
Master Cryptis 2019

Université de Limoges

ASSR

CAMARA Kerfala

ARCHITECTURES VIRTUALISÉES OU SOFTWARE-DEFINED NETWORKING "SDN"

Kubernetes



CookBook

18 novembre 2018

Table des matières

| | | |
|----------|--|-----------|
| 1 | Résumé | 3 |
| 2 | Présentation générale | 3 |
| 3 | Définition | 4 |
| 4 | Ingrédients Logiciels / Matériels | 5 |
| 5 | Recettes Illustratives | 6 |
| 5.1 | Installation | 6 |
| 5.2 | Déploiement | 7 |
| 5.3 | test minimal | 9 |
| 6 | Bibliographie / Documentation | 13 |

1 Résumé

L'architecture virtualisée ou SDN (software Defined Network) est un concept qui consiste à faire une abstraction de la couche réseau ainsi nous avons une vue unifiée d'une application dans le réseau. Elle a pour but de rendre programmable les réseaux par le biais d'un contrôleur (logiciel) centralisé.

SDN ou réseau à définition logicielle en français permet de moins se soucier de la contrainte réseau dans le cadre d'une future évolution d'une application car le réseau est directement géré par le programmeur, il a juste besoin de savoir si une architecture peut supporter ses exigences en terme de ressources sans se soucier de la façon dont l'application sera déployée.

la finalité d'une SDN est de fournir aux développeurs une API qui permettra le déploiement d'une application de façon simple et surtout permettre grâce à l'API de décider s'il faut augmenter le trafic (par moyen de répliques) ou faire l'opération contraire, décider de l'ordonnancement du flux.

2 Présentation générale

Kubernetes est un système open source permettant de gérer des applications conteneurisées sur plusieurs hôtes. Il fournit des mécanismes de base pour le déploiement, la maintenance et la mise à l'échelle des applications (disponibilité selon besoin utilisateur).

l'architecture principale de kubernetes est basé sur " kubernetes cluster " qui comprend principalement deux parties un Master qui coordonne l'ensemble des opérations du cluster et des noeuds lieu où les applications conteneurisées comme docker sont exécutées.

Master est élément qui permet le contrôle de l'ensemble des composants liés au fonctionnement d'une application sur kubernetes, il dispose d'une api server qui gère les noeuds, d'un etcd est une unité de stockage persistante qui contient l'état du cluster directement lié à l'api server pour permettre à des noeuds d'effectuer des changements d'états au besoin. D'un controller manager qui gère tout ce qui est affectation de ressources, accès et le contrôle de noeud.

Le noeud contient principalement un ou plusieurs pods qui contiennent en leur sein un ou plusieurs applications conteneurisées pouvant se partager des volumes.

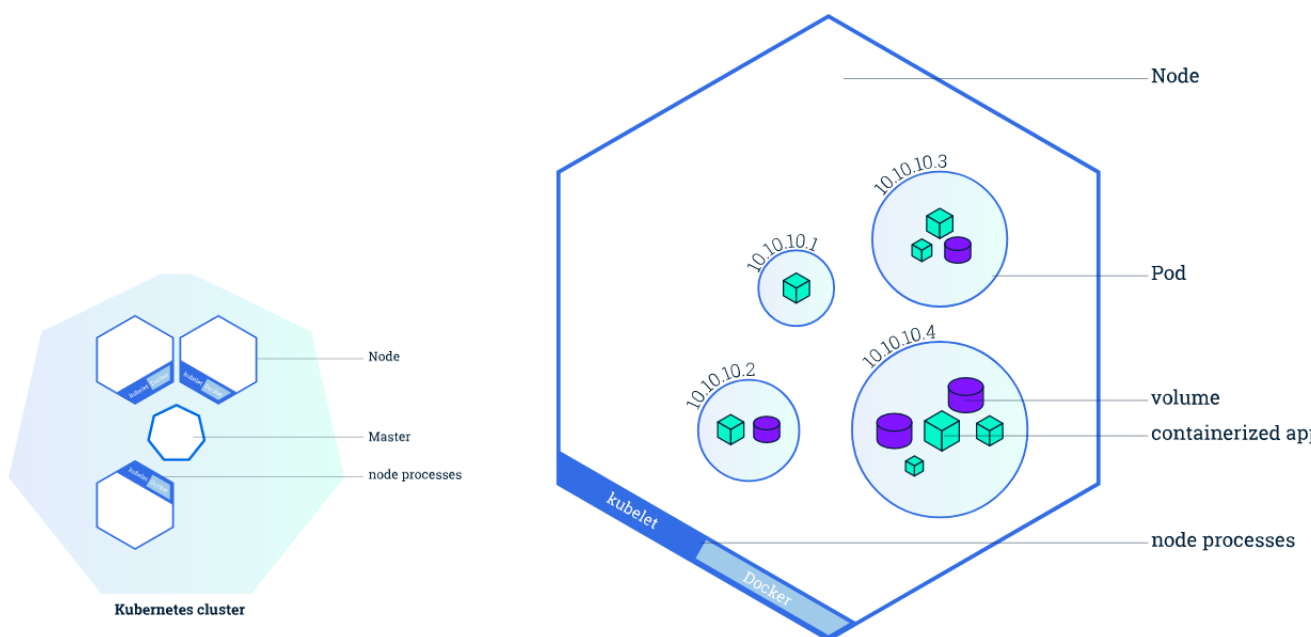
Après création du cluster de kubernetes, il fonctionne essentiellement sur quatre grandes parties : le déploiement d'une application ,l'exposition d'une application,le redimensionnement d'une application et enfin les mises à jour d'une application.

