M**arkup **L**anguage dans sa version 1.01. C'est un format de représentation de données par sérialisation Unicode.
- YAML a été proposé par Clark Evans en 2013, et implémenté par ses soins ainsi que par Brian Ingerson et Oren Ben-Kiki.
- Son objet est de représenter des informations plus élaborées que le simple CSV en gardant cependant une lisibilité presque comparable, et bien plus grande en tout cas que du XML.

Les fichiers descriptifs

Syntaxe.

- L'idée de YAML est que presque toute donnée peut être représentée par combinaison de listes, tableaux associatifs et données scalaires. YAML décrit ces formes de données (les représentations YAML), ainsi

Les fichiers descriptifs

Syntaxe.

- La syntaxe du flux YAML est relativement simple, efficace, moins verbeuse que du XML, moins compacte cependant que du CSV.
  - Les commentaires sont signalés par le signe dièse (#) et se prolongent sur toute la ligne. Si par contre le dièse apparaît dans une chaîne, il signifie alors un nombre littéral.
  - Il est possible d'inclure une syntaxe JSON dans une syntaxe YAML.
  - Les éléments de listes sont dénotés par le tiret (**-**), suivi d'une espace, à raison d'un élément par ligne.
  - Les tableaux sont de la forme clé: valeur, à raison d'un couple par ligne.
  - Les scalaires peuvent être entourés de guillemets doubles (**"**), ou simples (**'**), sachant qu'un guillemet s'échappe avec un antislash (****), alors qu'une apostrophe s'échappe avec une autre apostrophe.
  - L'indentation, par des **espaces**, manifeste une arborescence.

La syntaxe YAML se distingue de JSON par le fait qu'il se veut plus facilement lisible par une personne. Il se distingue du XML par le fait qu'il s'intéresse d'abord à la sérialisation de données, et moins à la documentation.

Les fichiers descriptifs

Syntaxe.

- Exemple :

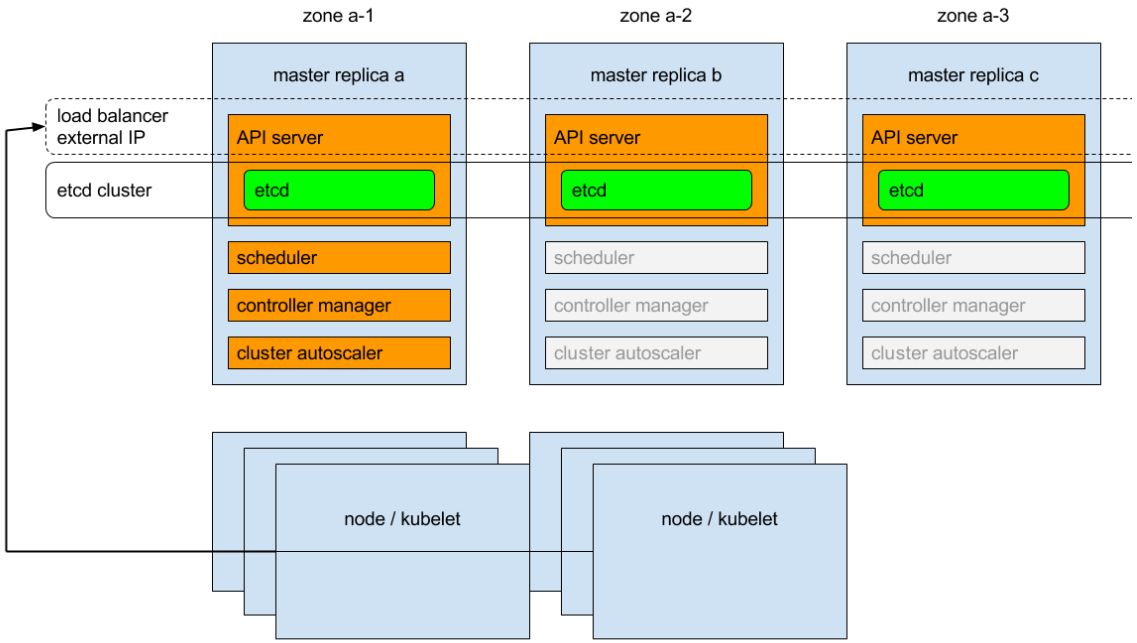
```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: petstore
  annotations:
    sidecar.istio.io/inject: "false"
spec:
  selector:
    matchLabels:
      app: petstore
    template:
      metadata:
```

Travaux pratiques

Un premier Pod

Architecture Kubernetes

Architecture Kubernetes



Architecture Kubernetes

Le Master :

- Le serveur maître sert de plan de contrôle primaire pour les clusters Kubernetes.
- Il sert de point de contact principal pour les administrateurs et les utilisateurs, et fournit également de nombreux systèmes à l'échelle du cluster pour les nœuds ‘worker’.

Architecture Kubernetes

Le Master : **etcd**

- Le projet **etcd** est une solution de stockage clé/valeurs légère et distribuée qui peut être configurée pour s'étendre sur plusieurs nœuds.
- Kubernetes utilise **etcd** pour stocker les données de configuration accessibles par chacun des nœuds du cluster.
- Il aide également à maintenir l'état du cluster grâce à des fonctions telles que l'élection du chef et le verrouillage distribué.
- Il fourni une API HTTP/JSON.
- **etcd** peut être configuré sur un seul serveur maître ou, dans les scénarios de production, réparti sur plusieurs machines

Architecture Kubernetes

Le Master : **kube-apiserver**

- Service maître le plus important. C’est un serveur **API**.
- C'est le principal point de gestion de l'ensemble du cluster car il permet à un utilisateur de configurer les charges de travail et les unités organisationnelles de Kubernetes.
- Il est responsable de s'assurer que le stockage **etcd** et les services actifs

Architecture Kubernetes

Le Master : **kube-controller-manager**

- Il gère les différents contrôleurs qui régulent l'état du cluster, gèrent les cycles de vie des charges de travail et exécutent les tâches de routine.
- Lorsqu'un changement est détecté, le contrôleur lit les nouvelles informations et met en œuvre la procédure qui permet d'atteindre l'état souhaité. Cela peut impliquer la mise à l'échelle d'une application vers le haut ou vers le bas, l'ajustement des ‘endpoints’, etc.

