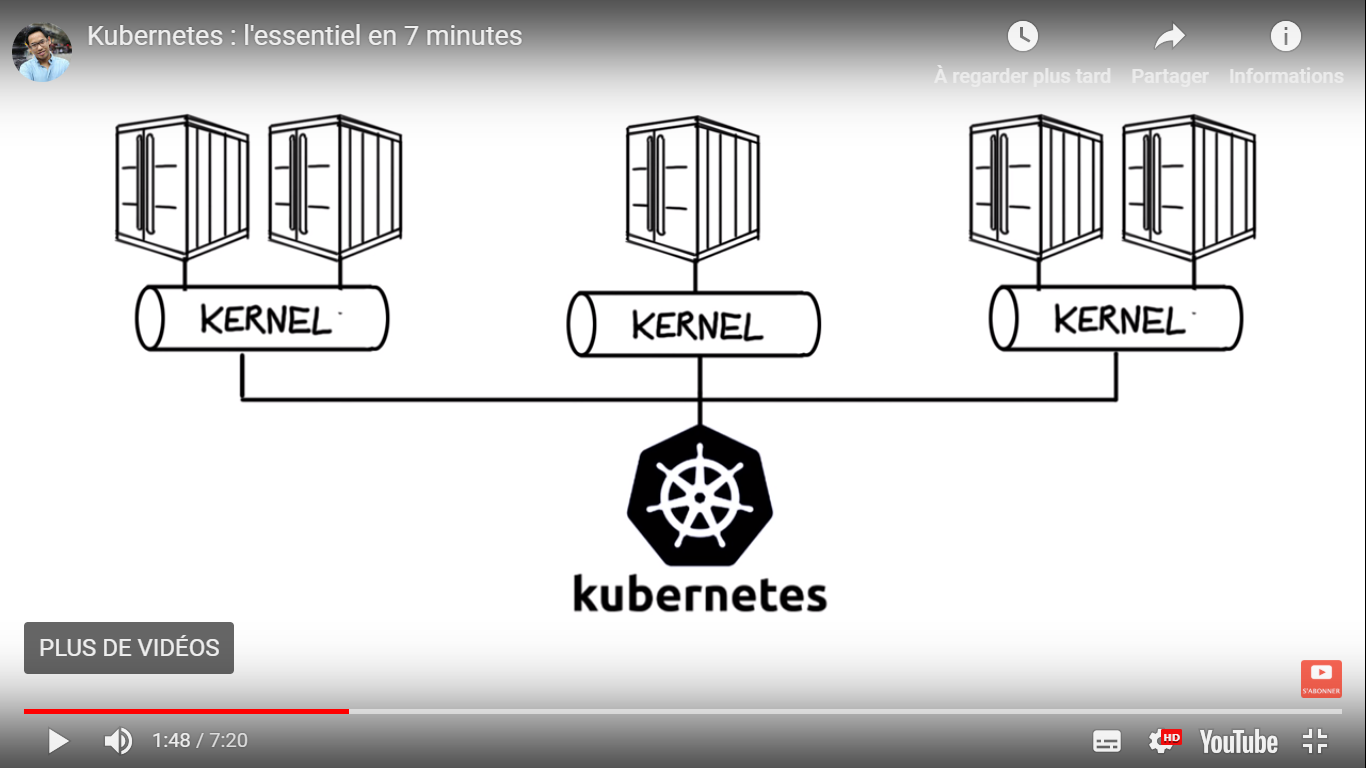
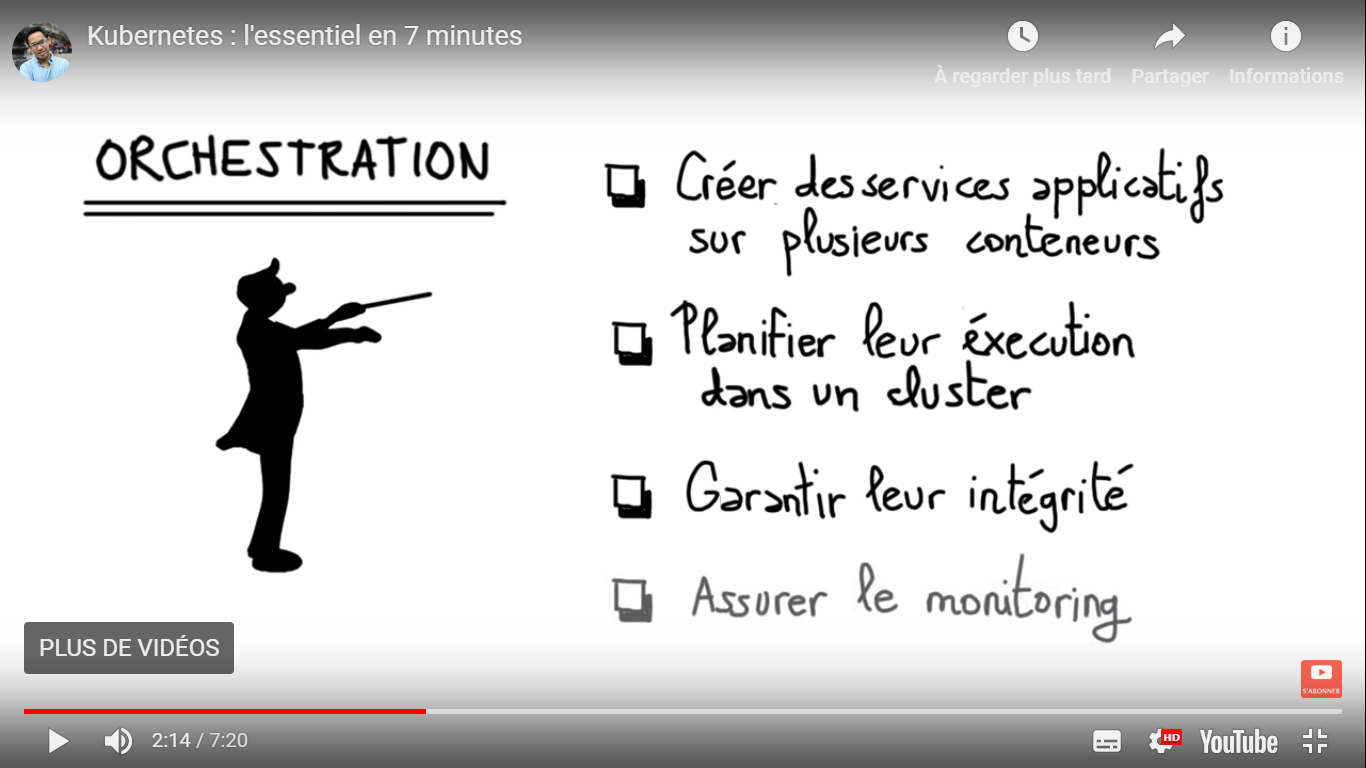
