Kubernetes

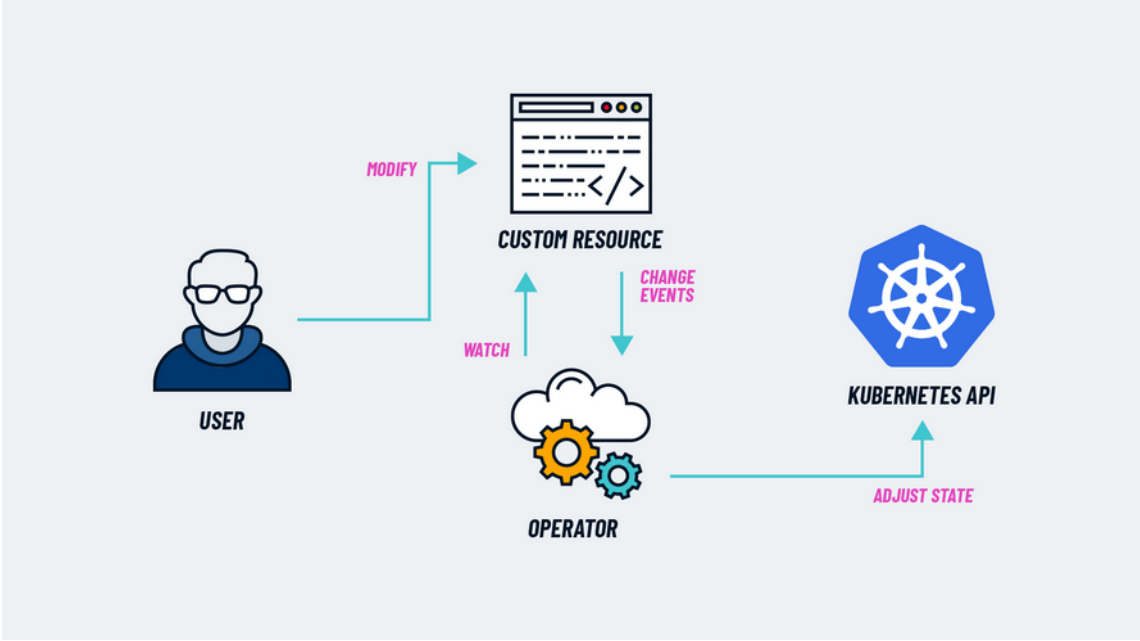
mardi 7 février 2023 09:27

https://kubernetes.io/fr/

https://fr.wikipedia.org/wiki/Kubernetes

https://wiki.archlinux.org/title/Kubernetes

Formateur : Loup FOURMENT et Mohamed AIJJOU

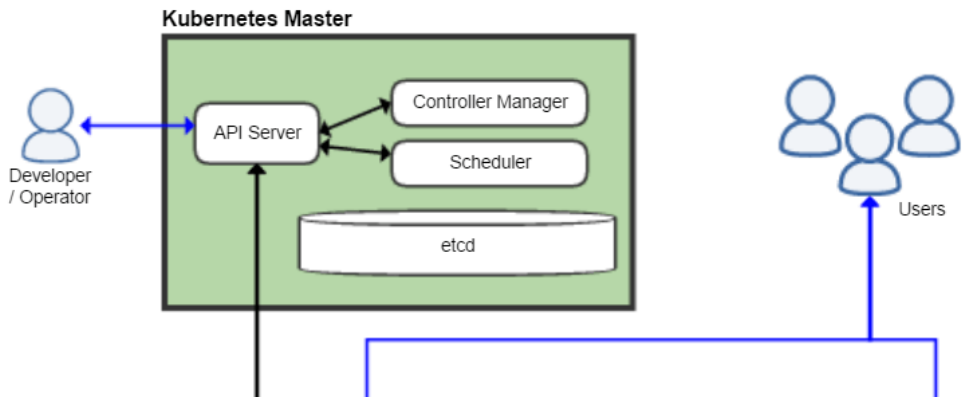


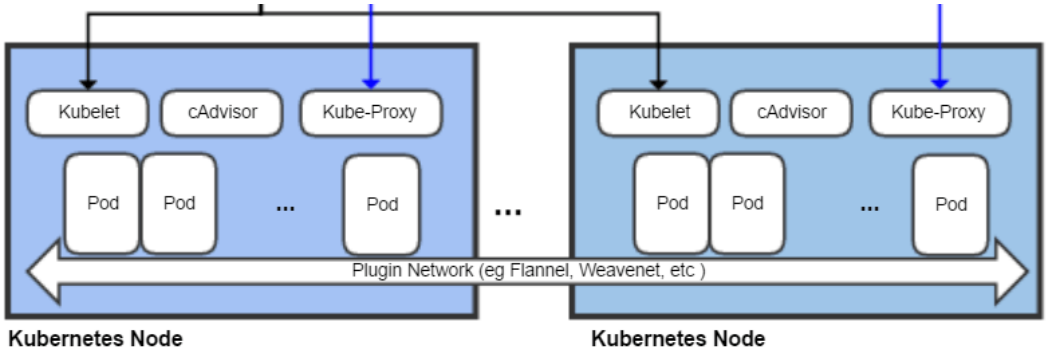
Kubernetes

Article Discussion

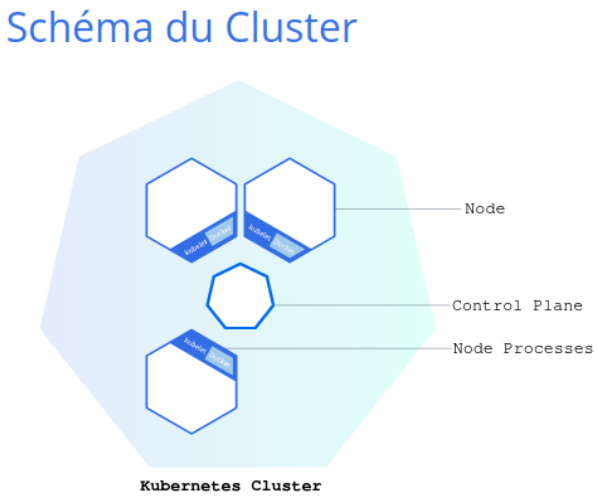
Kubernetes (communément appelé « K8s[2] ») est un système open source qui vise à fournir une « plate-forme permettant d'automatiser le déploiement, la montée en charge et la mise en œuvre de conteneurs d'application sur des grappes de serveurs[3] ». Il fonctionne avec toute une série de technologies de conteneurisation, et est souvent utilisé avec Docker. Il a été conçu à l'origine par Google, puis offert à la Cloud Native Computing Foundation.

Dans le principe, Kubernetes reprend les mêmes bases qu'Ansible : déployer des applications et les gérer





1ère partie : Découverte Minikube



Un pod représente une ou plusieurs application (un pod pourrait contenir le front et un autre le back par exemple)

Pods [modifier | modifier le code]

L'unité de base de l'ordonnancement dans Kubernetes est appelée « *pod* ». C'est une vue abstraite de composants conteneurisés. 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[14]. Chaque *pod* dans Kubernetes possède une adresse IP unique (à l'intérieur du cluster), qui permet aux applications d'utiliser les ports de la machine sans risque de conflit[15]. 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*[16]. Les *pods* peuvent être gérés manuellement au travers de l'API de Kubernetes. Leur gestion peut également être déléguée à un contrôleur[14].

Les *pods* sont rattachés au nœud qui les déploie jusqu'à leur expiration ou leur suppression. Si le nœud est défaillant, de nouveaux *pods* possédant les mêmes propriétés que les précédents seront déployés sur d'autres nœuds disponibles[17].

Tutoriel :

https://kubernetes.io/fr/docs/tutorials/

Utiliser Minikube pour créer un cluster : https://kubernetes.io/fr/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/

Objectifs

- Découvrez ce qu'est un cluster Kubernetes.
- Apprenez ce qu'est Minikube.
- Démarrez un cluster Kubernetes à l'aide d'un terminal en ligne.

Un cluster Kubernetes est constitué de deux types de ressources:

- Le **maître** coordonne le cluster.
- Les **nœuds** sont les serveurs qui exécutent des applications.

Cluster up and running

We already installed minikube for you. Check that it is properly installed, by running the *minikube version* command:

```
minikube version ↵
```

OK, we can see that minikube is in place.

Start the cluster, by running the *minikube start* command:

```
minikube start ↵
```

Great! You now have a running Kubernetes cluster in your online terminal. Minikube started a virtual machine for you, and a Kubernetes cluster is now running in that VM.

```
$ minikube version
minikube version: v1.18.0
commit: ec61815d60f66a6e4f6353030a40b12362557caa-dirty
$ minikube start
* minikube v1.18.0 on Ubuntu 18.04 (amd64)
* Using the none driver based on existing profile

X The requested memory allocation of 2200MiB does not leave room for system overhe
ad (total system memory: 2460MiB). You may face stability issues.
* Suggestion: Start minikube with less memory allocated: 'minikube start --memory=
2200mb'

* Starting control plane node minikube in cluster minikube
* Running on localhost (CPUs=2, Memory=2460MB, Disk=194868MB) ...
* OS release is Ubuntu 18.04.5 LTS
* Preparing Kubernetes v1.20.2 on Docker 19.03.13 ...
  - kubelet.resolv-conf=/run/systemd/resolve/resolv.conf
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring local host environment ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v4
* Enabled addons: default-storageclass, storage-provisioner
* Done! kubectl is now configured to use "minikube" cluster and "default" namespac
e by default
$
```

Cluster version

To interact with Kubernetes during this bootcamp we'll use the command line interface, kubectl. We'll explain kubectl in detail in the next modules, but for now, we're just going to look at some cluster information. To check if kubectl is installed you can run the *kubectl version* command:

```
kubectl version ↵
```

```
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"20", GitVersion:"v1.20.4", GitCommi
t:"e87da0bd6e03ec3fea7933c4b5263d151aafd07c", GitTreeState:"clean", BuildDate:"202
1-02-18T16:12:00Z", GoVersion:"go1.15.8", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"20", GitVersion:"v1.20.2", GitCommi
t:"faecb196815e248d3ecfb03c680a4507229c2a56", GitTreeState:"clean", BuildDate:"202
1-01-13T13:20:00Z", GoVersion:"go1.15.5", Compiler:"gc", Platform:"linux/amd64"}
$
```

Cluster details

Let's view the cluster details. We'll do that by running *kubectl cluster-info*:

```
kubectl cluster-info ↵
```

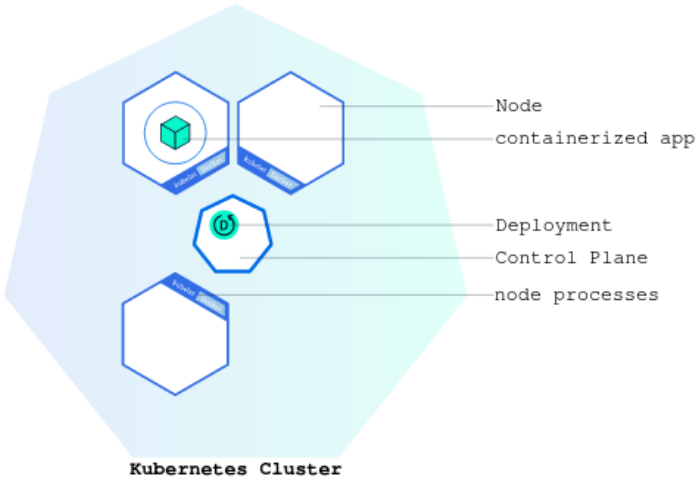
During this tutorial, we'll be focusing on the command line for deploying and exploring our application. To view the nodes in the cluster, run the *kubectl get nodes* command:

```
kubectl get nodes ↵
```

This command shows all nodes that can be used to host our applications. Now we have only one node, and we can see that its status is ready (it is ready to accept applications for deployment).

```
Kubernetes Bootcamp Terminal

$
$ minikube version
minikube version: v1.18.0
commit: ec61815d60f66a6e4f6353030a40b12362557caa-dirty
$ kubectl cluster-info
Kubernetes control plane is running at https://10.0.0.81:8443
KubeDNS is running at https://10.0.0.81:8443/api/v1/namespaces/kube-system/service
s/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
$ kubectl get nodes
NAME       STATUS   ROLES                  AGE    VERSION
minikube   Ready    control-plane,master   3m7s   v1.20.2
$
```

Deploying your first app on Kubernetes



< | Module 2 - Deploy an app 58:32

kubectl basics

Like minikube, kubectl comes installed in the online terminal. Type kubectl in the terminal to see its usage. The common format of a kubectl command is: kubectl action resource. This performs the specified action (like create, describe) on the specified resource (like node, container). You can use `--help` after the command to get additional info about possible parameters (`kubectl get nodes --help`).