Architecture Kubernetes

Le Master : **Kube-scheduler**

- Le processus qui dispatche les charges de travail à des nœuds spécifiques du cluster est **l'ordonnanceur**. Ce service
  - lit les exigences opérationnelles d'une charge de travail,
  - analyse l'environnement d'infrastructure actuel et

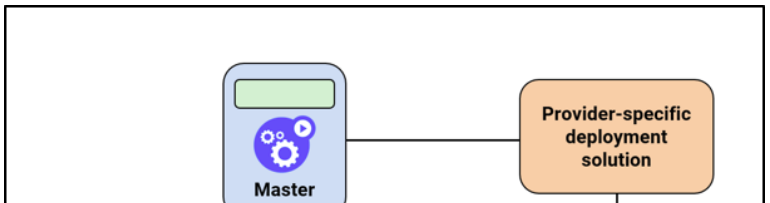
Architecture Kubernetes

Le Master : **cloud-controller-manager**

- Kubernetes peut être déployé dans de nombreux environnements différents.
- Les gestionnaires de contrôleurs de cloud agissent comme la colle qui permet à Kubernetes d'interagir avec des fournisseurs ayant des capacités, des fonctionnalités et des API différentes.

Architecture Kubernetes

Architecture d'un minion :



Architecture Kubernetes

Architecture d'un minion : **Runtime**

- La base de chaque nœud est un runtime de conteneur. Généralement, cette exigence est satisfaite en installant et en exécutant Docker, mais des alternatives comme rkt et runc sont également disponibles.
- Le runtime est responsable du démarrage et de la gestion des conteneurs, applications encapsulées dans un environnement d'exploitation relativement isolé mais léger.

Architecture Kubernetes

Architecture d'un minion : **kubelet**

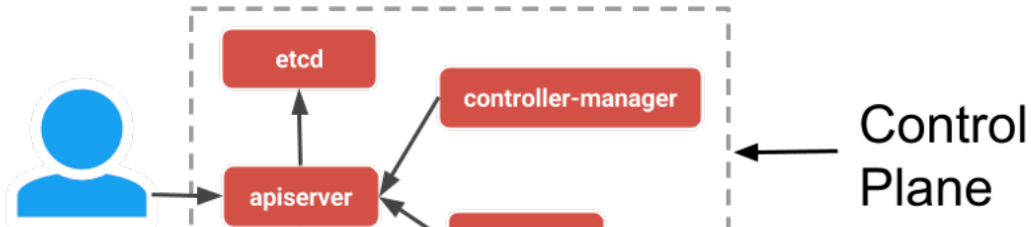
- Point de contact principal pour chaque nœud avec le cluster.
- Ce service est chargé de relayer les informations vers et depuis les services gestionnaires, ainsi que d'interagir avec **etcd** pour lire les détails de configuration ou écrire de nouvelles valeurs.
- **kubelet** communique avec les composants maîtres pour

Architecture Kubernetes

Architecture d'un minion : **kube-proxy**

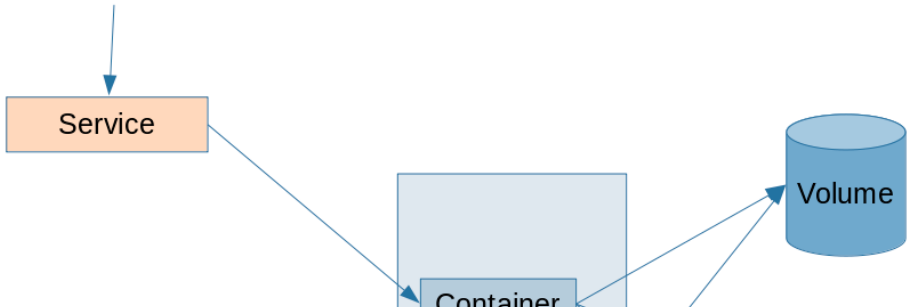
- Pour gérer les sous-réseaux d'hôtes individuels et rendre les services disponibles aux autres composants, un petit service proxy appelé **kube-proxy** est exécuté sur chaque minion. Ce processus achemine les demandes aux conteneurs appropriés, peut effectuer un équilibrage de charge primitif et est généralement chargé de s'assurer que l'environnement réseau est prévisible et accessible, mais isolé le cas échéant.

Architecture Kubernetes
Résumé



Objets de base

Objets de base



Objets de base

Pod

- Un **pod** est l'unité la plus basique avec lequel Kubernetes fonctionne. Les conteneurs eux-mêmes ne sont pas assignés à des hôtes. Au lieu de cela, un ou plusieurs conteneurs étroitement couplés sont encapsulés dans un objet appelé **pod**.
- Un **pod** représente généralement un ou plusieurs conteneurs qui doivent être contrôlés comme une seule application.
- Les pods se composent de conteneurs qui fonctionnent en étroite collaboration, ont un cycle de vie commun et doivent toujours être programmés sur le même nœud.
- Ils sont gérés comme une unité et partagent leur environnement, leurs volumes et leur espace IP.