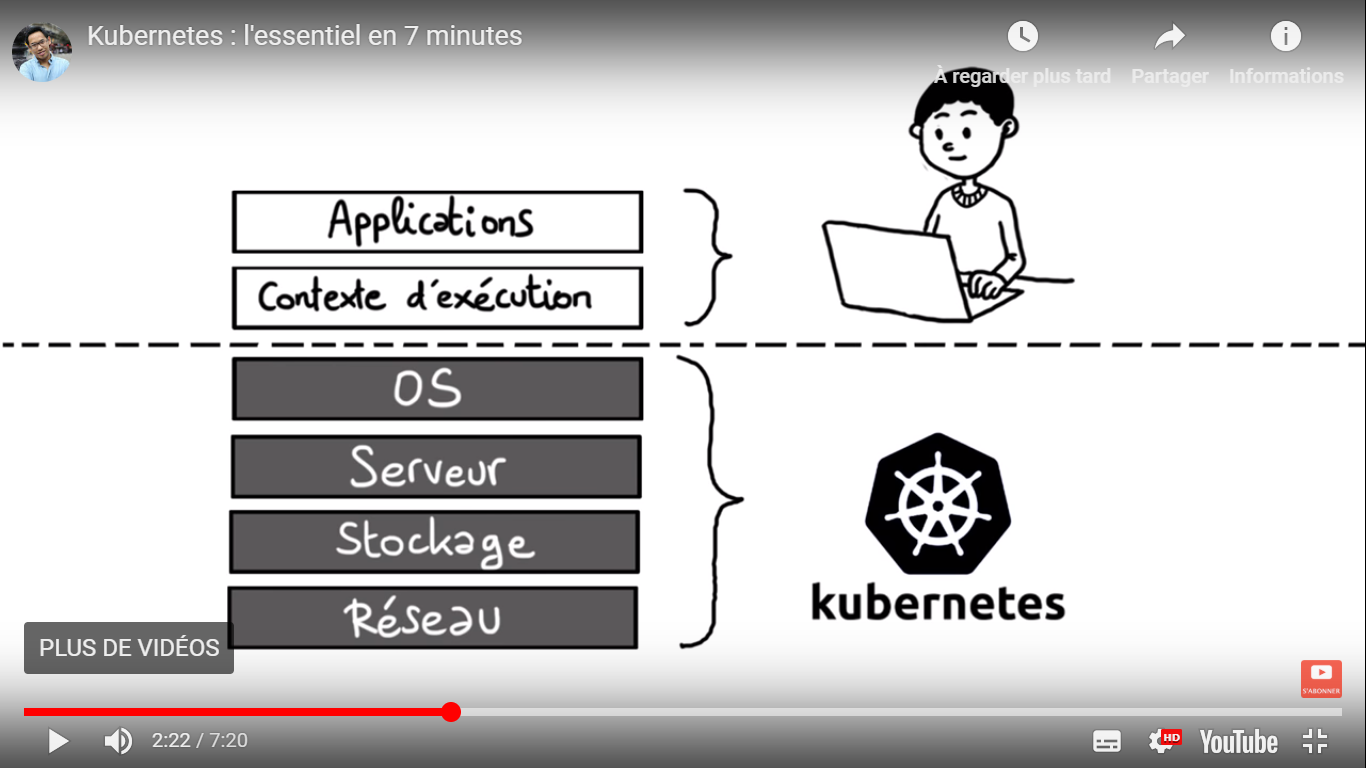
<http://cookieconnecte.fr/2018/01/26/kubernetes-pour-les-debutants/>