Check that kubectl is configured to talk to your cluster, by running the `kubectl version` command:

```
kubectl version ↵
```

```
$ sleep 1; launch.sh
Starting Kubernetes. This is expected to take less than a minute.....
Kubernetes Started
$
$
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"20", GitVersion:"v1.20.4", GitCommi
t:"e87da0bd6e03ec3fea7933c4b5263d151aafd07c", GitTreeState:"clean", BuildDate:"202
1-02-18T16:12:00Z", GoVersion:"go1.15.8", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"20", GitVersion:"v1.20.2", GitCommi
t:"faecb196815e248d3ecfb03c680a4507229c2a56", GitTreeState:"clean", BuildDate:"202
1-01-13T13:20:00Z", GoVersion:"go1.15.5", Compiler:"gc", Platform:"linux/amd64"}
$
```

OK, kubectl is installed and you can see both the client and the server versions.

To view the nodes in the cluster, run the `kubectl get nodes` command:

```
kubectl get nodes ↵
```

Here we see the available nodes (1 in our case). Kubernetes will choose where to deploy our application based on Node available resources.

```
$ kubectl get nodes
NAME       STATUS   ROLES                  AGE     VERSION
minikube   Ready    control-plane,master   3m23s   v1.20.2
$
```

Deploy our app

Let's deploy our first app on Kubernetes with the `kubectl create deployment` command. We

`kubectl create deployment` command. We need to provide the deployment name and app image location (include the full repository url for images hosted outside Docker hub).

```
kubectl create deployment kubernetes-
bootcamp --image=gcr.io/google-samples
/kubernetes-bootcamp:v1 ✓
```

```
minikube   Ready    control-plane,master   3m23s   v1.20.2
$ kubectl create deployment kubernetes-bootcamp --image=gcr.io/google-samples/kube
rnetes-bootcamp:v1
deployment.apps/kubernetes-bootcamp created
```

Great! You just deployed your first application by creating a deployment. This performed a few things for you:

- searched for a suitable node where an instance of the application could be run (we have only 1 available node)
- scheduled the application to run on that Node
- configured the cluster to reschedule the instance on a new Node when needed

To list your deployments use the `get deployments` command:

```
kubectl get deployments ↵
```

```
$ kubectl get deployments
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   1/1     1            1           2m19s
$
```

View our app

Pods that are running inside Kubernetes are running on a private, isolated network. By default they are visible from other pods and services within the same kubernetes cluster, but not outside that network. When we use `kubectl`, we're interacting through an API endpoint to communicate with our application.

We will cover other options on how to expose your application outside the kubernetes cluster in Module 4.

The `kubectl` command can create a proxy that will forward communications into the cluster-wide, private network. The proxy can be terminated by

pressing control-C and won't show any output while its running.

We will open a second terminal window to run the proxy.

```
echo -e "\n\n\n\e[92mStarting Proxy. After starting it will not output a response. Please click the first Terminal Tab\n";
kubectl proxy ↵
```

```
$ kubetctl versionecho -e "\n\n\n\e[92mStarting Proxy. After starting it will not
 Please click the first Terminal Tab\n";

Command 'kubetctl' not found, did you mean:

  command 'kubectl' from snap kubectl (1.20.4)

See 'snap info <snapname>' for additional versions.

$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```

We now have a connection between our host (the online terminal) and the Kubernetes cluster. The proxy enables direct access to the API from these terminals.

You can see all those APIs hosted through the proxy endpoint. For example, we can query the version directly through the API using the `curl` command:

```
curl http://localhost:8001/version ↵
```

```
$ curl http://localhost:8001/version
{
  "major": "1",
  "minor": "20",
  "gitVersion": "v1.20.2",
  "gitCommit": "faecb196815e248d3ecfb03c680a4507229c2a56",
  "gitTreeState": "clean",
  "buildDate": "2021-01-13T13:20:00Z",
  "goVersion": "go1.15.5",
  "compiler": "gc",
  "platform": "linux/amd64"
}$
```

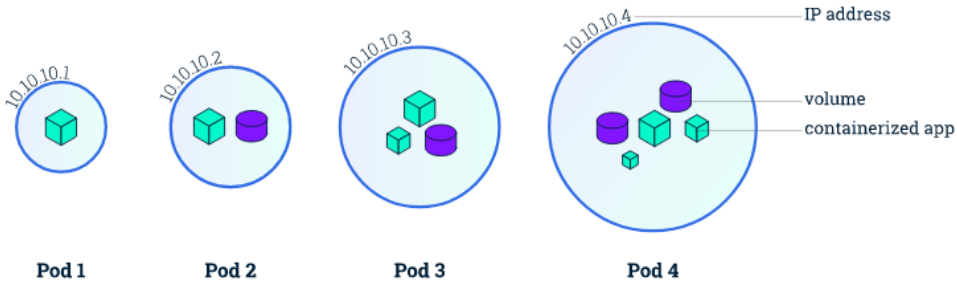
Viewing Pods and Nodes

Objectives

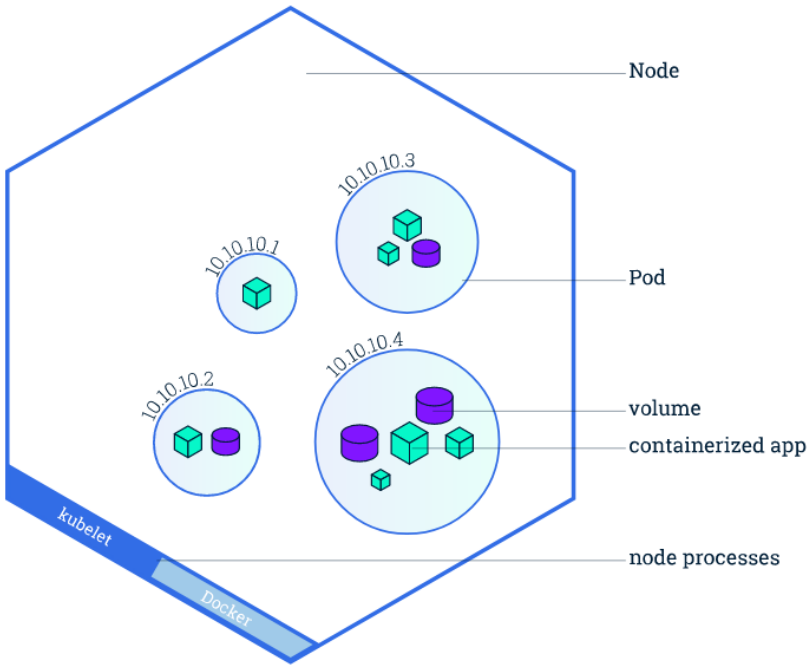
Objectives

- Learn about Kubernetes Pods.
- Learn about Kubernetes Nodes.
- Troubleshoot deployed applications.

Pods overview



Node overview



Troubleshooting with kubectl

In Module 2, you used Kubectl command-line interface. You'll continue to use it in Module 3 to get information about deployed applications and their environments. The most common operations can be done with the following kubectl commands:

- **kubectl get** - list resources
- **kubectl describe** - show detailed information about a resource
- **kubectl logs** - print the logs from a container in a pod
- **kubectl exec** - execute a command on a container in a pod

You can use these commands to see when applications were deployed, what their current statuses are, where they are running and what their configurations are.

Now that we know more about our cluster components and the command line, let's explore our application.

Step 1 Check

application configuration

Let's verify that the application we deployed in the previous scenario is running. We'll use the `kubectl get` command and look for existing Pods:

```
kubectl get pods ↵
```

If no pods are running then it means the interactive environment is still reloading it's previous state. Please wait a couple of seconds and list the Pods again. You can continue once you see the one Pod running.

```
NAME                                   READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-fb5c67579-tr6rc   1/1     Running   0          56s
$
```

Next, to view what containers are inside that Pod and what images are used to build those containers we run the `describe pods` command:

```
kubectl describe pods ↵
```

We see here details about the Pod's container: IP address, the ports used and a list of events related to the lifecycle of the Pod.

The output of the `describe` command is extensive and covers some concepts that we didn't explain yet, but don't worry, they will become familiar by the end of this bootcamp.

```
$ kubectl get pods
NAME                                   READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-fb5c67579-tr6rc   1/1     Running   0          56s
$ kubectl describe pods
Name:         kubernetes-bootcamp-fb5c67579-tr6rc
Namespace:    default
Priority:     0
Node:         minikube/10.0.0.16
Start Time:   Tue, 07 Feb 2023 09:27:15 +0000
Labels:       app=kubernetes-bootcamp
              pod-template-hash=fb5c67579
Annotations:  <none>
Status:       Running
IP:           172.18.0.5
IPs:
  IP:           172.18.0.5
Controlled By:  ReplicaSet/kubernetes-bootcamp-fb5c67579
```

Using a Service to Expose Your App

Objectives

- Learn about a Service in Kubernetes
- Understand how labels and LabelSelector objects relate to a Service
- Expose an application outside a Kubernetes cluster using a Service

Overview of Kubernetes Services

Kubernetes Pods are mortal. Pods have a lifecycle. When a worker node dies, the Pods running on the Node are also lost. A ReplicaSet might then dynamically drive the cluster back to the desired state via the creation of new Pods to keep your application running. As another example, consider an image-processing backend with 3 replicas. Those replicas are exchangeable; the front-end system should not care about backend replicas or even if a Pod is lost and recreated. That said, each Pod in a Kubernetes cluster has a unique IP address, even Pods on the same Node, so there needs to be a way of automatically reconciling changes among Pods so that your applications continue to function.

A Service in Kubernetes is an abstraction which defines a logical set of Pods and a policy by which to access them. Services enable a loose coupling between dependent Pods. A Service is defined using YAML (preferred) or JSON, like all Kubernetes objects. The set of Pods targeted by a Service is usually determined by a *LabelSelector* (see below for why you might want a Service without including a `selector` in the spec).

Although each Pod has a unique IP address, those IPs are not exposed outside the cluster without a Service. Services allow your applications to receive traffic. Services can be exposed in different ways by specifying a `type` in the ServiceSpec:

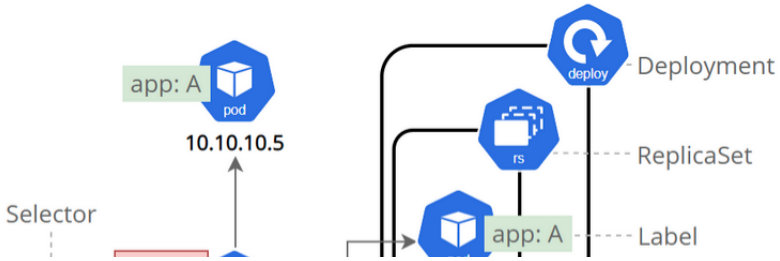
- *ClusterIP* (default) - Exposes the Service on an internal IP in the cluster. This type makes the Service only reachable from within the cluster.
- *NodePort* - Exposes the Service on the same port of each selected Node in the cluster using NAT. Makes a Service accessible from outside the cluster using `<NodeIP>:<NodePort>`. Superset of ClusterIP.
- *LoadBalancer* - Creates an external load balancer in the current cloud (if supported) and assigns a fixed, external IP to the Service. Superset of NodePort.
- *ExternalName* - Maps the Service to the contents of the `externalName` field (e.g. `foo.bar.example.com`), by returning a `CNAME` record with its value. No proxying of any kind is set up. This type requires v1.7 or higher of `kube-dns`, or CoreDNS version 0.0.8 or higher.

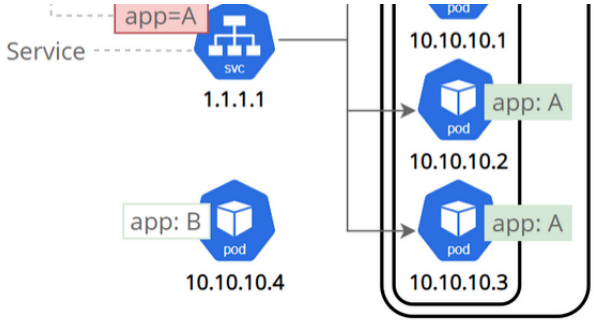
Services and Labels

A Service routes traffic across a set of Pods. Services are the abstraction that allows pods to die and replicate in Kubernetes without impacting your application. Discovery and routing among dependent Pods (such as the frontend and backend components in an application) are handled by Kubernetes Services.

Services match a set of Pods using labels and selectors, a grouping primitive that allows logical operation on objects in Kubernetes. Labels are key/value pairs attached to objects and can be used in any number of ways:

- Designate objects for development, test, and production
- Embed version tags
- Classify an object using tags





Step 1 Create a new service

Let's verify that our application is running. We'll use the `kubectl get` command and look for existing Pods:

```
kubectl get pods ↵
```

If no pods are running then it means the interactive environment is still reloading its previous state. Please wait a couple of seconds and list the Pods again. You can continue once you see the one Pod running.