Objets de base

Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: rss-site
  labels:
    app: rss
spec:
  containers:
```

Objets de base

Les PODs sont mortels, leur IP change, ...

Les PODs peuvent être ‘scalé’ (on augmente ou diminue leur nombre en fonction de la charge)

Il faut dès lors :

- Informer les autres containers
- Répartir la charge de façon adéquate

Objets de base

Deployment

Un Deployment est un des objets permettant de lancer des Pods. Dans les bonnes pratiques de Kubernetes il est encouragé d’utiliser des Deployments.

Lors de la création d’un Deployment le controller associé va créer un ReplicaSet à partir de votre configuration. Le controller associé au ReplicaSet va créer une série de Pod à partir de la configuration du ReplicaSet.

Objets de base

Deployment

Le Deployment est un objet stable depuis la version 1.9 de Kubernetes, disponible sur l'api "apps/v1".

NB :

Il ne faut pas gérer les ReplicaSets ou les pods appartenant à un déploiement.

Objets de base

Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: influx-app
  name: influxdb
spec:
  selector:
    matchLabels:
```

Objets de base

Service

Un pod doit communiquer avec l'extérieur, un service permet de 'publier' une application.

- **Port** : Port est le numéro de port qui rend un service visible aux autres services fonctionnant dans le même cluster K8s. En d'autres termes, si un service veut invoquer un autre service fonctionnant dans le même cluster Kubernetes, il pourra le faire en utilisant le port spécifié contre "port" dans le fichier de spécifications du service.
- **targetPort** : Le port cible est le port du POD sur lequel le service est exécuté.
- **Nodeport** : Node port est le port sur lequel le service peut être accédé à partir d'utilisateurs externes en utilisant Kube-Proxy. Normalement laissé à l'appréciation de Kubernetes (entre 30000 et 32000)

Objets de base

Service

```
apiVersion: v1
kind: Service
metadata:
  name: example-service
  labels:
    app: example-service
```

Objets de base

hostPath

Un volume hostPath monte un fichier ou un répertoire du système de fichiers du nœud hôte dans votre Pod.

Certaines utilisations d'un hostPath :

- exécuter un conteneur qui a besoin d'accéder au fonctionnement interne de Docker
  Ex :utiliser un chemin hôte de /var/lib/docker ou /var/run/docker.sock
- exécuter cAdvisor dans un conteneur, pour accéder à /sys
- permettre à un Pod de spécifier si un hostPath donné doit exister avant l'exécution du Pod, s'il doit être créé, et qu'il doit exister.

Objets de base

hostPath

En plus du chemin (obligatoire), l'utilisateur peut spécifier un type pour un volume hostPath.

Les valeurs prises en charge pour le type de zone sont :

Valeur	Description

Objets de base

hostPath

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /test-pd
      name: test-volume
  volumes:
  - name: test-volume
    hostPath:
      path: /data
      type: Directory
```

Objets de base

Les plugins de volume Kubernetes

Kubernetes propose beaucoup de plugins de volume :

- Stockage distant :
  - Disque persistant GCE
  - vSphere

Objets de base

PersistentVolume / persistentVolumeClaim / storageClass

Un PersistentVolume (PV) est un élément de stockage dans le cluster qui a été provisionné manuellement par un administrateur, ou dynamiquement provisionné par Kubernetes à l'aide d'une StorageClass.

Une PersistentVolumeClaim (PVC) est une demande de stockage par un utilisateur qui peut être satisfaite par un PersistentVolume.

PersistentVolumes et PersistentVolumeClaims sont indépendants du cycle de vie des Pods et préservent les données en redémarrant, reprogrammant et même en supprimant les Pods.

Objets de base

PersistentVolume / persistentVolumeClaim / storageClass

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  namespace: kube-system
  name: speed
  annotations:
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nextcloud-db-claim
  labels:
    app: nextcloud
spec:
```

Objets de base

PersistentVolume / persistentVolumeClaim / storageClass

```
apiVersion: apps/v1
kind: Deployment
…
spec:
…
  template:
…
    spec:
      containers:
…
        volumeMounts:
        - name: nextcloud-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: nextcloud-storage
        persistentVolumeClaim
          claimName: nextcloud-db-claim
```

Objets de base

Quel futur pour le stockage Kubernetes ?

Le stockage Kubernetes tend vers :

- Container Storage Interface (CSI)
  - Standardisation de plugins de volume block et fichier “Out-of-Tree”
- Stockage local

Objets de base

Secrets et configMap

Il est souvent nécessaire de référencer des données "spéciales", tels que des clés API, des jetons et d'autres secrets. L’application peut être paramétrable à l'aide de paramètres de configuration, par exemple, un fichier PHP.ini, ou des variables d'environnement.

Pour éviter de coder ces références en dur dans votre logique applicative. Kubernetes gère 2 types d’objets : **secrets** et **configMap**.

Objets de base

Secrets et configMap

```
apiVersion: v1
kind: Secret
metadata:
  name: next-conf
type: Opaque
data:
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  A_SPECIFIC_VAR: FOOBAR
```

Objets de base

Secrets et configMap

```
apiVersion: extensions/v1beta1
kind: Deployment
...
spec:
...
  template:
...
    spec:
      containers:
        - name: mypod
          image: easylinux/kubernetes:nginx
          volumeMounts:
            - mountPath: "/etc/mypassword"
              name: credential
      volumes:
        - name: credential
          secret:
            secretName: next-conf
```

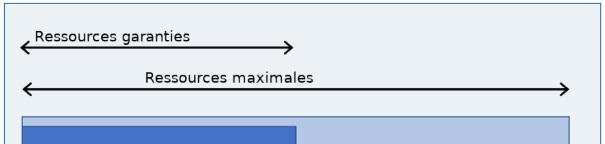
```
apiVersion: extensions/v1beta1
kind: Deployment
...
  template:
 ...
    spec:
      containers:
        - name: mypod
          image: easylinux/kubernetes:nginx
          env:
            - name: NEXT_DB
              valueFrom:
                secretKeyRef:
                  name: next-conf
                  key: NEXT_DB
```

Objets de base

Limiter les ressources

Limitation par container

En production, il est nécessaire de contrôler la consommation de



Objets de base

Limiter les ressources

```
apiVersion: extensions/v1beta1
kind: Deployment
...
  Containers:
  - name:
    resources:
      limits:
        cpu: 100m
        memory: 500Mi
      requests:
        cpu: 100m
        memory: 500Mi
```

* 200m signifie 200 milliCPU, soit 0.2 CPU

Objets de base

Limiter les ressources

Limitation par container et QosClass

- ➔ Si les valeurs Request et Limites (cpu et memory) sont égales, alors, le pod sera **Garanteed**
- ➔ Si un pod à une demande mémoire (request et limits)

Objets de base

DaemonSet

Un DaemonSet garantit que tous (ou certains) nœuds exécutent une copie d'un Pod. Au fur et à mesure que des nœuds sont ajoutés au cluster, des Pods y sont ajoutés. Au fur et à mesure que les nœuds sont retirés du cluster, les pods sont retirés.