K8S permet d’interconnecter en cluster des applications tournant dans des containers





K8S gère la partie infrastructure, le développeur se concentre sur la partie applicative



**K8S Master** : orchestrateur

**K8S Kubelet** : process sur chaque nœud dialoguant avec le master, utilisé pour connaitre l’état des resources de la machine « slave » et des pods/containers sur ce nœud.  
Kubelet est responsable de l'état d'exécution de chaque nœud. l prend en charge le démarrage, l'arrêt, et la maintenance des conteneurs d'applications dirigé par le master

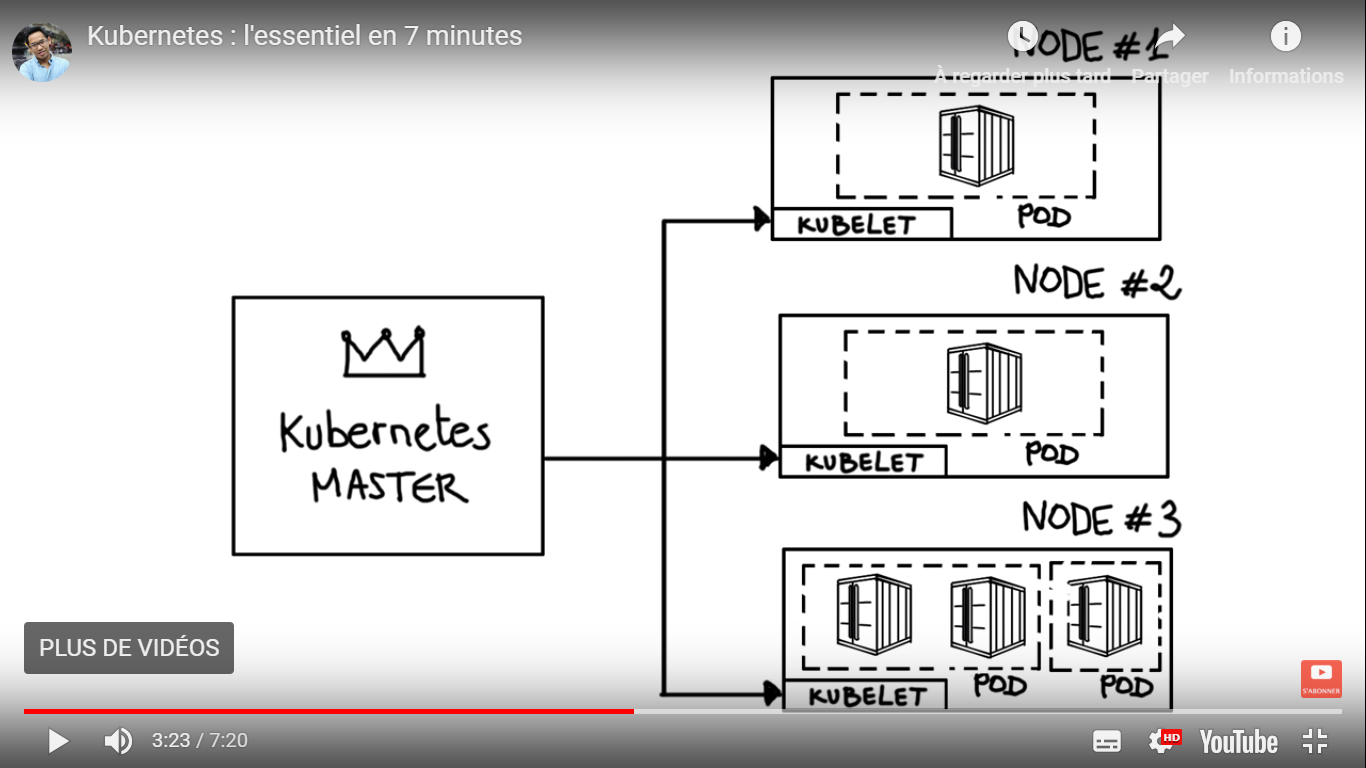
**K8S POD** : Un pod consiste en un ou plusieurs conteneurs qui ont la garantie d'être co-localisés sur une machine hôte et peuvent en partager les ressources. Chaque pod dans Kubernetes possède une adresse IP unique. Un pod peut définir un volume, comme un répertoire sur un disque local ou sur le réseau, et l'exposer aux conteneurs de ce pod. C’est l’unité de base de l'ordonnancement dans Kubernetes