- le déploiement d'une application se fait dans le noeud qui contient l'application conteneurisée, il est à noter que kubernetes à sa propre logique des conteneurs qui est le pod (abstraction logique d'un ou plusieurs applications), donc dans un noeud, kubernetes crée un pod dans lequel il place l'application concernée.
- l'exposition d'une application sur kubernetes est l'opération qui consiste à rendre une application disponible, elle se réalise par le biais d'un service. En résumé c'est toute la logique réseau liée l'accès de la ressource.
- le redimensionnement d'une application est l'ensemble des opérations nécessaires pour le bon fonctionnement de la mise en échelle qui passe forcément par des opérations de répliquions pour affecter plus de ressources ou faire l'opération inverse (désaffecter la ressource). il est à noter que lorsque nous disposons de plusieurs instances d'une application, kubernetes fait du load balancer (répartition de charges) entre les ressources.
- Vu les contraintes développeurs qui font les mise à jours d'une application après déploiement, kubernetes propose un principe de création d'un pod pour une nouvelle mise à jour qui est aussi accessible et par la suite confirmé la mise à jour par un rollout qui supprime l'ancienne version. Il est à noter qu'on peut revenir sur une version précédente d'une application avec rollout undo.

3 Définition

- **Cluster** représente toute l'architecture kubernetes (un master et ses noeuds).
- **Master** l'ordonnanceur de kubernetes permet de réaliser une opération sur tout composant de l'architecture kubernetes.
- **Noeud** appelé aussi worker c'est l'endroit où les conteneurs sont déployés.
- **Pod est** l'unité logique avec kubernetes, c'est une abstraction des composants conteneurisés. Il permet à kubernetes d'ordonnancer, rendre accessible, ou toutes opérations pouvant interagir avec une application.
- **Service** est le fait de rendre une application accessible c'est à dire l'exposer. Le master est responsable de la mise en marche d'un service.
- **load balancer** permet la répartition des charges entre plusieurs instances d'une application.
- **Application conteneurisée** est une application qui fonctionne dans un environnement d'exécution léger, elle utilise l'ensemble des fichiers et bibliothèques

nécessaire à son fonctionnement qui permet de maximiser sa portabilité ainsi elle fonctionne sur son OS hôte contrairement au VM.



4 Ingrédients Logiciels / Matériels

Docker est un logiciel de conteneurisation des applications, kubernetes est fortement lié à docker car il est un logiciel de gestion et d'orchestration de conteneurs. Donc cette technologie permet la gestion simplifiée des applications conteneurisées tout en permettant des opérations comme la réplication d'application dans le cluster.

Kubernetes fonctionne avec le cloud ce qui permet de déployer une application, la répliquer au besoin. Cette partie fonctionne par des mécanismes de plug in ainsi tout nouveau fournisseur de cloud peut facilement intégrer kubernetes. A ce jour kubernetes prend en charges les cloud :

- AWS EC2
- Alibaba Cloud
- Azure
- CenturyLink Cloud
- Google Compute Engine
- Stackpoint.io * (interface web permettant la configuration de n'importe quel cloud).