Supprimer un DaemonSet nettoiera les Pods qu'il a créés.

Objets de base

DaemonSet

Quelques utilisations typiques d'un DaemonSet sont :

- ➔ exécuter un démon de stockage en cluster, tel que glusterd, ceph, sur chaque noeud.
- ➔ l'exécution d'un démon de collecte de logs sur chaque noeud, tel que fluentd ou logstash.

Objets de base

DaemonSet

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  labels:
    app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: fluentd-elasticsearch
```

Objets de base

DaemonSet

Toleration Key	Effect	Version	Description
node.kubernetes.io/not-ready	NoExecute	1.13+	
node.kubernetes.io/unreachable	NoExecute	1.13+	

Objets de base

Service **Stateful** vs. Service **Stateless**

- Lors de la réalisation d'un service pour une architecture, il arrive de devoir introduire, dans certaines situations, les notions de contexte ou de session. Nous allons voir quels sont les impacts de tels mécanismes et qu'il suffit d'appliquer la célèbre méthode KISS (Keep It Simple, Stupid) pour s'en sortir.

Que signifient les termes *contexte* et *session* ? :

- le contexte d'un service est l'environnement fonctionnel (par exemple un cas d'utilisation) ou technique (par exemple une application web J2EE) dans lequel celui-ci est invoqué.
- la session permet de conserver des informations au travers de plusieurs invocations d'un même service jusqu'à la libération de cette session.

Le contexte et la session nuisent à la compréhension et à la maintenabilité

Objets de base

Où gérer le contexte ou la notion de session ?

- Qui mieux que l'appelant sait dans quel contexte il se trouve ? L'appelé ne devrait jamais être influencé par le contexte ou les données de la session.
- Introduire des services contextuels (ou **stateful**) dans une architecture ne doit pas se faire à la légère et peut grièvement impacter la réutilisabilité, la maintenance et les performances des services.

En résumé, il faut donc :

Travaux pratiques

Utilisation de deployment.

Création de

- pod
- Deployment
- Service
- HostPath
- Persistent volume
- Configmap
- secrets

Déploiement d'un cluster Kubernetes

Déploiement : d'un master-nodeadm, d'un master-node, d'un worker-node.

Déploiement d'un cluster Kubernetes

Préparation des nœuds.

- Chaque machine doit tourner sur un OS supporté ex Debian 9 (stretch).
- Installer Docker
- Vérifier la visibilité réseau

Déploiement d'un cluster Kubernetes

Déploiement : d'un master-node

- Installer les outils kubelet kubeadm kubectl
- Lancer kubeadm init

Déploiement d'un cluster Kubernetes

Déploiement : d'un worker-node.

- Installer les outils kubelet kubeadm
- Lancer kubeadm join

Déploiement d'un cluster Kubernetes

Mise en place du Dashboard et du réseau.

- Installer une couche réseau Weave / flannel, …
- Installer le Dashboard

Déploiement d'un cluster Kubernetes

Travaux pratiques

- Mise en place cluster à 3 noeuds

Exploiter Kubernetes

Namespace

Pour gérer efficacement Kubernetes, il est conseillé de créer des namespaces par usage, puis de donner des accès et de contraintes d'utilisation par namespace.

Ex :

```
apiVersion: v1
```

Exploiter Kubernetes

Limiter les ressources – Quota

Un quota de ressources, défini par un objet ResourceQuota, fournit des contraintes qui limitent la consommation globale de ressources par **namespace**. Il peut limiter la quantité d'objets qui peuvent être créés dans un espace de noms par type, ainsi que la quantité totale de ressources de calcul qui peuvent être consommées par les ressources dans ce projet.

Exploiter Kubernetes

Limiter les ressources – Quota

Utilisation :

- ➔ Des équipes différentes travaillent dans des espaces de noms différents.
- ➔ L'administrateur crée un quota de ressources pour chaque espace

Exploiter Kubernetes

Limiter les ressources – Quota

Utilisation :

- Si la création ou la mise à jour d'une ressource viole une contrainte de quota, la demande échoue avec le code d'état HTTP 403 FORBIDDEN avec un message expliquant la contrainte qui aurait été violée.
- Si le quota est activé dans un espace de noms pour calculer les ressources telles que le processeur et la mémoire, les utilisateurs doivent spécifier des requêtes ou des limites pour ces valeurs ; sinon, le système de quota peut rejeter la création de pods. Astuce : Utilisez le contrôleur d'admission LimitRanger pour forcer les valeurs par défaut pour les pods qui ne nécessitent aucun calcul de ressources. Voir la marche à suivre pour un exemple de la façon d'éviter ce problème.

Exploiter Kubernetes

Limiter les ressources – Quota

Voici des exemples de stratégies qui pourraient être créées à l'aide d'espaces de noms et de quotas :

- Dans un cluster d'une capacité de 32 GiB RAM, et 16 cœurs,
  - l'équipe A utilise 20 GiB et 10 cœurs,
  - B utilise 10GiB et 4 cœurs,

Exploiter Kubernetes

Limiter les ressources – Quota

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-demo
  namespace : dev
spec:
  hard:
    requests.cpu: "1"
    limits.cpu: "2"
    requests.memory: 1Gi
    limits.memory: 2Gi
```

Exploiter Kubernetes

nodeSelector

Première étape : Fixer un label au nœud

Exécuter `$kubectl get nodes` pour obtenir la liste des noeuds de votre cluster.