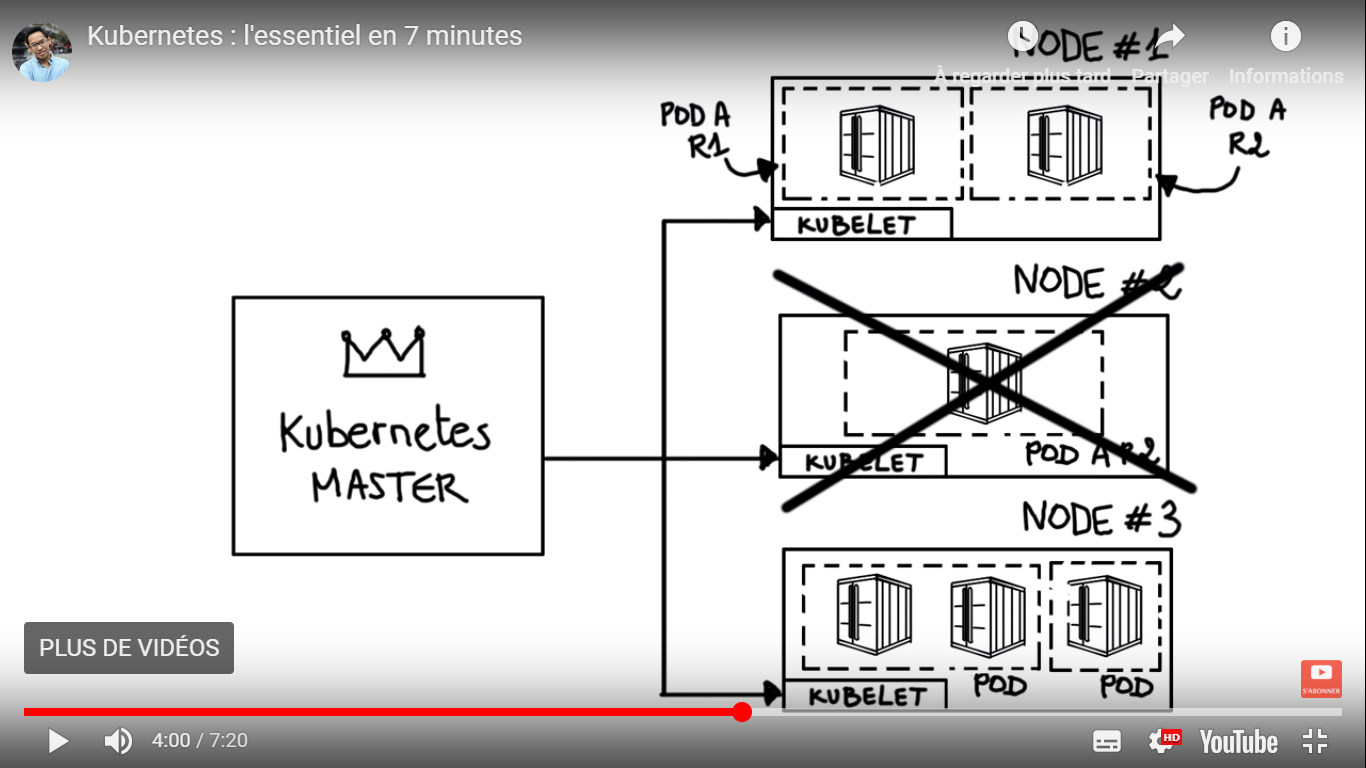
**K8S Service** : Un service Kubernetes est un groupe de pods travaillant ensemble

Il existe <> type de services :

* « normal » ( proxy entre les **pods** ips & une ip générique )
* « headless » ( proxy entre les **nodes** ips & une ip générique + utilise un DNS entry par node (et met à jour qd un pod déménage sur un autre node).  
  Utilisé en général pour les « statefulset » càd application avec volume persistent ( DB, ELK, Kafka, …)

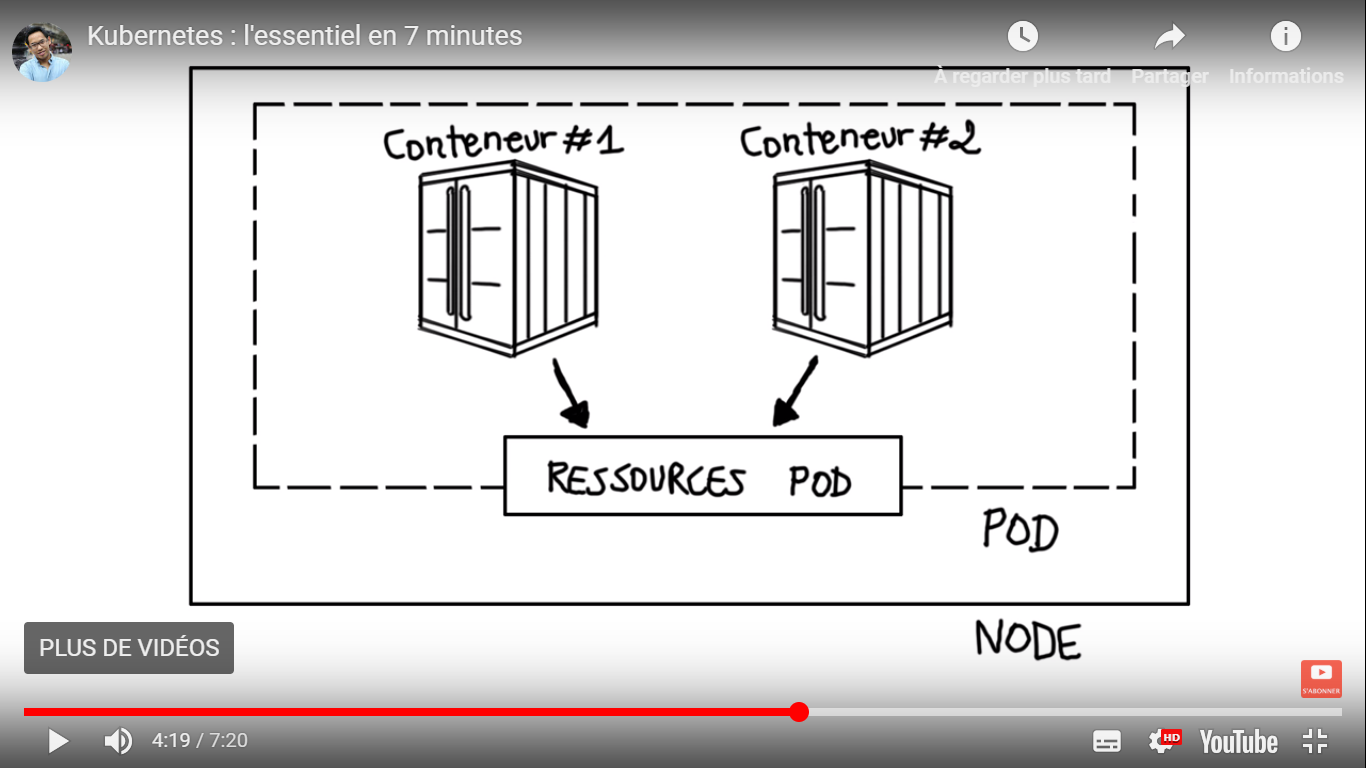
K8S gère le # d’exécution demandé et redémarre les pod ailleurs en cas de perte de nœud