```
$ kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-fb5c67579-ngrnh   1/1     Running   0          2m10s
$ kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   2m44s
```

```
kubectl get services ↵
```

We have a Service called kubernetes that is created by default when minikube starts the cluster. To create a new service and expose it to external traffic we'll use the expose command with NodePort as parameter (minikube does not support the LoadBalancer option yet).

```
$ kubectl expose deployment/kubernetes-bootcamp --type="NodePort" --port 8080
service/kubernetes-bootcamp exposed
$ kubectl get services
NAME                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          A
```

```
GE
kubernetes            ClusterIP   10.96.0.1       <none>        443/TCP          6
m37s
kubernetes-bootcamp   NodePort    10.99.170.251   <none>        8080:31789/TCP   4
s
$
```

To find out what port was opened externally (by the NodePort option) we'll run the `describe service` command:

```
kubectl describe services/kubernetes-
bootcamp ↵
```

Create an environment variable called NODE_PORT that has the value of the Node port assigned:

```
export NODE_PORT=$(kubectl get
services/kubernetes-bootcamp -o go-
template='{{(index .spec.ports
0).nodePort}}')
echo NODE_PORT=$NODE_PORT ↵
```

```
$ kubectl describe services/kubernetes-bootcamp
Name:                     kubernetes-bootcamp
Namespace:                default
Labels:                   app=kubernetes-bootcamp
Annotations:              <none>
Selector:                 app=kubernetes-bootcamp
Type:                     NodePort
IP Families:              <none>
IP:                       10.104.173.224
IPs:                      10.104.173.224
Port:                     <unset>  8080/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  31257/TCP
Endpoints:                172.18.0.5:8080
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
$
```

```
$ export NODE_PORT=$(kubectl get services/kubernetes-bootcamp -o go-template='{{(
ndex .spec.ports 0).nodePort}}')
$ echo NODE_PORT=$NODE_PORT
NODE_PORT=31257
$
```

```
$ curl $(minikube ip):$NODE_PORT
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-fb5c67579-4dbrt | v=1
$
```

Now we can test that the app is exposed outside of the cluster using `curl`, the IP of the Node and the externally exposed port:

the externally exposed port.

```
curl $(minikube ip):$NODE_PORT ✓
```

And we get a response from the server. The Service is exposed.

Step 2: Using labels

The Deployment created automatically a label for our Pod. With `describe deployment` command you can see the name of the label:

```
kubectl describe deployment ✓
```

Let's use this label to query our list of Pods. We'll use the `kubectl get pods` command with -l as a parameter, followed by the label values:

```
kubectl get pods -l app=kubernetes-
bootcamp ✓
```

You can do the same to list the existing services:

```
kubectl get services -l
app=kubernetes-bootcamp ↩
```

```
$ kubectl get pods -l app=kubernetes-bootcamp
NAME                                  READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-fb5c67579-4dbrt   1/1     Running   0          4m51s
$ kubectl get services -l app=kubernetes-bootcamp
NAME                  TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)          A
GE
kubernetes-bootcamp   NodePort   10.104.173.224   <none>        8080:31257/TCP   5
m7s
$
```

Get the name of the Pod and store it in the POD_NAME environment variable:

```
export POD_NAME=$(kubectl get pods -o
go-template --template '{{range
.items}}{{.metadata.name}}{{"\n"}}
{{end}}')
echo Name of the Pod: $POD_NAME ✓
```

To apply a new label we use the label command

followed by the object type, object name and the new label:

```
kubectl label pods $POD_NAME
version=v1 ↵
```

```
$ export POD_NAME=$(kubectl get pods -o go-template --template '{{range .items}}{{
.metadata.name}}{{"\n"}}{{end}}')
$ echo Name of the Pod: $POD_NAME
Name of the Pod: kubernetes-bootcamp-fb5c67579-4dbrt
$ kubectl label pods $POD_NAME version=v1
pod/kubernetes-bootcamp-fb5c67579-4dbrt labeled
$
```

This will apply a new label to our Pod (we pinned the application version to the Pod), and we can check it with the describe pod command:

```
kubectl describe pods $POD_NAME ↵
```

We see here that the label is attached now to our Pod. And we can query now the list of pods using the new label:

```
kubectl get pods -l version=v1 ↵
```

And we see the Pod.

```
    Port:           8080/TCP
$ kubectl describe pods $POD_NAME
Name:             kubernetes-bootcamp-fb5c67579-4dbrt
Namespace:        default
Priority:         0
Node:             minikube/10.0.0.6
Start Time:       Tue, 07 Feb 2023 10:24:53 +0000
Labels:           app=kubernetes-bootcamp
                  pod-template-hash=fb5c67579
                  version=v1
Annotations:      <none>
Status:           Running
IP:               172.18.0.5
IPs:
  IP:           172.18.0.5
Controlled By:  ReplicaSet/kubernetes-bootcamp-fb5c67579
Containers:
```

```
$ kubectl get pods -l version=v1
NAME                                  READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-fb5c67579-4dbrt   1/1     Running   0          9m15s
$
```

Step 3 Deleting a service

To delete Services you can use the `delete service` command. Labels can be used also here:

```
kubectl delete service -l
app=kubernetes-bootcamp ↵
```

Confirm that the service is gone:

```
kubectl get services ↵
```

```
$ kubectl delete service -l app=kubernetes-bootcamp
service "kubernetes-bootcamp" deleted
$ kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   11m
$
```

This confirms that our Service was removed. To confirm that route is not exposed anymore you can `curl` the previously exposed IP and port:

```
curl $(minikube ip):$NODE_PORT ✓
```

This proves that the app is not reachable anymore from outside of the cluster. You can confirm that the app is still running with a curl inside the pod:

```
kubectl exec -ti $POD_NAME -- curl
localhost:8080 ↵
```

We see here that the application is up. This is because the Deployment is managing the application. To shut down the application, you

would need to delete the Deployment as well.

```
$ curl $(minikube ip):$NODE_PORT
curl: (7) Failed to connect to 10.0.0.6 port 31257: Connection refused
$ kubectl exec -ti $POD_NAME -- curl localhost:8080
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-fb5c67579-4dbrt | v=1
$
```

Installation :

https://kubernetes.io/fr/docs/setup/

https://minikube.sigs.k8s.io/docs/start/

1 Installation

Click on the buttons that describe your target platform. For other architectures, see the release page for a complete list of minikube binaries.

Operating system: Linux | macOS | Windows

Architecture: x86-64 | ARM64 | ARMv7 | ppc64 | S390x

Release type: Stable | Beta

Installer type: Binary download | Debian package | RPM package

To install the latest minikube **stable** release on **x86-64 Linux** using **binary download**:

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

```
vel@vel-Precision-3650-Tower:~$ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 77.3M  100 77.3M    0     0  64.5M      0  0:00:01  0:00:01 --:--:-- 64.6M
vel@vel-Precision-3650-Tower:~$ sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

2 Start your cluster

From a terminal with administrator access (but not logged in as root), run:

```
minikube start
```

If minikube fails to start, see the drivers page for help setting up a compatible container or virtual-machine manager.

```
vel@vel-Precision-3650-Tower:~$ minikube start
😄  minikube v1.29.0 sur Ubuntu 22.04
✨  Choix automatique du pilote docker. Autres choix: qemu2, virtualbox, ssh, none
📌  Utilisation du pilote Docker avec le privilège root
👍  Démarrage du noeud de plan de contrôle minikube dans le cluster minikube
🚜  Extraction de l'image de base...
💾  Téléchargement du préchargement de Kubernetes v1.26.1...
    > preloaded-images-k8s-v18-v1...:  397.05 MiB / 397.05 MiB  100.00% 60.00 M
    > gcr.io/k8s-minikube/kicbase...:  407.19 MiB / 407.19 MiB  100.00% 32.34 M
🔥  Création de docker container (CPUs=2, Memory=7900Mo) ...
🐳  Préparation de Kubernetes v1.26.1 sur Docker 20.10.23...
    ▪ Génération des certificats et des clés
    ▪ Démarrage du plan de contrôle ...
    ▪ Configuration des règles RBAC ...
🔗  Configuration de bridge CNI (Container Networking Interface)...
    ▪ Utilisation de l'image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Modules activés: storage-provisioner, default-storageclass
🔎  Vérification des composants Kubernetes...
💡  kubectl introuvable. Si vous en avez besoin, essayez : 'minikube kubectl -- get pods -A'
🏄  Terminé ! kubectl est maintenant configuré pour utiliser "minikube" cluster et espace de noms "default" par défaut.
```

3 Interact with your cluster

If you already have kubectl installed, you can now use it to access your shiny new cluster:

```
kubectl get po -A
```

Alternatively, minikube can download the appropriate version of kubectl and you should be able to use it like this:

```
minikube kubectl -- get po -A
```

You can also make your life easier by adding the following to your shell config:

```
alias kubectl="minikube kubectl --"
```

Initially, some services such as the storage-provisioner, may not yet be in a Running state. This is a normal condition during cluster bring-up, and will resolve itself momentarily. For additional insight into your cluster state, minikube bundles the Kubernetes Dashboard, allowing you to get easily acclimated to your new environment:

```
minikube dashboard
```

Pour allez plus loin dans le tutoriel, il faut installer kubectl

Installer Kubectl (deux façon de faire : curl et snap) :

https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/

Install kubectl binary with curl on Linux

1. Download the latest release with the command:

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/
```

> **Note:**
> To download a specific version, replace the `$(curl -L -s https://dl.k8s.io/release/stable.txt)` portion of the command with the specific version.
>
> For example, to download version v1.26.0 on Linux, type:
>
> ```
> curl -LO https://dl.k8s.io/release/v1.26.0/bin/linux/amd64/kubectl
> ```

2. Validate the binary (optional)

   Download the kubectl checksum file:

   ```
   curl -LO "https://dl.k8s.io/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.
   ```

   Validate the kubectl binary against the checksum file:

   ```
   echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check
   ```

   If valid, the output is:

   ```
   kubectl: OK
   ```

   If the check fails, `sha256` exits with nonzero status and prints output similar to:

   ```
   kubectl: FAILED
   ```

```
sha256sum: WARNING: 1 computed checksum did NOT match
```

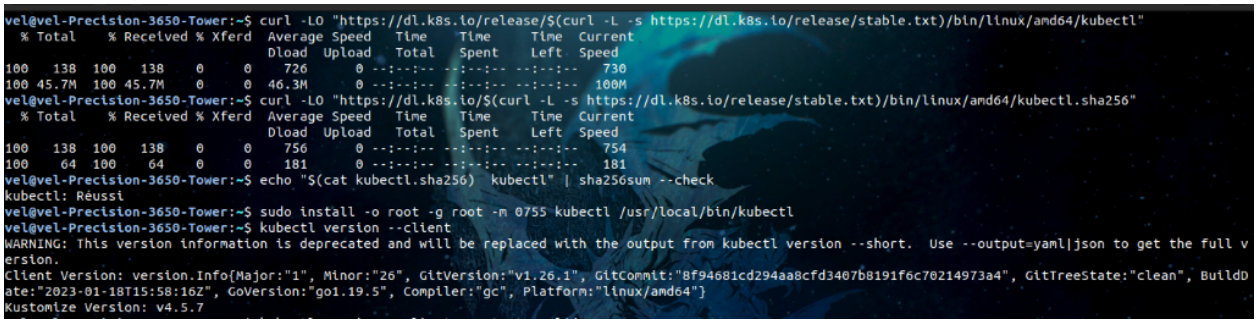
> **Note:** Download the same version of the binary and checksum.

3. Install kubectl