Utilisation de machines virtuelles afin de pouvoir travailler de façon transparente si on veut l'utiliser sur sa machine personnelle pour ressortir la notion de cluster et noeud (tous joignables par leur adresse ip).

5 Recettes Illustratives

5.1 Installation

Configuration de base kubernetes proposé le site officiel :

- `apt-get update apt-get install -y apt-transport-https curl`
- `curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -`
- `cat «EOF >/etc/apt/sources.list.d/kubernetes.list deb https://apt.kubernetes.io/kubernetes-xenial main EOF`
- `apt-get update`
- `apt-get install -y kubelet kubeadm kubectl`
- `apt-mark hold kubelet kubeadm kubectl`

A ces commandes il faut ajouter les instructions qui ont pu rendre la technologie émuable sur ma machine.

- `swapoff -a`
- Modification du fichier `/etc/fstab`
- Configuration réseau , création d'interface et affectation d'une adresse statique pour les deux hôtes.
- Modification du fichier `/etc/hosts`
- Installation du docker pour des besoins de conteneurs.

En résumé pour mettre en place la plateforme il m'a fallu installer kubernetes sur deux machines donc il faut faire l'ensemble des instructions pour les deux hôtes, puisse qu'il s'agit de deux hôtes différentes (maitre et esclave).

Configuration interface

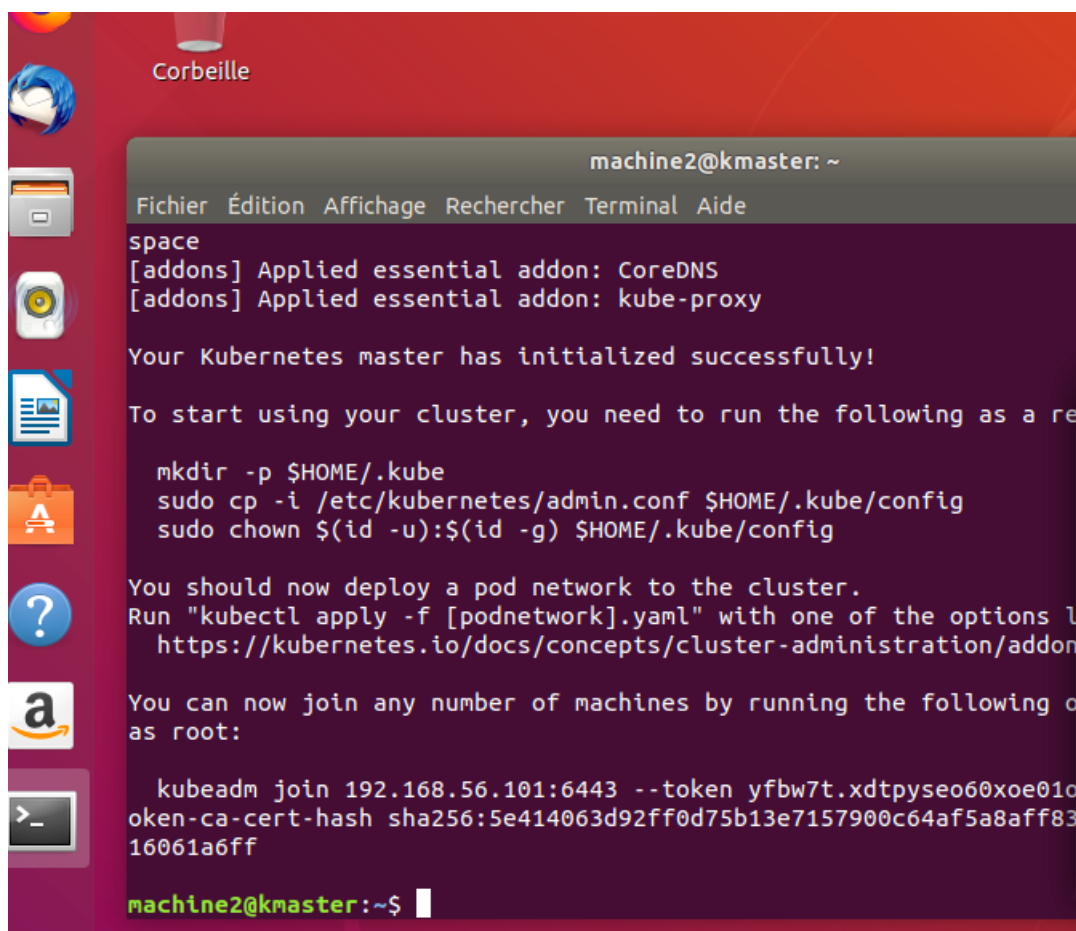
```
: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:38:be:e2 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.102/24 brd 192.168.56.255 scope global enp0s8
        valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe38:bee2/64 scope link
        valid_lft forever preferred_lft forever
: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:cf:9b:9c:aa brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
ode@knode:~$
```