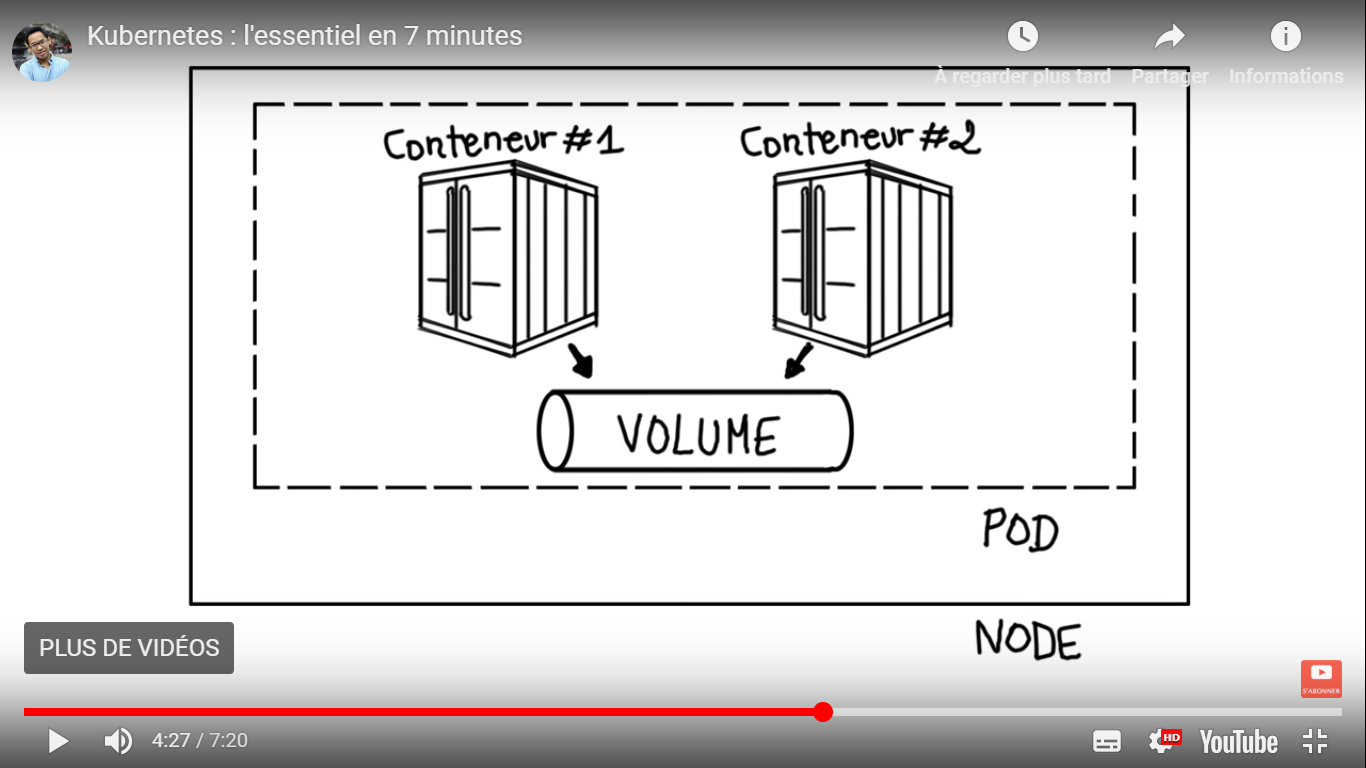




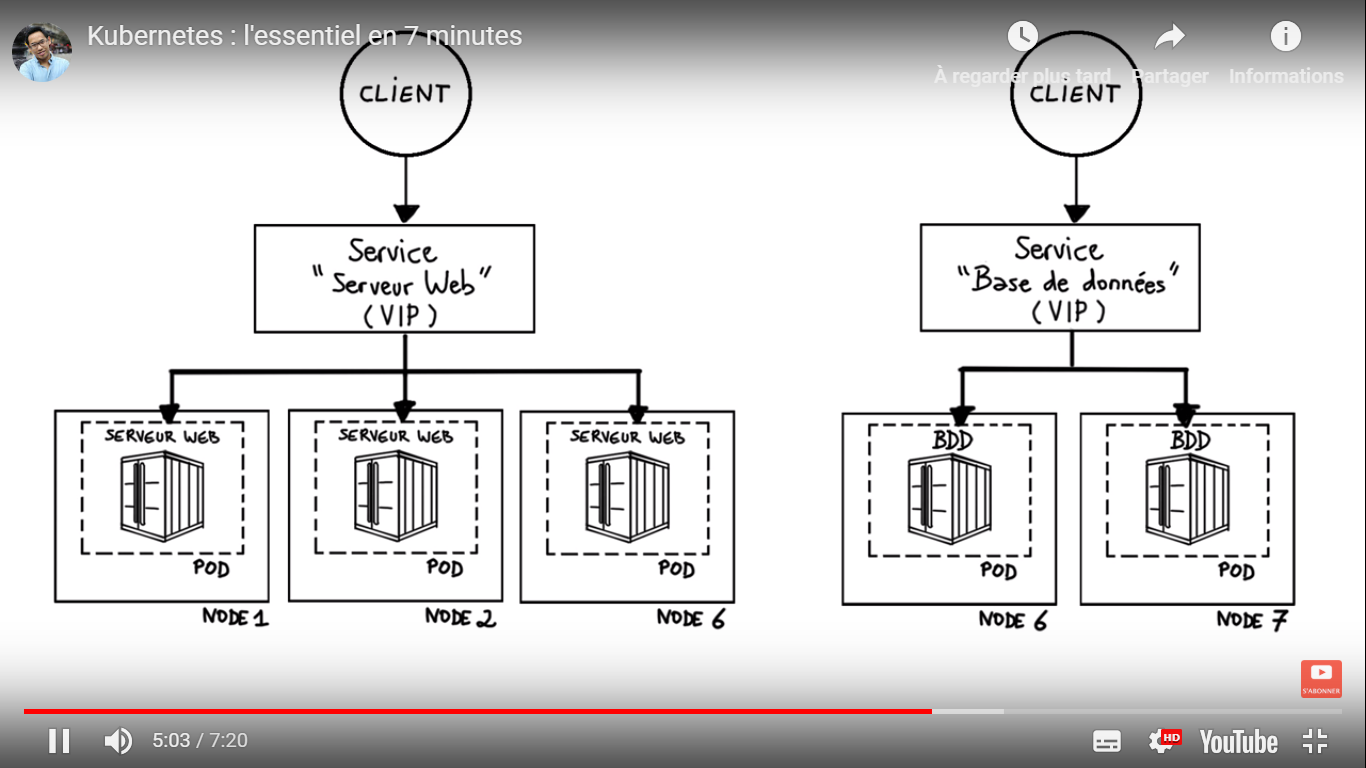
Les container d’un meme pod partagent des resources

* Meme IP
* Ports
* Volumes

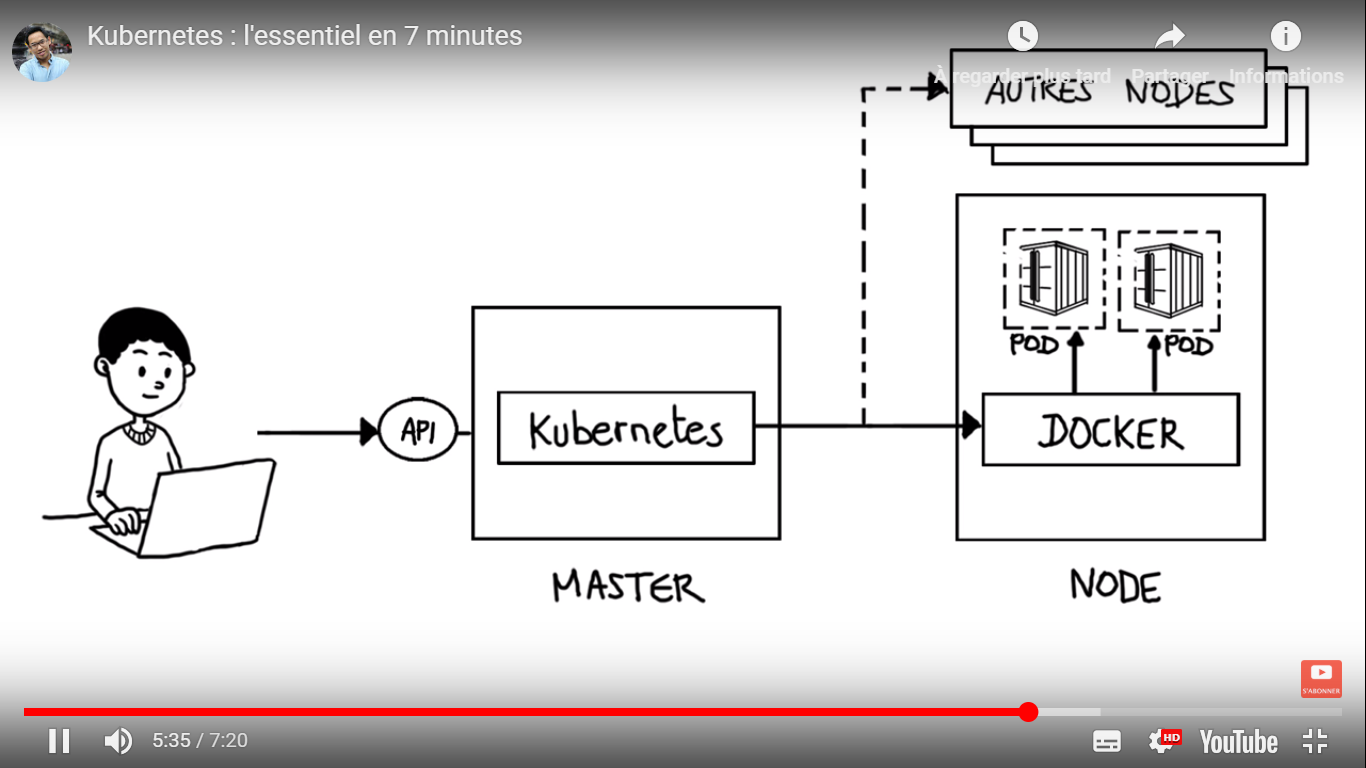




Via un système de virtual ip géré par K8S on peut gérer le load balancing



L’orchestrateur n’est plus une personne mais une application qui decide où executer les taches en fonction de la charge