```
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

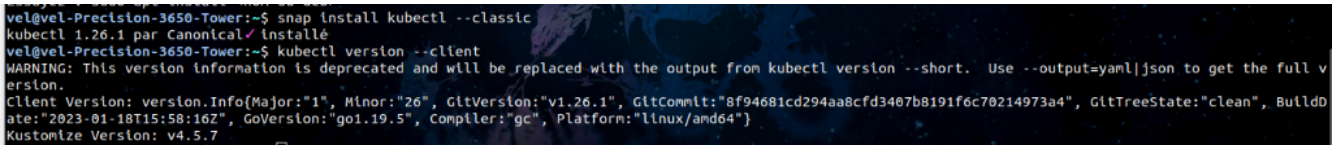
4. Test to ensure the version you installed is up-to-date:

```
kubectl version --client
```



Autre façon d'installer kubectl : snap

```
snap install kubectl --classic
kubectl version --client
```



Une fois la commande kubectl installé : on peut reprendre l'initiation à MInikube :

Alternatively, minikube can download the appro

```
minikube kubectl -- get po -A
```

You can also make your life easier by adding the

```
alias kubectl="minikube kubectl --"
```

Initially, some services such as the storage-prov
cluster bring-up, and will resolve itself moment
Kubernetes Dashboard, allowing you to get easi

```
minikube dashboard
```

```
vel@vel-Precision-3650-Tower:~$ kubectl get po -A
NAMESPACE     NAME                               READY   STATUS    RESTARTS      AGE
kube-system   coredns-787d4945fb-nzfh9           1/1     Running   0             12m
kube-system   etcd-minikube                      1/1     Running   0             12m
kube-system   kube-apiserver-minikube            1/1     Running   0             12m
kube-system   kube-controller-manager-minikube   1/1     Running   0             12m
kube-system   kube-proxy-trfzn                   1/1     Running   0             12m
kube-system   kube-scheduler-minikube            1/1     Running   0             12m
kube-system   storage-provisioner                1/1     Running   1 (12m ago)   12m
```

Pour ce tp, je ne ferais pas l'alias

Déploiement du dashboard :

```
vel@vel-Precision-3650-Tower:~$ minikube dashboard
🔌  Activation du tableau de bord...
    ▪ Utilisation de l'image docker.io/kubernetesui/dashboard:v2.7.0
    ▪ Utilisation de l'image docker.io/kubernetesui/metrics-scraper:v1.0.8
💡  Certaines fonctionnalités du tableau de bord nécessitent le module metrics-server. Pour activer toutes les fonctionnalités, veuillez exécuter :

        minikube addons enable metrics-server

🤔  Vérification de l'état du tableau de bord...
🚀  Lancement du proxy...
🤔  Vérification de l'état du proxy...
🎉  Ouverture de http://127.0.0.1:33073/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ dans votre navigateur par défaut...

Gtk-Message: 12:03:45.257: Not loading module "atk-bridge": The functionality is provided by GTK natively. Please try to not load it.
```

Vérification des containers : on s'aperçois que notre dashboard s'exécute via dockers

```
vel@vel-Precision-3650-Tower:~$ docker ps
CONTAINER ID   IMAGE                                 COMMAND                  CREATED          STATUS          PORTS                                                                                                                                  NA
MES
4d089f0201a2   gcr.io/k8s-minikube/kicbase:v0.0.37   "/usr/local/bin/entr…"   21 minutes ago   Up 21 minutes   127.0.0.1:49157->22/tcp, 127.0.0.1:49156->2376/tcp, 127.0.0.1:49155->5000/tcp, 127.0.0.1:49154->8443/tcp, 127.0.0.1:49153->32443/tcp   minikube
36011283a72a   portainer/portainer-ce:2.9.3          "/portainer"             5 days ago       Up 3 hours      0.0.0.0:8000->8000/tcp, :::8000->8000/tcp, 0.0.0.0:9443->9443/tcp, :::9443->9443/tcp, 9000/tcp                                        portainer
```

Déploiement application :

4 Deploy applications

Service | LoadBalancer | Ingress

Create a sample deployment and expose it on port 8080:

```
kubectl create deployment hello-minikube --image=kicbase/echo-server:1.0
kubectl expose deployment hello-minikube --type=NodePort --port=8080
```

It may take a moment, but your deployment will soon show up when you run:

```
kubectl get services hello-minikube
```

The easiest way to access this service is to let minikube launch a web browser for you:

```
minikube service hello-minikube
```

Service

Alternatively, use kubectl to forward the port:

```
kubectl port-forward service/hello-minikube 7080:8080
```

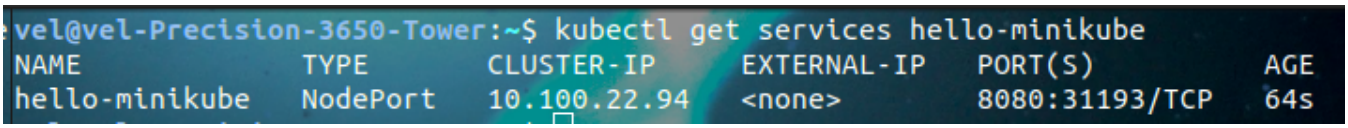
Tada! Your application is now available at http://localhost:7080/.

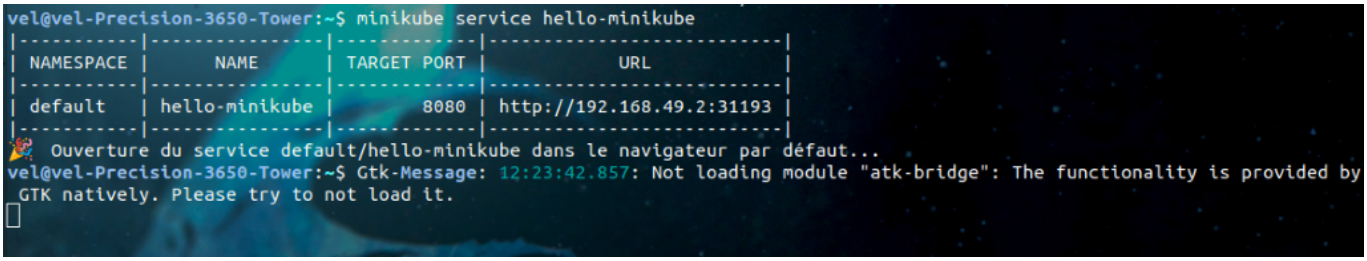
You should be able to see the request metadata in the application output. Try changing the path of the request and observe the changes. Similarly, you can do a POST request and observe the body show up in the output.

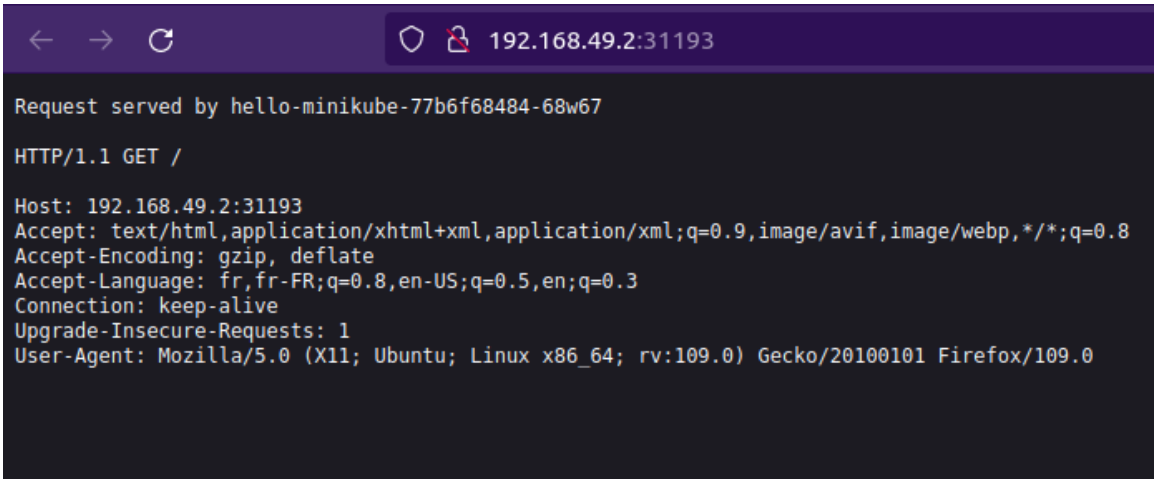
On déploie une application qu'on expose sur le port 8080

```
vel@vel-Precision-3650-Tower:~$ kubectl create deployment hello-minikube --image=kicbase/echo-server:1.0
deployment.apps/hello-minikube created
vel@vel-Precision-3650-Tower:~$ kubectl expose deployment hello-minikube --type=NodePort --port=8080
service/hello-minikube exposed
```

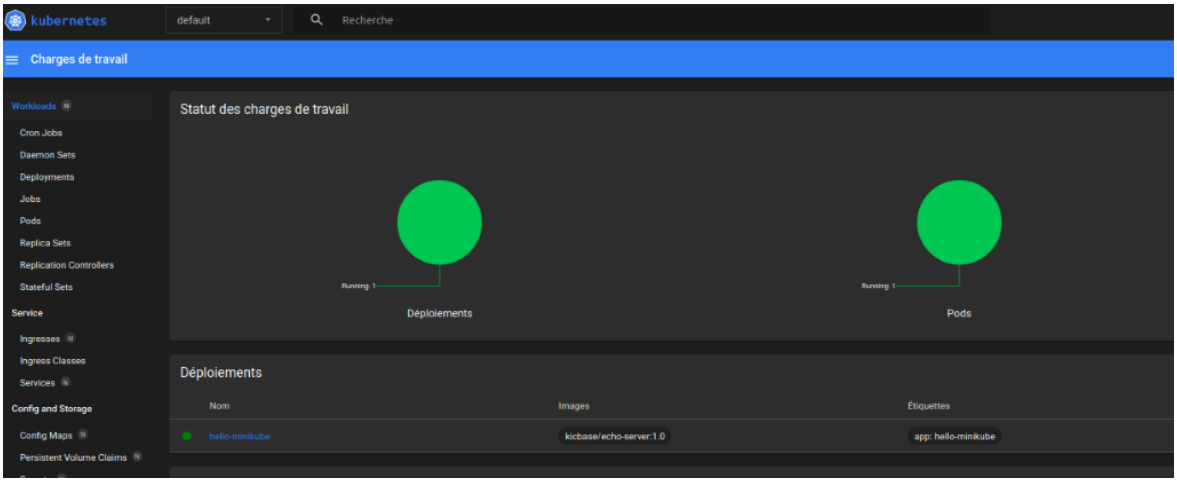
Vérification des services lancés



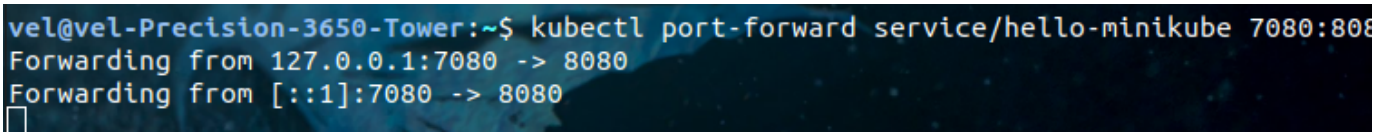


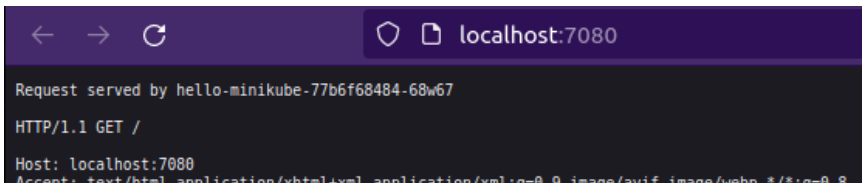


On constate également le rajout du pods sur la page de management :