5.2 Déploiement

Première phase

Consiste à déployer la technologie kubernetes, je l'ai réalisé sur deux machines virtuelles, l'une maître contenant le cluster (kmaster) et l'autre contenant les nœuds. Pour ce faire la première partie consiste à initialiser le cluster au niveau de la machine concernée puis récupérer le token d'initialisation qui permet à tous nœuds désireux de rejoindre ce cluster.

```
sudo kubeadm init --pod-network-cidr=192.168.0.0/16 --apiserver-advertise-address =192.168.56.101
```



```
machine2@kmaster: ~  
Fichier  Édition  Affichage  Rechercher  Terminal  Aide  
space  
[addons] Applied essential addon: CoreDNS  
[addons] Applied essential addon: kube-proxy  
  
Your Kubernetes master has initialized successfully!  
  
To start using your cluster, you need to run the following as a re  
  
mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config  
  
You should now deploy a pod network to the cluster.  
Run "kubectl apply -f [podnetwork].yaml" with one of the options l  
https://kubernetes.io/docs/concepts/cluster-administration/addon  
  
You can now join any number of machines by running the following o  
as root:  
  
kubeadm join 192.168.56.101:6443 --token yfbw7t.xdtpyseo60xoe01o  
oken-ca-cert-hash sha256:5e414063d92ff0d75b13e7157900c64af5a8aff83  
16061a6ff  
  
machine2@kmaster:~$
```

Deuxième phase

La deuxième partie, le noeud rejoint le cluster : le token du cluster est généré lors de l'initiation du cluster avec kubeadm init. cette consiste à kubeadm join associé à l'adresse ip de du cluster et son token pour rejoindre l'architecture.


```

pde@knode:~$ sudo kubeadm join 192.168.56.101:6443 --token yrbw7t.xdtpyse060xoe0io --discovery-token-ca-cert-hash sha256:5e414063d92ff0d75b13e7157900c64af5a8aff8369b2715eb428e216061a6ff
preflight] running pre-flight checks
[WARNING RequiredIPVSKernelModulesAvailable]: the IPVS proxier will not be used, because the following required kernel modules are not loaded: [ip_vs_rr ip_vs_wrr ip_vs_sh ip_vs] or no builtin kernel ipvs support: map[ip_vs_rr:{} ip_vs_wrr:{} ip_vs_sh:{} nf_conntrack_ipv4:{} ip_vs:{}]. You can solve this problem with following methods:
1. Run 'modprobe -- ' to load missing kernel modules;
2. Provide the missing builtin kernel ipvs support

[WARNING Service-Docker]: docker service is not enabled, please run 'systemctl enable docker.service'
[discovery] Trying to connect to API Server "192.168.56.101:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://192.168.56.101:6443"
[discovery] Requesting info from "https://192.168.56.101:6443" again to validate TLS against the pinned public key
[discovery] Cluster info signature and contents are valid and TLS certificate validates against pinned roots, will use API Server "192.168.56.101:6443"
[discovery] Successfully established connection with API Server "192.168.56.101:6443"
[kubelet] Downloading configuration for the kubelet from the "kubelet-config-1.12" ConfigMap in the kube-system namespace
[kubelet] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[preflight] Activating the kubelet service
[tlscert] Waiting for the kubelet to perform the TLS Bootstrap...
[patchnode] Uploading the CRI Socket information "/var/run/docker.sock" to the Node API object "knode" as an annotation

This node has joined the cluster:
Certificate signing request was sent to apiserver and a response was received.
The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the master to see this node join the cluster.