Choisir celui auquel vous voulez ajouter une étiquette, lancer

```
$ kubectl label nodes <node-name> <label-key>=<label-value>
```

Exploiter Kubernetes

Seconde étape : Ajouter un sélecteur à votre définition

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    disktype: ssd
```

Lancer `$ run kubectl create -f pod-nginx.yaml`, le pod sera affecté au nœud avec le label.
Pour vérifier :

```
$ kubectl get pods -o wide
```

Exploiter Kubernetes

Affinité et anti-affinité

nodeSelector fournit un moyen très simple de contraindre les pods à des nœuds avec des étiquettes particulières.

La fonction affinité/anti-affinité, **actuellement en version bêta**, étend considérablement les types de contraintes que vous pouvez exprimer. Les principales améliorations sont les suivantes :

Exploiter Kubernetes

Health check

Afin de vérifier si un conteneur dans un pod est sain et prêt à servir le trafic, Kubernetes met en place une série de mécanismes de contrôles. Les HealthCheck, ou probes comme on les appelle dans Kubernetes, sont effectués par le kubelet pour déterminer quand redémarrer un conteneur (pour LivenessProbe) et par les services pour déterminer si un pod doit recevoir du trafic ou non (pour readinessProbe).

NB :

Il est de la responsabilité du développeur de l'application d'exposer une URL que le kubelet peut utiliser pour déterminer si le conteneur est sain (et potentiellement prêt).

Exploiter Kubernetes

Créer un pod qui expose un endpoint /health, en répondant avec un code d'état HTTP 200 :

```
livenessProbe:
  initialDelaySeconds: 2
  periodSeconds: 5
  httpGet:
    path: /health
```

Exploiter Kubernetes

Pour visualiser l'état de santé du pod :

```
$ kubectl describe pod hc
Name:                   hc
Namespace:              default
Security Policy:        anyuid
Node:                   192.168.99.100/192.168.99.100
Start Time:             Tue, 25 Apr 2017 16:21:11 +0100
Labels:                 <none>
Status:                 Running
...
Events:
  FirstSeen     LastSeen        Count   From
...
```

En plus d'un **livenessProbe** vous pouvez également spécifier un **readinessProbe**, qui peut être configuré de la même manière mais qui a un cas d'utilisation et une sémantique différents : il est utilisé pour vérifier la phase de démarrage d'un conteneur dans le pod. Imaginez un conteneur qui charge des données à partir d'un stockage externe tel que S3 ou une base de données qui a besoin d'initialiser certaines tables. Dans ce cas, vous voulez signaler quand le conteneur est prêt à servir le trafic.

Travaux pratiques

Déploiement d'une application

- Mise en place d'affinité sur un nœud
- Contrainte sur une machine
- Mise en œuvre de healthcheck

Création de conteneur personnalisé

Qu’est-ce qu’un Dockerfile ?

Les ‘Dockerfile’ sont des fichiers qui permettent de construire une image Docker adaptée à nos besoins, étape par étape.

Le fichier doit s’appeler Dockerfile et être à la racine de votre projet.

La première chose à faire dans un Dockerfile est de définir de quelle image vous héritez.

```
FROM alpine:3.9
```

FROM permet de définir notre image de base.

Création de conteneur personnalisé

Qu’est-ce qu’un Dockerfile ?

Exemple :

```
FROM python:3.5
RUN apt-get update && apt-get install -y libffi-dev
RUN mkdir -p /usr/src/app
RUN groupadd myappgroup \
    && useradd --create-home --home-dir /home/myappuser \
    --uid 4242 -g myappgroup myappuser
```

Création de conteneur personnalisé

FROM: importe récursivement la définition d'images parentes

LABEL: définit une valeur (author)

RUN: lance une commande via /bin/sh

CMD: définit la commande par défaut du conteneur

EXPOSE: définit les ports (UDP/TCP) écoutés au sein du conteneur

ENV: définit une variable d'environnement

COPY: copie des fichiers/dossiers dans l'image

Création de conteneur personnalisé

ENTRYPOINT: définit une commande exécutée au lancement du conteneur

VOLUME: espace de données persistent partagé entre l'hôte et le conteneur

USER: utilisateur au sein du process

Création de conteneur personnalisé

Attention à ne pas confondre RUN et CMD :

RUN exécute des commande et créer des commit et nouvelles images

CMD n'exécute rien au moment du build (ça définit dans l'image la commande à exécuter par un container)

Création de conteneur personnalisé

ENTRYPOINT vs. CMD

Un Dockerfile doit spécifier au moins un des deux :

➔ **ENTRYPOINT** doit être utilisé quand on utilise un container comme un exécutable

Création de conteneur personnalisé

ONBUILD

Ajouter à l'image un déclencheur (trigger) qui sera déclenché plus tard : lorsque cette image sera utilisée comme base image (FROM) pour la construction (build) d'une nouvelle image.

Cas d'utilisation :

Une image réutilisable.

Création de conteneur personnalisé

ONBUILD

Lors de la première construction, on ne peut pas ajouter des sources avec ADD car à ce moment nous n'avons pas encore d'application.