**Kubernetes & affinités/label**

Imaginons qu’on ait 2 clusters – 1 Europe & 1 US

Dans chaque cluster on a des machines SSD & des machines spinning disks

Pour des services « StateFulSet » ( comme une db, kafka, elasticsearch ) on va

* Mettre les labels suivants sur les nodes :
  + Location : eu or usa
  + Disk : ssd or spinnning
* On va créer des Persistent volumes auquel on va mettre des metadata/labels
  + Disk : ssd or spinning
  + Location : eu or usa
* Les Pods devront avoir aussi des affinités – par exemple dans le cas d’un cluster ELK on va mettre des labels du genre – type : hot/warm, location (EU/USA) et on va mettre des affinités au niveau
  + Du pod ( s’assurer qu’un seul pod de type ELK tourne sur le nœud )
  + Du persistentVolumeClaim : Disk & location

**Attach pods to nodes**

### Step One: Attach label to the node

Run kubectl get nodes to get the names of your cluster’s nodes. Pick out the one that you want to add a label to, and then run kubectl label nodes <node-name> <label-key>=<label-value> to add a label to the node you’ve chosen. For example, if my node name is ‘kubernetes-foo-node-1.c.a-robinson.internal’ and my desired label is ‘disktype=ssd’, then I can run kubectl label nodes kubernetes-foo-node-1.c.a-robinson.internal disktype=ssd.

You can verify that it worked by re-running kubectl get nodes –show-labels and checking that the node now has a label. You can also use kubectl describe node “nodename” to see the full list of labels of the given node.

Then add a nodeSelector like so:

| [pods/pod-nginx.yaml](https://raw.githubusercontent.com/kubernetes/website/master/content/en/examples/pods/pod-nginx.yaml) |
| --- |
| apiVersion: v1  kind: Pod  metadata:  name: nginx  labels:  env: test  spec:  containers:  - name: nginx  image: nginx  imagePullPolicy: IfNotPresent  nodeSelector :  disktype : ssd |

**PV/PVC exemple**

[https ://docs.okd.io/latest/install\_config/storage\_examples/binding\_pv\_by\_label.html](https://docs.okd.io/latest/install_config/storage_examples/binding_pv_by_label.html)

### Persistent Volume with Labels

**apiVersion: v1**

**kind: PersistentVolume**

**metadata:**

**name: gluster-volume**

**labels: (1)**

**storage-tier: gold**

**aws-availability-zone: us-east-1**

**spec:**

**capacity:**

**storage: 2Gi**

**accessModes:**

**- ReadWriteMany**

**glusterfs:**

**endpoints: glusterfs-cluster (2)**

**path: myVol1**

**readOnly: false**

**persistentVolumeReclaimPolicy : Retain**

|  |  |
| --- | --- |
|  | 1. Use labels to identify common attributes or characteristics shared among volumes. In this case, we defined the Gluster volume to have a custom attribute (key) named **storage-tier** with a value of **gold** assigned. A claim will be able to select a PV with **storage-tier=gold** to match this PV. |
|  | 1. Endpoints define the Gluster trusted pool and are discussed below. |

### Persistent Volume Claim with Selectors

A claim with a **selector** stanza (see example below) attempts to match existing, unclaimed, and non-prebound PVs. The existence of a PVC selector ignores a PV’s capacity. However, **accessModes** are still considered in the matching criteria.

It is important to note that a claim must match **all** of the key-value pairs included in its **selector** stanza. If no PV matches the claim, then the PVC will remain unbound (Pending). A PV can subsequently be created and the claim will automatically check for a label match.

**apiVersion: v1**

**kind: PersistentVolumeClaim**

**metadata:**

**name: gluster-claim**

**spec:**

**accessModes:**

**- ReadWriteMany**

**resources:**

**requests:**

**storage: 1Gi**

**selector: (1)**

**matchLabels:**

**storage-tier: gold**

**aws-availability-zone: us-east-1**

|  |  |
| --- | --- |
|  | 1. The **selector** stanza defines all labels necessary in a PV in order to match this claim. |

Volume Endpoints

To attach the PV to the Gluster volume, endpoints should be configured before creating our objects.