On utilise maintenant kubectl pour ouvrir les ports





```
Accept-Encoding: gzip, deflate, br
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Connection: keep-alive
Cookie: jenkins-timestamper-offset=-3600000
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
Sec-Fetch-User: ?1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/109.0
```

Service | **LoadBalancer** | Ingress

To access a LoadBalancer deployment, use the "minikube tunnel" command. Here is an example deployment:

```
kubectl create deployment balanced --image=kicbase/echo-server:1.0
kubectl expose deployment balanced --type=LoadBalancer --port=8080
```

In another window, start the tunnel to create a routable IP for the 'balanced' deployment:

```
minikube tunnel
```

To find the routable IP, run this command and examine the `EXTERNAL-IP` column:

```
kubectl get services balanced
```

Your deployment is now available at <EXTERNAL-IP>:8080

```
vel@vel-Precision-3650-Tower:~$ kubectl create deployment balanced --image=kicbas
e/echo-server:1.0
deployment.apps/balanced created
vel@vel-Precision-3650-Tower:~$ kubectl expose deployment balanced --type=LoadBal
ancer --port=8080
service/balanced exposed
```

```
vel@vel-Precision-3650-Tower:~$ minikube tunnel
[sudo] Mot de passe de vel :
Status:
        machine: minikube
        pid: 67575
        route: 10.96.0.0/12 -> 192.168.49.2
        minikube: Running
        services: [balanced]
    errors:
                minikube: no errors
                router: no errors
                loadbalancer emulator: no errors
```

Déploiements

Nom	Images	Étiquettes
balanced	kicbase/echo-server:1.0	app: balanced
hello-minikube	kicbase/echo-server:1.0	app: hello-minikube

Pods

Nom	Images	Étiquettes	Noeud	Statut
balanced-56c586bfb9-d7bt4	kicbase/echo-server:1.0	app: balanced pod-template-hash: 56c586bfb9	minikube	Running
hello-minikube-77b6f68484-68w67	kicbase/echo-server:1.0	app: hello-minikube pod-template-hash: 77b6f68484	minikube	Running

```
vel@vel-Precision-3650-Tower:~$ kubectl get services balanced
NAME       TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
balanced   LoadBalancer   10.106.53.141   10.106.53.141   8080:30182/TCP   117s
```

Addon ingress :

Service | LoadBalancer | Ingress

Enable ingress addon:

```
minikube addons enable ingress
```

Copy

The following example creates simple echo-server services and an Ingress object to route to these services.

```
vel@vel-Precision-3650-Tower:~$ minikube addons enable ingress
💡  ingress est un addon maintenu par Kubernetes. Pour toute question, contactez minikube sur GitHub.
Vous pouvez consulter la liste des mainteneurs de minikube sur : https://github.com/kubernetes/minikube/blob/master/OWNERS
    ▪ Utilisation de l'image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20220916-gd32f8c343
    ▪ Utilisation de l'image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20220916-gd32f8c343
    ▪ Utilisation de l'image registry.k8s.io/ingress-nginx/controller:v1.5.1
🔎  Vérification du module ingress...
🌟  Le module 'ingress' est activé
vel@vel-Precision-3650-Tower:~$
```

Apply the contents

```
kubectl apply -f https://storage.googleapis.com/minikube-site-examples/ingress-example.yaml
```

Copy

Wait for ingress address

Ingress

```
kubectl get ingress
NAME              CLASS   HOSTS   ADDRESS          PORTS   AGE
example-ingress   nginx   *       <your_ip_here>   80      5m45s
```

Note for Docker Desktop Users:
To get ingress to work you'll need to open a new terminal window and run `minikube tunnel` and in the following step use `127.0.0.1` in place of `<ip_from_above>`.

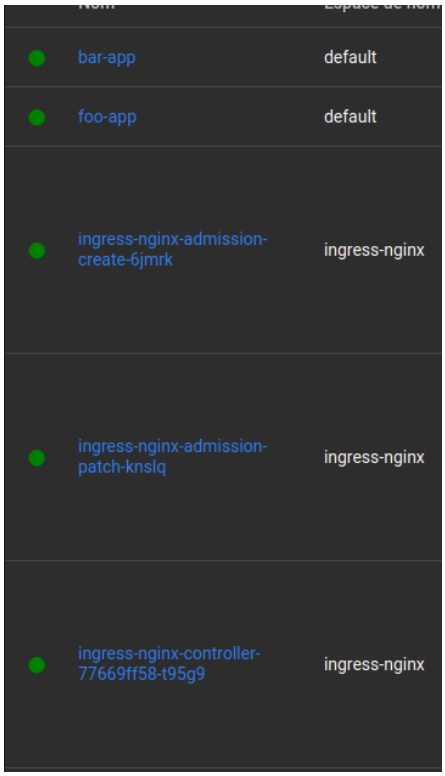
Now verify that the ingress works

```
$ curl <ip_from_above>/foo
Request served by foo-app
...

$ curl <ip_from_above>/bar
Request served by bar-app
...
```

```
vel@vel-Precision-3650-Tower:~$ minikube addons enable ingress
💡  ingress est un addon maintenu par Kubernetes. Pour toute question, contactez minik
Vous pouvez consulter la liste des mainteneurs de minikube sur : https://github.com/ku
    ▪ Utilisation de l'image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20220
    ▪ Utilisation de l'image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20220
    ▪ Utilisation de l'image registry.k8s.io/ingress-nginx/controller:v1.5.1
🔎  Vérification du module ingress...
🌟  Le module 'ingress' est activé
vel@vel-Precision-3650-Tower:~$ kubectl apply -f https://storage.googleapis.com/miniku
pod/foo-app created
service/foo-service created
pod/bar-app created
service/bar-service created
ingress.networking.k8s.io/example-ingress created
vel@vel-Precision-3650-Tower:~$ kubectl get ingress
NAME              CLASS   HOSTS   ADDRESS        PORTS   AGE
example-ingress   nginx   *       192.168.49.2   80      85s
```

```
vel@vel-Precision-3650-Tower:~$ curl 192.168.49.2/foo
Request served by foo-app

HTTP/1.1 GET /foo

Host: 192.168.49.2
Accept: */*
User-Agent: curl/7.81.0
X-Forwarded-For: 192.168.49.1
X-Forwarded-Host: 192.168.49.2
X-Forwarded-Port: 80
X-Forwarded-Proto: http
X-Forwarded-Scheme: http
X-Real-Ip: 192.168.49.1
X-Request-Id: 170a97423fb3cc90e21018ae5422e247
X-Scheme: http
vel@vel-Precision-3650-Tower:~$ curl 192.168.49.2/bar
```

Suite à ces manipulations, on voit que nous avons 5 nouveaux pods :



Suite du TP : tenter de lancer une image nginx avec minikube :

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Minicube sert uniquement à créer des VM en local, il ne nous sera pas utilise pour ce TP

Pour faire ce TP, il faudrait utiliser KubeADM :

https://kubernetes.io/fr/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/

Installation KubeADM

Vérifiez les ports requis

Vérifiez les ports requis

nœuds maîtres (masters)

Protocole	Direction	Plage de Port	Utilisé pour	Utilisé par
TCP	Entrant	6443*	Kubernetes API server	Tous
TCP	Entrant	2379-2380	Etcd server client API	kube-apiserver, etcd
TCP	Entrant	10250	Kubelet API	Lui-même, Control plane
TCP	Entrant	10251	kube-scheduler	Lui-même
TCP	Entrant	10252	kube-controller-manager	Lui-même

nœuds workers

Protocole	Direction	Plage de Port	Utilisé pour	Utilisé par
TCP	Entrant	10250	Kubelet API	Lui-même, Control plane
TCP	Entrant	30000-32767	NodePort Services**	Eux-mêmes

** Plage de ports par défaut pour les Services NodePort.

Tous les numéros de port marqués d'un * sont écrasables. Vous devrez donc vous assurer que les ports personnalisés que vous utilisez sont également ouverts.

Bien que les ports etcd soient inclus dans les nœuds masters, vous pouvez également héberger votre propre cluster etcd en externe ou sur des ports personnalisés.

Le plug-in de réseau de pod que vous utilisez (voir ci-dessous) peut également nécessiter certains ports à ouvrir. Étant donné que cela diffère d'un plugin à l'autre, veuillez vous reporter à la documentation des plugins sur le(s) port(s) requis(s).

Par défaut, Kubernetes utilise le Container Runtime Interface (CRI) pour s'interfacer avec votre environnement d'exécution de conteneur choisi.

Si vous ne spécifiez pas de runtime, kubeadm essaie automatiquement de détecter un Runtime de conteneur en parcourant une liste de sockets de domaine Unix bien connus. Le tableau suivant répertorie les environnements d'exécution des conteneurs et leurs chemins de socket associés:

Runtime	Chemin vers le socket de domaine Unix
Docker	`/var/run/docker.sock`
containerd	`/run/containerd/containerd.sock`
CRI-O	`/var/run/crio/crio.sock`

Si Docker et containerd sont détectés, Docker est prioritaire. C'est nécessaire car Docker 18.09 est livré avec containerd et les deux sont détectables même si vous installez Docker. Si deux autres environnements d'exécution ou plus sont détectés, kubeadm se ferme avec une erreur.
Le kubelet s'intègre à Docker via l'implémentation CRI intégrée de `dockershim`.

Voir runtimes de conteneur pour plus d'informations.

Installation de kubeadm, des kubelets et de kubectl

Vous installerez ces paquets sur toutes vos machines:

- `kubeadm` : la commande pour initialiser le cluster.
- la `kubelet` : le composant qui s'exécute sur toutes les machines de votre cluster et fait des actions comme le démarrage des pods et des conteneurs.
- `kubectl` : la ligne de commande utilisée pour parler à votre cluster.

kubeadm **n'installera pas** ni ne gèrera les `kubelet` ou `kubectl` pour vous. Vous devez vous assurer qu'ils correspondent à la version du control plane de Kubernetes que vous souhaitez que kubeadm installe pour vous. Si vous ne le faites pas, vous risquez qu'une erreur de version se produise, qui pourrait conduire à un comportement inattendu. Cependant, une version mineure entre les kubelets et le control plane est pris en charge, mais la version de la kubelet ne doit jamais dépasser la version de l'API server. Par exemple, les kubelets exécutant la version 1.7.0 devraient être entièrement compatibles avec un API server en 1.8.0, mais pas l'inverse.

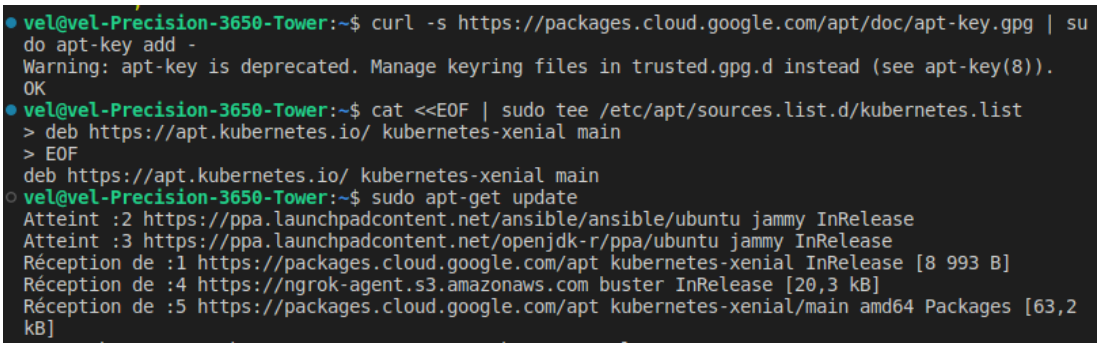
For information about installing `kubectl`, see Installation et configuration kubectl.

> **Attention:** Ces instructions excluent tous les packages Kubernetes de toutes les mises à niveau du système d'exploitation. C'est parce que kubeadm et Kubernetes ont besoin d'une attention particulière lors de la mise à niveau.