```
...
ONBUILD ADD . /app/src
```

Création de conteneur personnalisé

ONBUILD

Quand l'image de base est utilisée (FROM) pour un nouveau build.

Le builder vérifie s'il y a des trigger enregistrés et si oui, ils sont exécutés.

Les trigger sont nettoyés de l'image résultante : donc les trigger ne sont pas hérités par les "arrières petits-enfants".

Création de conteneur personnalisé

Bonnes pratiques :

- ➔ Un conteneur doit être éphémère
- ➔ Ne pas installer des paquets inutiles
- ➔ Un seul but / application par conteneur
- ➔ Limiter le nombre de couches

Création de conteneur personnalisé

Dockerfile : spécificités Windows

WORKDIR C:\\App

VOLUME C:\\Data

Possibilité d'utiliser ‘ à la place de #

ADD web.config C:/App

SHELL [“PowerShell”]

Par défaut : Cmd.exe

NB:

Pas de **FROM SCRATCH**

Création de conteneur personnalisé

Construire une image depuis un Dockerfile:

```
docker build -t <target> <rep>
```

Exemple

Création de conteneur personnalisé

L’image créée, nous pouvons la lancer via un docker run :

```
docker run -d easylinux/apache:2.5
```

Création de conteneur personnalisé

Les bonnes pratiques imposent de séparer les services dans des conteneurs séparés. Mais dans certains cas (Nginx/Php), il peut être lourd d’avoir 2 conteneurs.

Solution : **supervisor**

Création de conteneur personnalisé

Supervisor lit un fichier de configuration situé dans /etc/supervisor.

Chaque fichier finissant en .conf et situé dans le chemin /etc/supervisor/conf.d représente un processus qui est surveillé par Supervisor.

```
[program:<nom>]
command = <programme>
user = <utilisateur>
directory = <chemin>
autostart = true
autorestart = true
```

Travaux pratiques

Création et automatisation d'images personnalisées.

- Mise en œuvre d’un registre privé
- Créer un conteneur Mariadb

Kubernetes en production

Frontal Ingress.

- Kubernetes Ingress est un ensemble de règles de routage qui régissent la manière dont les utilisateurs externes accèdent aux services fonctionnant dans un cluster Kubernetes.
- Dans les déploiements réels de Kubernetes il y a souvent d'autres considérations que le routage pour la gestion d’Ingress.

Kubernetes en production

Ingress dans Kubernetes

Dans Kubernetes, il existe trois approches générales pour exposer votre application.

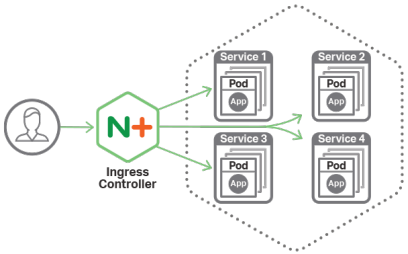
- Utiliser un service Kubernetes de type NodePort, qui expose l'application sur un port à travers chacun de vos nœuds.

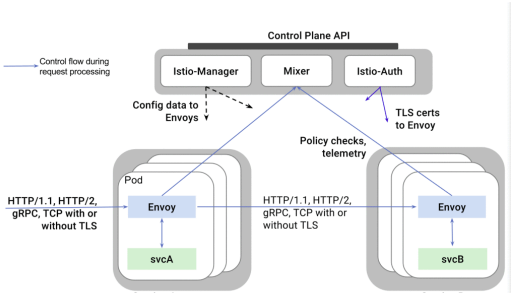
Kubernetes en production

Ingress Contrôleurs et Ingress ressources

- Une entrée est un concept de base (en bêta) de Kubernetes, mais elle est toujours implémentée par un **proxy tiers**.
- Le contrôleur Ingress est responsable de lire l'information sur les ressources d'entrée et de traiter ces données en conséquence. Différents contrôleurs Ingress ont étendu la spécification de différentes manières pour prendre en charge des cas d'utilisation supplémentaires.
- Ingress est étroitement intégré à Kubernetes, ce qui signifie que vos workflows existants autour de kubectl s'étendront probablement bien à la gestion des entrées.
- Un contrôleur Ingress n'élimine généralement pas le besoin d'un équilibreur de charge externe - le contrôleur Ingress ajoute simplement une couche supplémentaire de routage et de contrôle derrière l'équilibreur de charge.

Kubernetes en exploitation





Kubernetes en production

Gestion des ressources et autoscaling.

- L'Autoscaler Horizontal Pod Autoscaler met automatiquement à l'échelle le nombre de pods dans un contrôleur de réplication, un déploiement ou un ensemble de répliques en fonction de l'utilisation observée du CPU (ou, avec le support de métriques personnalisées, sur certaines autres métriques fournies par l'application). Notez que la mise à l'échelle horizontale ne s'applique pas aux objets qui ne peuvent pas être mis à l'échelle, par exemple, les DaemonSets.
- L'Autoscaler Horizontal Pod est implémenté en tant que ressource API Kubernetes et contrôleur. La ressource détermine le comportement du régulateur. Le contrôleur ajuste périodiquement le nombre de répliques d'un contrôleur de réplication ou d'un déploiement afin de faire correspondre l'utilisation moyenne observée du CPU à la cible spécifiée par l'utilisateur.