*Example 3. Glusterfs-ep.yaml*

**apiVersion: v1**

**kind: Endpoints**

**metadata:**

**name: glusterfs-cluster**

**subsets :**

**- addresses :**

**- ip : 192.168.122.221**

**ports :**

**- port : 1**

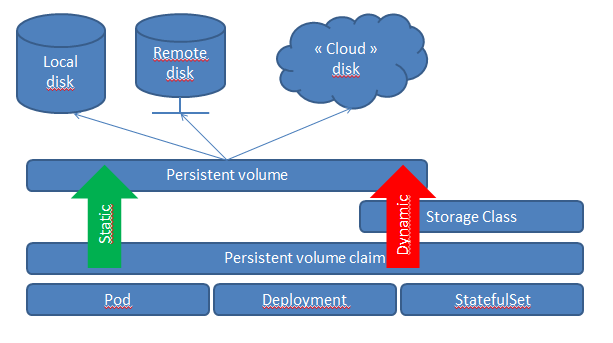
**- addresses :**

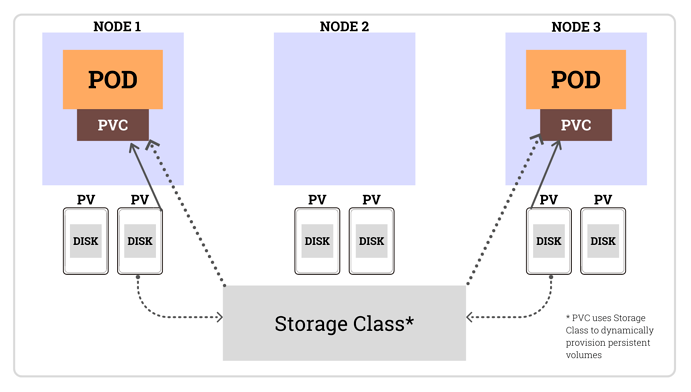
**- ip : 192.168.122.222**

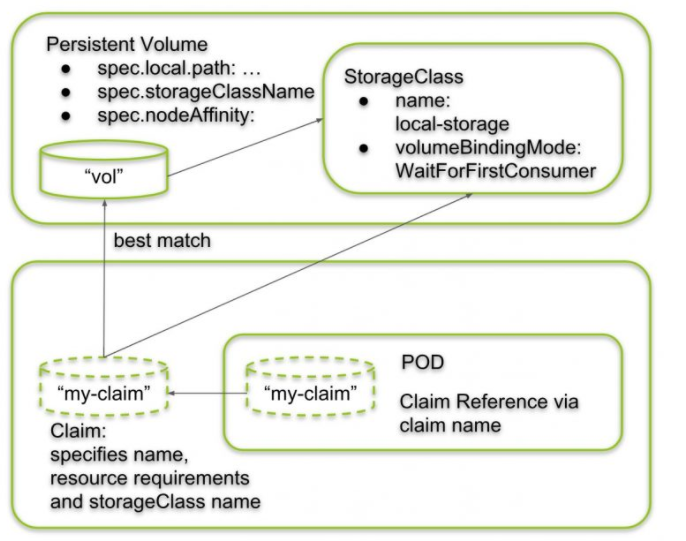
**ports :**

**- port : 1**

## Storage Model







Le persistent volume definit le lien vers le volume physique. On peut avoir

* Des disques locaux
* Des disques remote ( nfs,vpshere, ..)
* Des disques dans le cloud ( GCP, AWS EBS, … )

On va lier un pod / deployment / statefulset a un (ou +ieurs) disques via un PVC ( Persistent Volume Claim ). Ce PVC est soit statiquement lié à un PV soit à un storageClass. Le storageclass – soit sert de lien vert le PV, soit contient les infos necessaires pour dynamiquement créer un PV.

Note : un storageclass n’est pas toujours necessaire ( on peut lier le PV au PVC )

**Exemple de « storage » dans le cloud + pv static (pas de storageClass)**

1/ On crée un volume sur AWS EBS

*aws ec2 create-volume --region us-west-2 --availability-zone us-west-2a --size 5 --volume-type gp2*

{

    "AvailabilityZone":"us-west-2a",

    "Encrypted":false,

    "VolumeType":"gp2",

    "**VolumeId":"vol-47f59cce",**

    "State":"creating",

    "Iops":100,

    "SnapshotId":"",

    "CreateTime":"2016-07-29T21:57:43.343Z",

    "Size":5

}

2/ On crée un persistent volume qui utilise le volume créé en 1/

kind:PersistentVolume

apiVersion:v1

metadata:

  name:couchbase-pv

**labels:**

**type:amazonEBS**

**spec:**

  capacity:

    storage:5Gi

  accessModes:

    -ReadWriteOnce

**awsElasticBlockStore**:

**volumeID:vol-47f59cce**

    fsType:ext4

3/ On crée un PVC avec un label identique au PV pour faire le lien:

kind:PersistentVolumeClaim

apiVersion:v1

metadata:

**name:couchbase-pvc**

**labels:**

**type:amazonEBS**

spec:

  accessModes:

    -ReadWriteOnce

  resources:

    requests:

      storage:5Gi

4/ Usage du PVC (ici dans un ReplicationController). Le lien avec le PVC se fait avec le nom

apiVersion:v1

kind:ReplicationController

metadata:

  name:couchbase

spec:

  replicas:1

  template:

    metadata:

      name:couchbase-rc-pod

      labels:

        name:couchbase-rc-pod

        context:couchbase-pv

    spec:

      containers:

      -name:couchbase-rc-pod

        image:arungupta/couchbase

        volumeMounts:

        -mountPath:"/opt/couchbase/var"

          name:mypd

        ports:

        -containerPort:8091

        -containerPort:8092

        -containerPort:8093

        -containerPort:11210

      volumes:

      -name:mypd

**persistentVolumeClaim**:

**claimName:couchbase-pvc**