```

5.3 test minimal

Déploiement d'une application conteneurisée qui va servir d'exemple.

```

machine2@kmaster:~$ kubectl run nginx --image=nginx:1.10.0
kubectl run --generator=deployment/apps.v1beta1 is DEPRECATED and will be removed in a future version. Use kubectl create instead.
deployment.apps/nginx created
machine2@kmaster:~$

```

Affichage des pods

```

machine2@kmaster:~$ kubectl get pods
NAME                                READY    STATUS    RESTARTS   AGE
nginx-5fc69dfb5d-556sm              0/1      Pending   0           101s

```

Intance actuellement déployé sur le cluster

```

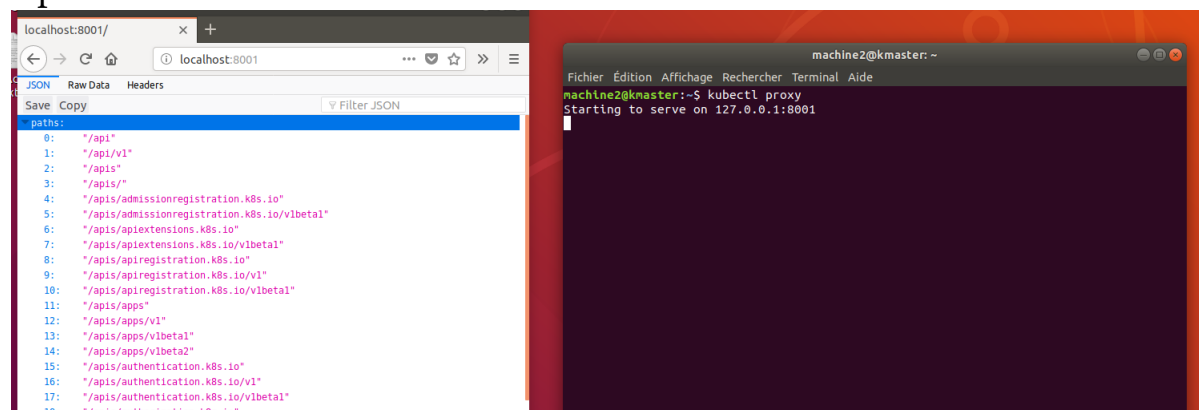
machine2@kmaster:~$ kubectl get deployments
NAME      DESIRED    CURRENT    UP-TO-DATE    AVAILABLE    AGE
nginx     1          1          1             0            4m6s

```

Rédimensionnement de l'application en utilisant les opérations de répliques dans ce cas précis : création de six instances. On remarque sur l'image en dessous une cinq nouvelles instances qui ont été crée à la même période (plus précisément il y'a : 77s).

```
machine2@kmaster:~$ kubectl scale deployments/nginx --replicas=6
deployment.extensions/nginx scaled
machine2@kmaster:~$ kubectl get deployments
NAME      DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
nginx     6        6        6           0          6m57s
machine2@kmaster:~$ kubectl get pods -o wide
NAME                                READY  STATUS   RESTARTS  AGE    IP             NODE              NOMINATED NODE
nginx-5fc69dfb5d-556sm              0/1    Pending  0         7m24s  <none>         <none>            <none>
nginx-5fc69dfb5d-6nqkf              0/1    Pending  0         77s    <none>         <none>            <none>
nginx-5fc69dfb5d-6tfc9              0/1    Pending  0         77s    <none>         <none>            <none>
nginx-5fc69dfb5d-892mb              0/1    Pending  0         77s    <none>         <none>            <none>
nginx-5fc69dfb5d-k76c5              0/1    Pending  0         77s    <none>         <none>            <none>
nginx-5fc69dfb5d-nfk2x              0/1    Pending  0         77s    <none>         <none>            <none>
```