```
$ kubectl autoscale rs monApp --min=2 --max=5 --cpu-percent=80
```

Kubernetes en production

Service Discovery (env, DNS).

Le service Discovery est un processus qui consiste à déterminer comment se connecter à un service. Bien qu'il existe une option de découverte de service basée sur les variables d'environnement disponibles, la découverte de service basée sur DNS est préférable. Notez que le DNS est un add-on cluster, donc assurez-vous que

Travaux pratiques

- Mise en œuvre du frontal Ingress
- Autoscaling du frontal Web

Kubernetes en production

Gestion des accès.

Il y a trois éléments en jeu :

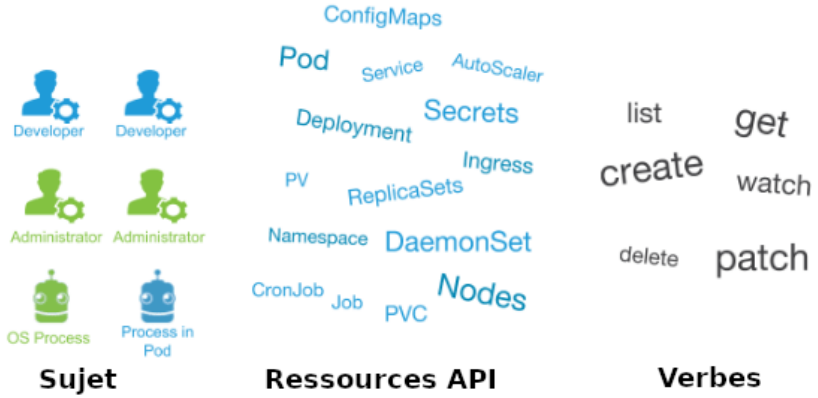
- **Sujets** : L'ensemble des utilisateurs et des processus qui souhaitent accéder à l'API Kubernetes.
- **Ressources** : L'ensemble des objets API Kubernetes disponibles dans le cluster. Exemples : Pods, Déploiements, Services, Nœuds et

Kubernetes en production

Gestion des accès.

Les trois éléments en jeu :

- **Sujets**
- **Ressources**
- **Verbes**



Kubernetes en production

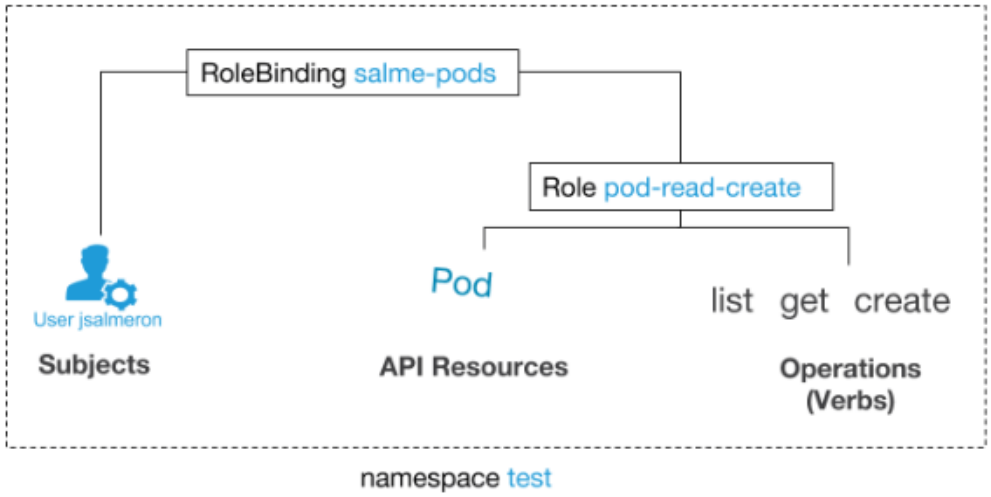
Gestion des accès.

Pour connecter ces trois types d'entités, on utilise les différents objets API RBAC disponibles dans Kubernetes.

- **Rôles** : Permet de connecter les ressources API et les verbes. Ceux-ci peuvent être réutilisés pour différents sujets.
  Ils sont liés à un espace de nommage (nous ne pouvons pas utiliser de caractères génériques pour en représenter plus d'un, mais nous pouvons déployer le même

Kubernetes en production

Gestion des accès.



Kubernetes en production

Gestion des accès.

Supposons que nous voulons un utilisateur jdoe qui pourra lancer :

```
kubectl get pods --namespace test
kubectl describe pod --namespace test pod-name
```

Kubernetes en production

Gestion des accès.

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: role
metadata:
  name: pod-read-create
  namespace: test
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get","list","create"]
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: jdoe-pods
  namespace: test
subjects:
- kind: user
  name: jdoe
  apiGroup: rbac.authorization.k8s.io/v1beta1
roleRef:
  kind: Role
  name: ns-admin
  apiGroup: rbac.authorization.k8s.io/v1beta1
```

Kubernetes en production

Gestion des accès

Question : maintenant que l'utilisateur peut créer des pods, pouvons-nous en limiter le nombre ?

- Pour ce faire, d'autres objets, non directement liés à la spécification RBAC, permettent de configurer la quantité de

Kubernetes en production

Gestion des accès

Quelle est la différence entre les utilisateurs réguliers et les ServiceAccounts.

- **Utilisateurs** : Ceux-ci sont globaux et s'adressent aux êtres humains ou aux processus vivant en dehors du cluster.
- **ServiceAccounts** : destinés aux processus intra-cluster qui se déroulent à l'intérieur des pods.

Tous deux ont en commun de devoir s'authentifier par rapport à l'API pour effectuer un ensemble d'opérations sur un ensemble de ressources.

Leurs domaines semblent être clairement définis. Ils peuvent aussi appartenir à ce qu'on appelle des groupes

Kubernetes en production

Gestion des accès

un RoleBinding peut lier plus d'un sujet

ServiceAccounts ne peut appartenir qu'au groupe "system:serviceaccounts".

NB : les utilisateurs n'ont pas d'objet API Kubernetes associé.

Kubernetes en production

Gestion des accès

Conséquence directe : si le cluster ne stocke aucune information sur les utilisateurs, l'administrateur devra gérer les identités en dehors du cluster. Il y a différentes façons de le faire :

- Certificats TLS,
- jetons et
- OAuth2, ...

De plus, il faudra créer des contextes kubectl pour pouvoir accéder au cluster avec ces nouvelles informations d'identification.

Pour créer les fichiers d'identification, il est possible d’ utiliser les commandes de configuration de kubectl (qui ne nécessitent aucun accès à l'API de Kubernetes, donc elles peuvent être exécutées par n'importe quel utilisateur).

Kubernetes en production

Gestion des accès

RBAC et les déploiements

On peut maintenant définir ce qu'un utilisateur donné peut faire à l'intérieur du cluster. Cependant, qu'en est-il des déploiements qui nécessitent un accès à l'API Kubernetes ?

Prenons l’exemple d’une application d'infrastructure : RabbitMQ.

Kubernetes en production

Gestion des accès

En matière de ServiceAccounts, la bonne pratique consiste :

- Créer un ServiceAccounts par déploiement avec le minimum de privilèges pour travailler.