Ubuntu, Debian or HypriotOS | CentOS, RHEL or Fedora | Fedora CoreOS ou Flatcar Container Linux

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

Kubelet redémarre maintenant toutes les quelques secondes, car il attend les instructions de kubeadm dans une boucle de crash.



Installation KubeADM sur Arch :

https://wiki.archlinux.org/title/Kubernetes

1 Installation

When manually creating a Kubernetes cluster **install** `etcd`[AUR] and the **package group** `kubernetes-control-plane` (for a control-plane node) and `kubernetes-node` (for a worker node).

When creating a Kubernetes cluster with the help of `kubeadm`, **install** `kubeadm` and `kubelet` on each node.

Both control-plane and regular worker nodes require a container runtime for their `kubelet` instances which is used for hosting containers. **Install** either `containerd` or `cri-o` to meet this dependency.

To control a kubernetes cluster, install `kubectl` on the control-plane hosts and any external host that is supposed to be able to interact with the cluster.

`helm` is a tool for managing pre-configured Kubernetes resources which may be helpful for getting started.

Objectifs

- Installer un cluster Kubernetes à master unique ou un cluster à haute disponibilité
- Installez un réseau de pods sur le cluster afin que vos pods puissent se parler

Initialiser votre master

Le master est la machine sur laquelle s'exécutent les composants du control plane, y compris etcd (la base de données du cluster) et l'API serveur (avec lequel la CLI kubectl communique).

1. Choisissez un add-on réseau pour les pods et vérifiez s'il nécessite des arguments à passer à l'initialisation de kubeadm. Selon le fournisseur tiers que vous choisissez, vous devrez peut-être définir le `--pod-network-cidr` sur une valeur spécifique au fournisseur. Voir Installation d'un add-on réseau de pod.
2. (Facultatif) Sauf indication contraire, kubeadm utilise l'interface réseau associée avec la passerelle par défaut pour annoncer l'IP du master. Pour utiliser une autre interface réseau, spécifiez l'option `--apiserver-advertise-address=<ip-address>` à `kubeadm init`. Pour déployer un cluster Kubernetes en utilisant l'adressage IPv6, vous devez spécifier une adresse IPv6, par exemple `--apiserver-advertise-address=fd00::101`
3. (Optional) Lancez `kubeadm config images pull` avant de faire `kubeadm init` pour vérifier la connectivité aux registres gcr.io.

Deuxième partie du cours : Kind et TP

Bascule sur la suite du cours : Rappel de la philosophie DevOps et Kubernetes

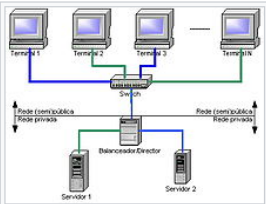
Ne pas oublier que Kubernetes est extrêmement intéressant pour gérer le Load Balancing

Pour les articles homonymes, voir Load et balance.

En informatique, la **répartition de charge** (en anglais : *load balancing*) désigne le processus de répartition d'un ensemble de tâches sur un ensemble de ressources, dans le but d'en rendre le traitement global plus efficace. Les techniques de répartition de charge permettent à la fois d'optimiser le temps de réponse pour chaque tâche, tout en évitant de surcharger de manière inégale les nœuds de calcul.

La *répartition de charge* est issue de la recherche dans le domaine des ordinateurs parallèles. Deux principales approches coexistent : les algorithmes statiques, qui ne tiennent pas compte de l'état des différentes machines, et les algorithmes dynamiques, qui sont en général plus généraux et performants, mais nécessitent des échanges d'information entre les différentes unités de calculs, au risque d'une perte d'efficacité.

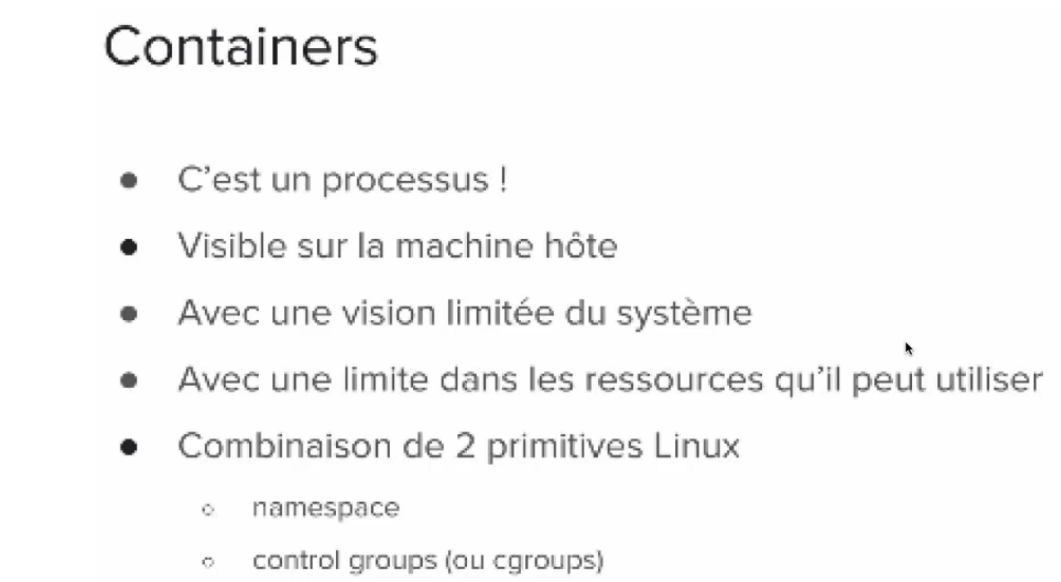
Ces techniques sont par exemple très utilisées dans le domaine des services HTTP où un site à forte audience doit pouvoir gérer des centaines de milliers de requêtes par seconde.

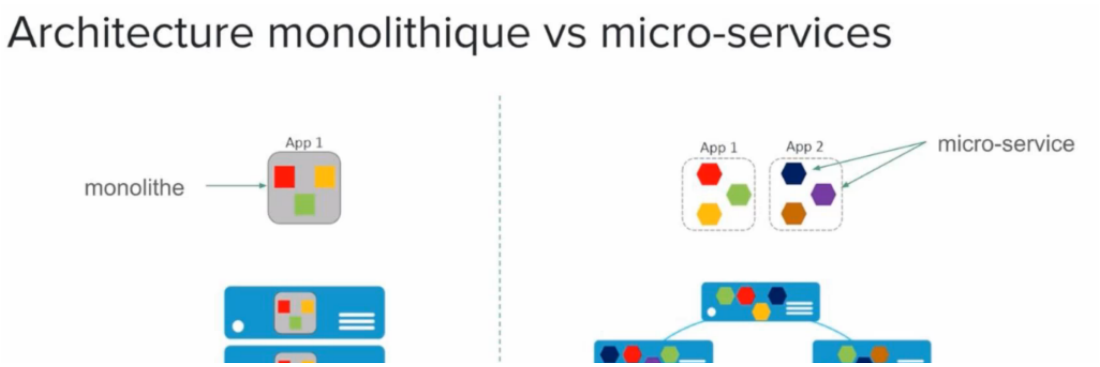


Répartition de charge entre deux serveurs accédés par 4 postes clients, ici le switch et le répartiteur sont deux points de panne potentiels, aucun des deux n'est doublé.

Si jamais une monté en charge est sur le point d'arriver, Kubernetes va nous permettre de déployer très rapidement un conteneur similaire à celui de notre app pour réduire sa montée en charge.

Rappel containers :





Architecture micro-services

- Découpage de l'application en multiples services
- Processus indépendant ayant sa propre responsabilité métier
- Plus grande liberté de choix dans le langage
- Équipe dédiée pour chaque service
- Un service peut être mis à jour indépendamment des autres services
- Containers très adaptés pour les micro-services
- Nécessite des interfaces bien définies
- Déplace la complexité dans l'orchestration de l'application globale

Si un micro-service tombe sur un infra monolithique : l'application complète tombe

Avec Kubernetes, si un pod tombe, il est directement remplacé pour garantir la tolérance de panne

La **tolérance aux pannes** (ou « insensibilité aux pannes ») désigne une méthode de conception permettant à un système de continuer à fonctionner, éventuellement de manière réduite (on dit aussi en « mode dégradé »), au lieu de tomber complètement en panne, lorsque l'un de ses composants ne fonctionne plus correctement.

L'expression est employée couramment pour les systèmes informatiques étudiés de façon à rester plus ou moins opérationnels en cas de panne partielle, c'est-à-dire éventuellement avec une réduction du débit ou une augmentation du temps de réponse. En d'autres termes, le système ne s'arrête pas de fonctionner, qu'il y ait défaillance matérielle ou défaillance logicielle.

Un exemple en dehors de l'informatique est celui du véhicule à moteur conçu pour être toujours en état de rouler même si l'un de ses pneus est crevé.

Application Cloud Native

- Application orientée microservices
- Packagée dans des containers
- Orchestration dynamique
- Nombreux projets portés par la CNCF (Cloud Native Computing Foundation)
  - Kubernetes
  - Prometheus
  - Fluentd
  - ...
- cncf.io

CNCF :

https://www.cncf.io/

https://fr.wikipedia.org/wiki/Cloud_Native_Computing_Foundation

La **Cloud Native Computing Foundation** (**CNCF**) est un projet de la Linux Foundation qui a été fondé en 2015 pour aider à faire progresser la technologie des conteneurs[1] et rassembler les industries technologiques autour de son évolution.

Il a été annoncé en même temps Kubernetes 1.0, un gestionnaire de cluster de conteneurs open source, qui a été contribué à la Linux Foundation par Google en tant qu'initiateur de cette technologie. Les membres fondateurs incluent Google, CoreOS, Mesosphere, Red Hat, Twitter, Huawei, Intel, Cisco, IBM, Docker, Univa et VMware[2,3]. Aujourd'hui, la CNCF est soutenue par plus de 450 membres. Afin d'établir des représentants qualifiés des technologies régies par la CNCF, un programme a été annoncé lors du premier CloudNativeDay[4] à Toronto en août 2016[5].

Cloud Native Computing Foundation (CNCF)

CLOUD NATIVE COMPUTING FOUNDATION

Construire des écosystèmes durables pour les logiciels Cloud Native

Situation	
Création	2015
Type	501(c)(6) organization

Outils de monitoring évoqué pendant le cours : Prometheus

https://fr.wikipedia.org/wiki/Prometheus_(logiciel)

Prometheus est un logiciel libre de surveillance informatique et générateur d'alertes. Il enregistre des métriques en temps réel dans une base de données de séries temporelles (avec une capacité d'acquisition élevée) en se basant sur le contenu de point d'entrée exposé à l'aide du protocole HTTP. Ces métriques peuvent ensuite être interrogées à l'aide d'un langage de requête simple (PromQL) et peuvent également servir à générer des alertes. Le projet est écrit en Go et est disponible sous licence Apache 2. Le code source est disponible sur GitHub[2], et est un projet maintenu par la Cloud Native Computing Foundation à côté d'autres projets comme Kubernetes et Envoy[3].