Aperçu des ressources du kubectl via le localhost



Création de services en kubernetes est une opération permettant de rendre une application accessible. Dans la capture ci-dessous il y'a trois commandes dont une pour lister les services, une pour exposer l'application, et la dernière pour décrire le service qui à été déployé.

```

machine2@kmaster:~$ kubectl get services
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
kubernetes    ClusterIP     10.96.0.1     <none>         443/TCP    2d2h
machine2@kmaster:~$ kubectl expose deployment/nginx --type="NodePort" --port 8080
service/nginx exposed
machine2@kmaster:~$ kubectl get services
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
kubernetes    ClusterIP     10.96.0.1     <none>         443/TCP    2d2h
nginx         NodePort      10.107.149.132 <none>         8080:31595/TCP 9s
machine2@kmaster:~$ kubectl describe services/nginx
Name:         nginx
Namespace:    default
Labels:       run=nginx
Annotations:   <none>
Selector:     run=nginx
Type:         NodePort
IP:           10.107.149.132
Port:         <unset> 8080/TCP
TargetPort:   8080/TCP
NodePort:     <unset> 31595/TCP
Endpoints:    <none>
Session Affinity: None
External Traffic Policy: Cluster
Events:       <none>
machine2@kmaster:~$

```

Suppression de services

```

machine2@kmaster:~$ kubectl get services
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
kubernetes    ClusterIP     10.96.0.1     <none>         443/TCP    2d2h
nginx         NodePort      10.107.149.132 <none>         8080:31595/TCP 3m21s
machine2@kmaster:~$ kubectl delete service -l run=nginx
service "nginx" deleted
machine2@kmaster:~$ kubectl get services
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
kubernetes    ClusterIP     10.96.0.1     <none>         443/TCP    2d2h
machine2@kmaster:~$

```

Vue générale de l'ensemble des applicatins déployées sur le cluster avec les composants du kmaster.

```

machine2@kmaster:~$ kubectl get pods -o wide --all-namespaces
NAMESPACE      NAME                                     READY   STATUS             RESTARTS   AGE   IP              NODE      NOMINATED NODE
default        nginx-5fc69dfb5d-556sm                 0/1     Pending            0          25m   <none>          <none>    <none>
default        nginx-5fc69dfb5d-6nqkf                 0/1     Pending            0          19m   <none>          <none>    <none>
default        nginx-5fc69dfb5d-6tfc9                 0/1     Pending            0          19m   <none>          <none>    <none>
default        nginx-5fc69dfb5d-892mb                 0/1     Pending            0          19m   <none>          <none>    <none>
default        nginx-5fc69dfb5d-k76c5                 0/1     Pending            0          19m   <none>          <none>    <none>
default        nginx-5fc69dfb5d-nfk2x                 0/1     Pending            0          19m   <none>          <none>    <none>
kube-system    calico-kube-controllers-84c8654497-nffjn 0/1     CrashLoopBackOff   45       41h   192.168.56.102 knode     <none>
kube-system    calico-node-zhc79                      0/2     Pending            0          39h   <none>          <none>    <none>
kube-system    coredns-576cbf47c7-252jk               0/1     ContainerCreating  0        2d2h   <none>          knode     <none>
kube-system    coredns-576cbf47c7-zv4th               0/1     ContainerCreating  0        2d2h   <none>          knode     <none>
kube-system    etcd-kmaster                           1/1     Running            3        39h   192.168.56.101 kmaster    <none>
kube-system    kube-apiserver-kmaster                  1/1     Running            4        39h   192.168.56.101 kmaster    <none>
kube-system    kube-controller-manager-kmaster         1/1     Running            5        39h   192.168.56.101 kmaster    <none>
kube-system    kube-proxy-qks2d                        1/1     Running            1        39h   192.168.56.102 knode     <none>
kube-system    kube-proxy-shtn7                        1/1     Running            3        2d2h   192.168.56.101 kmaster    <none>
kube-system    kube-scheduler-kmaster                  1/1     Running            4        39h   192.168.56.101 kmaster    <none>
kube-system    kubernetes-dashboard-77fd78f978-bqc65   0/1     ContainerCreating  0        41h   <none>          knode     <none>
machine2@kmaster:~$

```

On remarque les applications qui sont affectées au kmaster et les applications liées au knode, et tous avec la colonne age qui est le temps depuis que le composant a été mis en marche.

6 Bibliographie / Documentation

<https://kubernetes.io/>
<https://kubernetes.io/docs/tutorials/>
<https://kubernetes.io/docs/setup/independent/install-kubeadm/>
<https://www.edureka.co/kubernetes-certification>
<https://fr.wikipedia.org/wiki/Kubernetes>
<https://www.lemagit.fr/definition/SDN-Software-Defined-Networking>
<https://www.youtube.com/watch?v=UWg3ORRRF60>