Dans le cas d'applications qui nécessitent un accès à l'API Kubernetes,on pourrait être tenté d'avoir un type de "ServiceAccount privilégié" qui pourrait faire presque tout dans le cluster. Mais des opérations non désirées pourraient se produire.

En outre, les différents déploiements auront des besoins différents en termes d'accès aux API, il est donc logique d'avoir différents ServiceAccounts pour chaque déploiement.

Quelle doit être la configuration RBAC appropriée pour notre déploiement RabbitMQ.

D'après la documentation, RabbitMQ interroge l'API Kubernetes pour la liste des Endpoints.

Kubernetes en production

Gestion des accès – le service

```
{{- if .Values.rbacEnabled }}
apiVersion: v1
kind: ServiceAccount
metadata:
  name: {{ template "rabbitmq.fullname" . }}
```

Kubernetes en production

Gestion des accès – le role

```
{{- if .Values.rbacEnabled }}
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: {{ template "rabbitmq.fullname" . }}-endpoint-reader
  labels:
    app: {{ template "rabbitmq.name" . }}
    chart: {{ template "rabbitmq.chart" .  }}
    release: "{{ .Release.Name }}"
    heritage: "{{ .Release.Service }}"
rules:
- apiGroups: [""]
  resources: ["endpoints"]
  verbs: ["get"]
{{- end }}
```

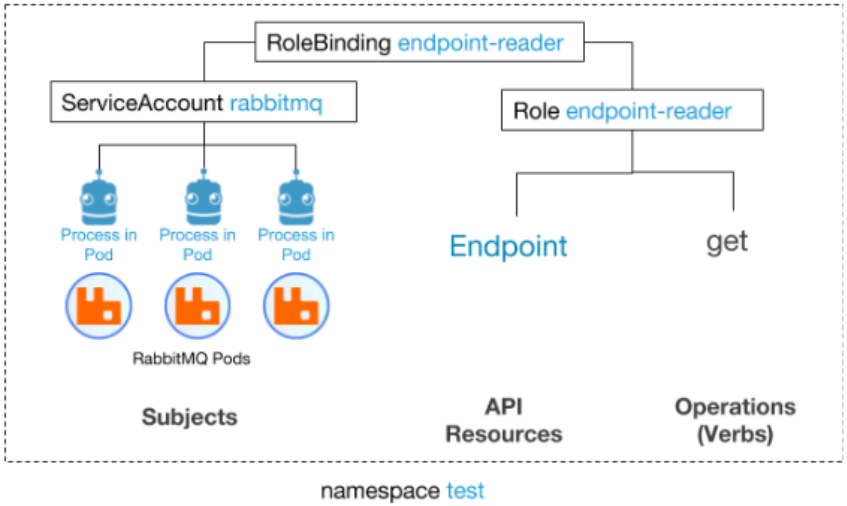
Kubernetes en production

Gestion des accès – le roleBinding

```
{{- if .Values.rbacEnabled }}
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: {{ template "rabbitmq.fullname" . }}-endpoint-reader
  labels:
```

Kubernetes en production

Gestion des accès



Travaux pratiques

- Création d'un namespace
- Ajout d'un utilisateur à ce namespace
- Gérer les droits d'accès d'une application

Kubernetes en production

Haute disponibilité.

Il y a deux approches différentes pour mettre en place un cluster Kubernetes hautement disponible avec kubeadm :

- Avec des nœuds master. Cette approche nécessite moins d'infrastructure. Les etcd et les masters sont situés au même niveau.
- Avec un cluster etcd. Cette approche nécessite davantage d'infrastructure. Les nœuds masters et les etcd sont séparés.

Avant de choisir, vous devriez examiner attentivement quelle approche répond le mieux aux besoins de vos applications et de votre environnement.

Vos clusters doivent exécuter Kubernetes version 1.12 ou ultérieure.

Sachez également que la mise en place de clusters HA avec kubeadm est encore expérimentale et sera encore simplifiée dans les versions futures. Vous pourriez rencontrer des problèmes lors de la mise à niveau de vos clusters, par exemple.

Kubernetes en production

Haute disponibilité.

Pour les deux méthodes, vous avez besoin de :

- ✔ Trois machines qui répondent aux exigences minimales de kubeadm pour les maîtres.
- ✔ Trois machines qui répondent aux exigences minimales de kubeadm pour les minions.
- ✔ Connectivité réseau complète entre toutes les machines du cluster (réseau public ou privé)
- ✔ privilèges sudo sur toutes les machines

Kubernetes en production

Mode maintenance.

Si vous avez besoin de redémarrer un nœud (comme pour une mise à niveau du noyau, une mise à niveau de libc, une réparation matérielle, etc.), et que le temps d'arrêt est bref, quand le Kubelet redémarre, il va tenter de redémarrer automatiquement les modules.

Si le redémarrage prend plus de temps (le temps par défaut est de 5 minutes, contrôlé par --pod-eviction-timeout sur le contrôleur-manager), alors le node controler les pods liés au noeud non disponible.

S'il existe un jeu de répliques correspondant (ou un contrôleur de réplication), alors une nouvelle copie du pod sera lancée sur un autre nœud. Ainsi, dans le cas où tous les pods sont répliqués, les mises à niveau peuvent se faire sans action spéciale, en supposant que tous les nœuds ne s'arrêtent pas en même temps.

Si vous voulez plus de contrôle sur le processus de mise à niveau, vous pouvez utiliser le workflow suivant :

Utiliser cette commande kubectl pour rendre le nœud inactif :

```
$ kubectl drain $NODENAME
```

Kubernetes en production

Mode maintenance.

Ceci empêche les nouveaux pod d'être démarré sur ce nœud pendant l'arrêt.

Pour les pods avec un replicaSet, le pod sera remplacé par un nouveau pod qui sera programmé sur un nouveau nœud. De plus, si le pod fait partie d'un service, les clients seront automatiquement redirigés vers le nouveau pod.

Travaux pratiques

Remplacement d’un node

Questions ?