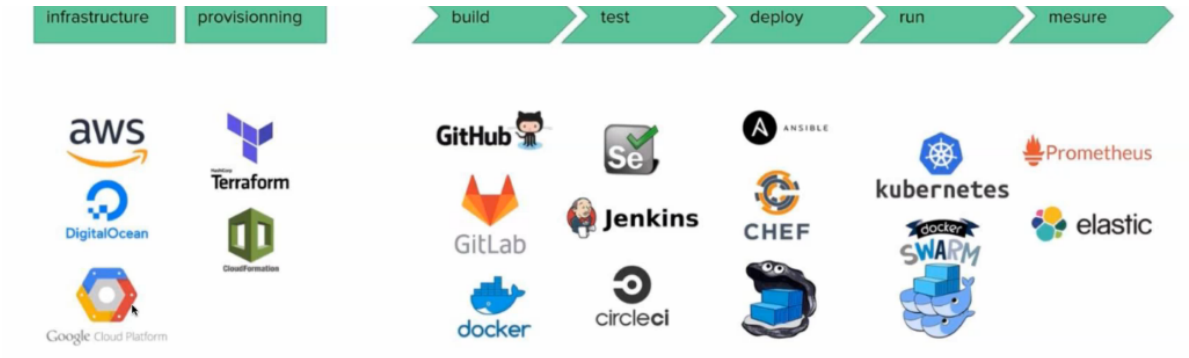
Devops :

DevOps

- Un objectif : minimiser le temps de livraison d'une fonctionnalité
- Déploiements réguliers
- Mise en avant des tests
- Automatisation des processus
  - provisioning / configuration
  - Infrastructure As Code (IaC)
  - Intégration Continue / Déploiement Continu (CI/CD)
  - monitoring
- Une boucle d'amélioration courte

Ne pas oublier que le DevOPs est une philosophie qui cherche à faire en sorte d'aider les développeurs et les opérationnels à travailler ensemble

DevOps : quelques outils et produits



Historique Kubernetes :

Historique

- Kubernetes / k8s / kube
- “Homme de barre” / “Pilote” en grec
- Plateforme open source d’orchestration de containers
- Inspirée du système Borg de Google
- v1.0, juillet 2015
- v1.19.0, aout 2020

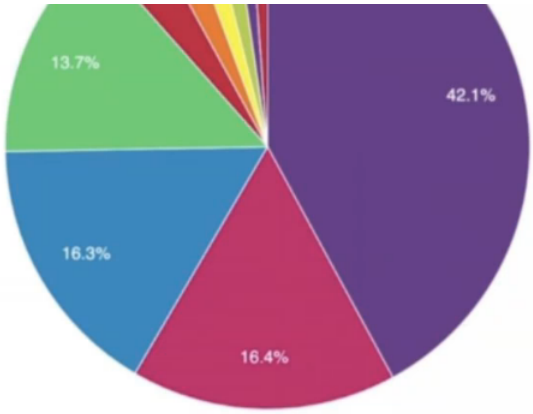
Fonctionnalités

- Gestion d’applications tournant dans des containers
  - déploiement
  - scaling
  - self-healing
- Boucles de réconciliation vers l’état souhaité (contrôleurs)
- Applications stateless et stateful
- Secrets et des Configurations
- Long-running process et batch jobs
- RBAC

Les principaux contributeurs (janvier 2020)

#	Company	Commits
1	Google	21202

	*independent	8242
2	Red Hat	8212
3	Huawei	1788
4	ZTE Corporation	1181
5	Microsoft	860
6	VMware	815
7	FathomDB	587
8	IBM	585
9	CoreOS	505



Cluster

- Ensemble de **nodes** Linux ou Windows (VM / bare metal)
- Nodes **Masters** + nodes **Workers**
- Un Master expose l'**API Server** - point d'entrée pour la gestion du cluster

Un nodes master va gérer le cluster, il contient l'API server : https://kubernetes.io/docs/reference/command-line-tools-reference/kube-apiserver/

Synopsis

The Kubernetes API server validates and configures data for the api objects which include pods, services, replicationcontrollers, and others. The API Server services REST operations and provides the frontend to the cluster's shared state through which all other components interact.

```
kube-apiserver [flags]
```

Un master peut aussi contenir des pods, mais cette pratique n'est pas conseillé

ReplicaSet : https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

ReplicaSet

A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.

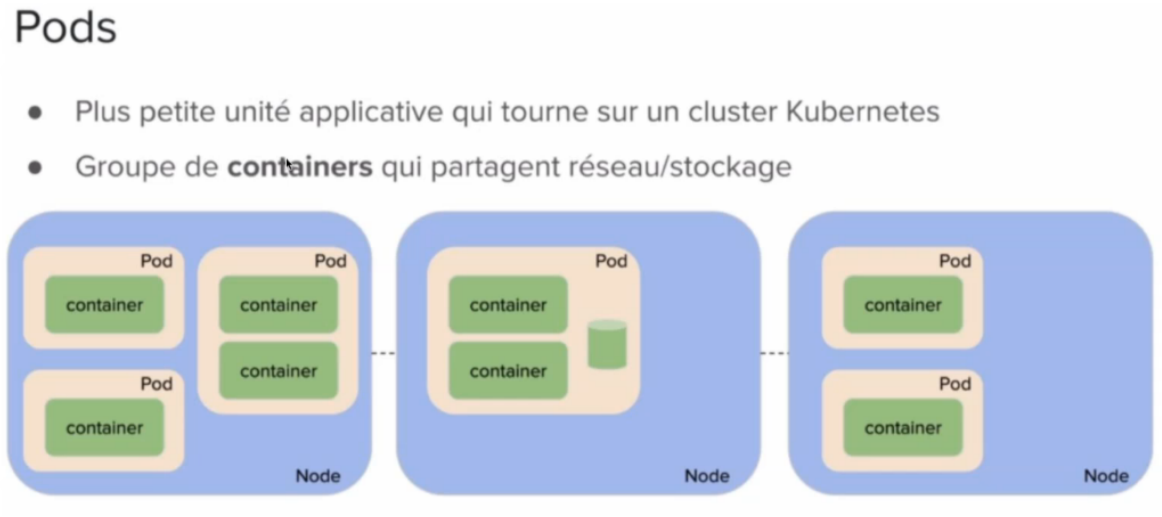
How a ReplicaSet works

A ReplicaSet is defined with fields, including a selector that specifies how to identify Pods it can acquire, a number of replicas indicating how many Pods it should be maintaining, and a pod template specifying the data of new Pods it should create to meet the number of replicas criteria. A ReplicaSet then fulfills its purpose by creating and deleting Pods as needed to reach the desired number. When a ReplicaSet needs to create new Pods, it uses its Pod template.

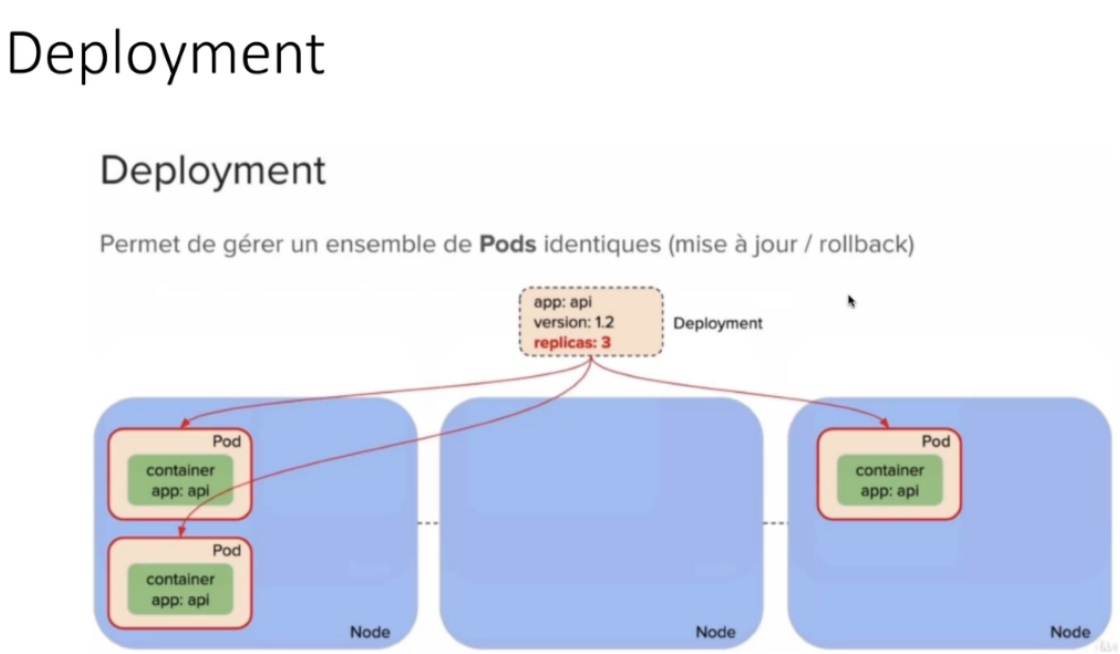
A ReplicaSet is linked to its Pods via the Pods' metadata.ownerReferences field, which specifies what resource the current object is owned by. All Pods acquired by a ReplicaSet have their owning ReplicaSet's identifying information within their ownerReferences field. It's through this link that the ReplicaSet knows of the state of the Pods it is maintaining and plans

accordingly.

A ReplicaSet identifies new Pods to acquire by using its selector. If there is a Pod that has no OwnerReference or the OwnerReference is not a Controller and it matches a ReplicaSet's selector, it will be immediately acquired by said ReplicaSet.



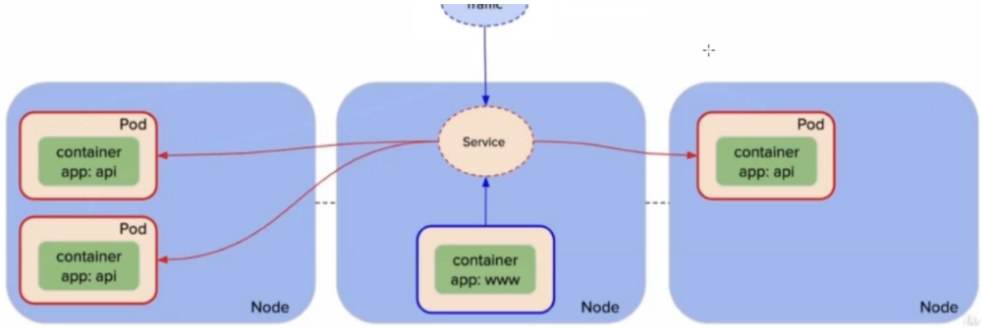
Les services vont pouvoir faire communiquer les pods entre eux



Le Namespaces est importer pour isoler les groupes de ressources dans un cluster : https://kubernetes.io/docs/concepts/overview/working-with-objects/names

Namespaces

In Kubernetes, *namespaces* provides a mechanism for isolating groups of resources within a single cluster. Names of resources need to be unique within a namespace, but not across namespaces. Namespace-based scoping is applicable only for namespaced objects *(e.g. Deployments, Services, etc)* and not for cluster-wide objects *(e.g. StorageClass, Nodes, PersistentVolumes, etc)*.



Le pod www a besoin d'accéder aux autres pods API du ReplicaSet : il va accéder au service qui va le rediriger vers les pods API disponible sur les autres poc
ReplicaSet

https://kubernetes.io/docs/concepts/services-networking/ingress/

Ingress

FEATURE STATE: `Kubernetes v1.19 [stable]`

An API object that manages external access to the services in a cluster, typically HTTP.

Ingress may provide load balancing, SSL termination and name-based virtual hosting.

Terminology

For clarity, this guide defines the following terms:

- Node: A worker machine in Kubernetes, part of a cluster.
- Cluster: A set of Nodes that run containerized applications managed by Kubernetes. For this example, and in most common Kubernetes deployments, nodes in the cluster are not part of the public internet.
- Edge router: A router that enforces the firewall policy for your cluster. This could be a gateway managed by a cloud provider or a physical piece of hardware.
- Cluster network: A set of links, logical or physical, that facilitate communication within a cluster according to the Kubernetes networking model.
- Service: A Kubernetes Service that identifies a set of Pods using label selectors. Unless mentioned otherwise, Services are assumed to have virtual IPs only routable within the cluster network.

https://ibm.github.io/kubernetes-networking/route/route/

Spécification des ressources

- Généralement en **yaml**
- Une structure commune

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: www
  labels:
    app: w3
spec:
  containers:
  - name: www
    image: nginx:1.16
```

Exemple de spécification d'un Pod

```yaml
apiVersion: v1
kind: Service
metadata:
  name: www
spec:
  selector:
    app: w3
  ports:
  - port: 80
    targetPort: 80
```

Exemple de spécification d'un Service

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: www
spec:
  replicas: 3
  selector:
    matchLabels:
      app: w3
  template:
    metadata:
      labels:
        app: w3
    spec:
      containers:
      - name: www
        image: nginx:1.16
```

Exemple de spécification d'un Deployment

Metadata : sert à rattacher les pods à leur service (exemple : **labels : app : w3** permet de classer le pods avec les autres pods qui ont le même labels)

Pour le service : **spec: selector: app: w3** permet à chaque pods faisant partie du label w3 de bénéficier de ce service

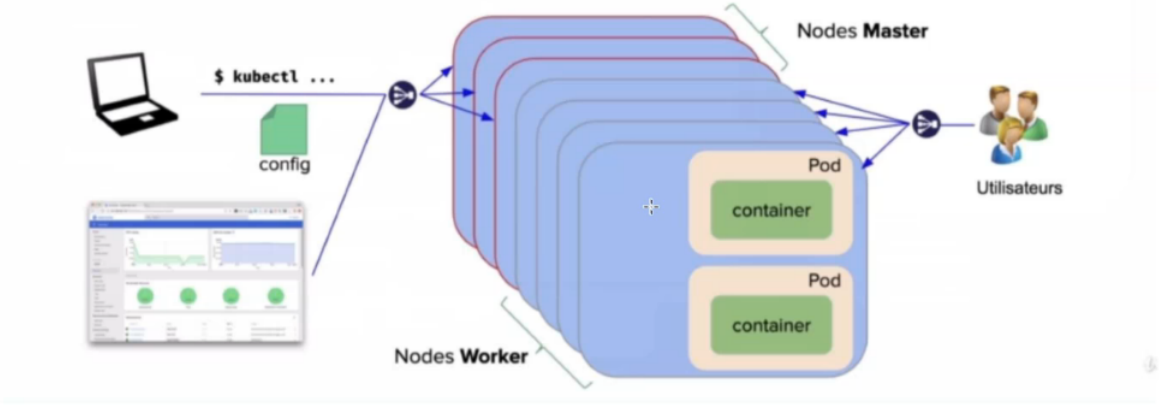
Pour le Deployment : **spec: replicas: 3** pour 3 ReplicaSet | **selector: matchLabels: app: w3** pour sélectionner le label w3 à déployer

Attention : on ne doit pas créer un pods et un Deployement en même temps : on a plus de marge de manœuvre pour gérer notre cluster avec un Deployment

Gestion du cluster

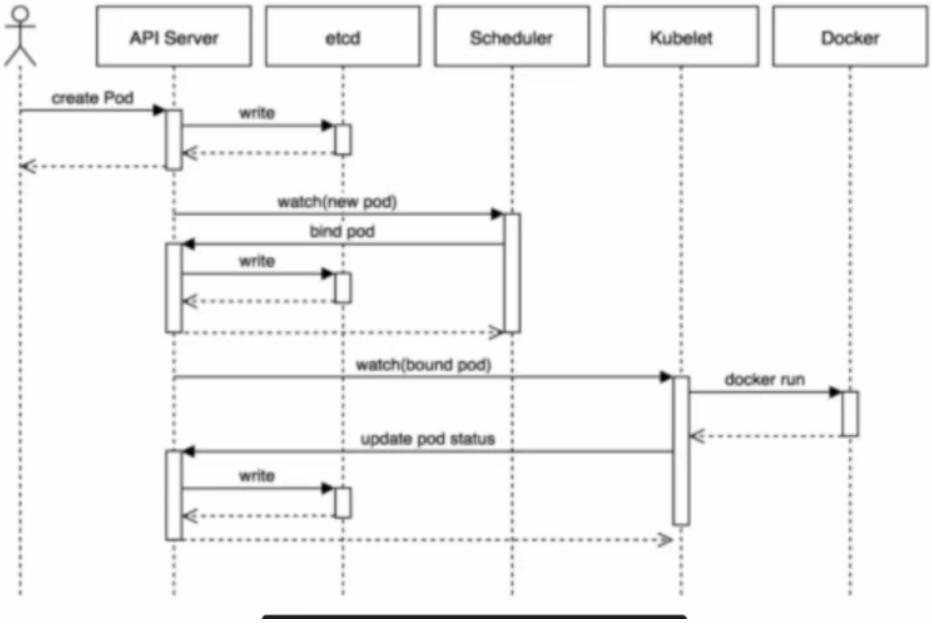
- Envoi de requêtes HTTPS à l'**API Server**

- Envoi de requêtes HTTPS à l'**API Server**
- Utilisation du binaire **kubectl** ou de l'une des nombreuses interfaces (web / CLI)



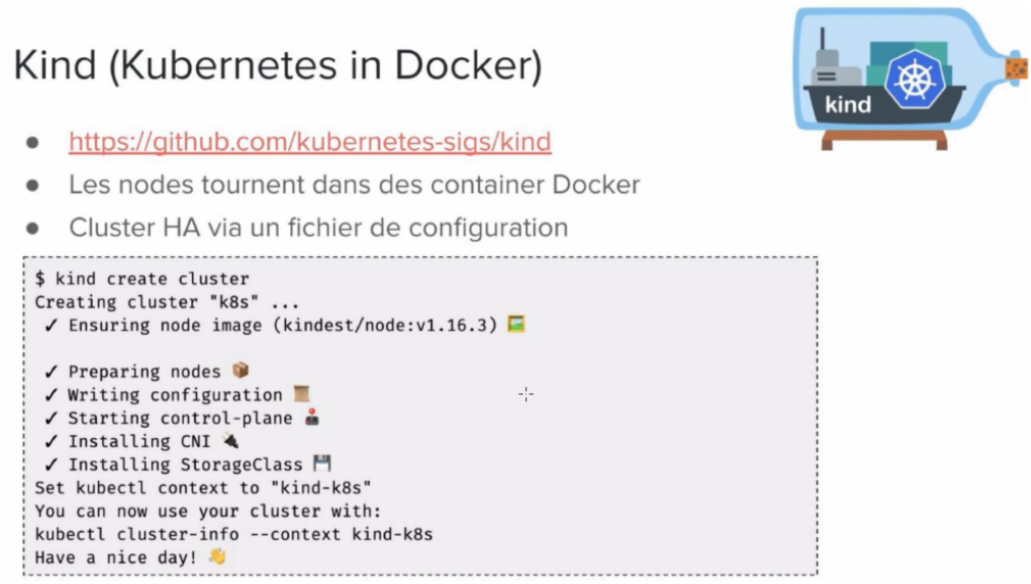
Kubectl -> communication avec le master

Les processus : workflow de création d'un Pod



Minikube

- Tous les composants de Kubernetes dans une seule VM locale
- S'intègre avec différents hyperviseurs
  - HyperKit
  - Hyper-V
  - KVM
  - VirtualBox
  - VMWare
- Nécessite le binaire minikube
  - https://github.com/kubernetes/minikube/releases



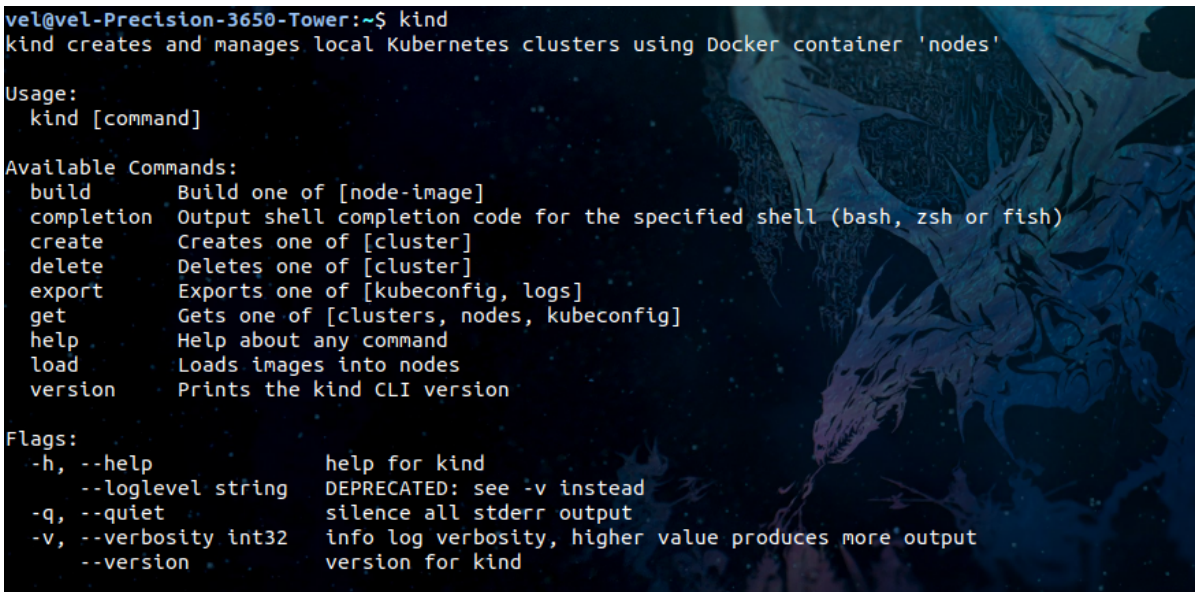
Installation KInd :

https://kind.sigs.k8s.io/docs/user/quick-start/

On Linux:

```
curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.17.0/kind-linux-amd64
chmod +x ./kind
sudo mv ./kind /usr/local/bin/kind
```

Commande Kind pour vérifier que le logiciel est bien installé :



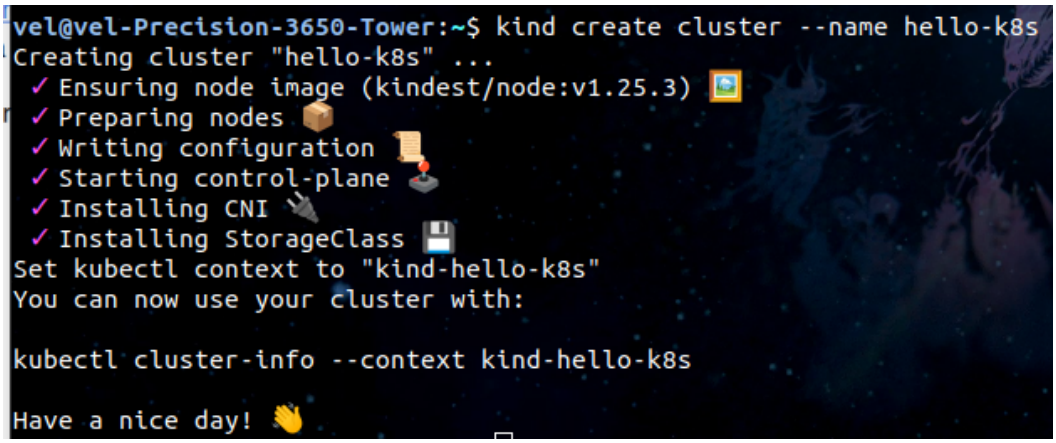
Création cluster avec kind : kind create cluster **--argument**

Creating a Cluster

Creating a Kubernetes cluster is as simple as `kind create cluster`.

This will bootstrap a Kubernetes cluster using a pre-built node image. Prebuilt images are hosted at `kindest/node`, but to find images suitable for a given release currently you should check the release notes for your given kind version (check with `kind version`) where you'll find a complete listing of images created for a kind release.

To specify another image use the `--image` flag – `kind create cluster --image=...`.

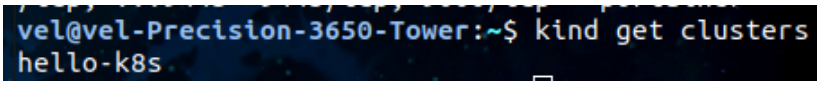


When you list your kind clusters, you will see something like the following:

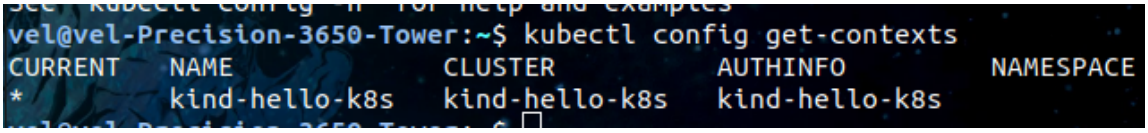
```
kind get clusters
kind
kind-2
```

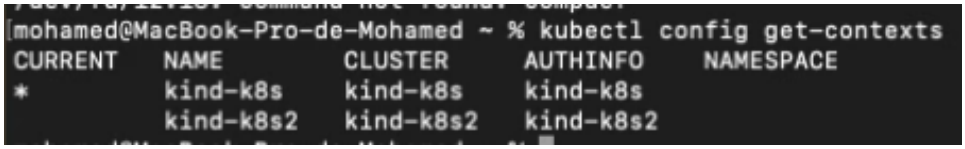
In order to interact with a specific cluster, you only need to specify the cluster name as a context in kubectl:

```
kubectl cluster-info --context kind-kind
kubectl cluster-info --context kind-kind-2
```

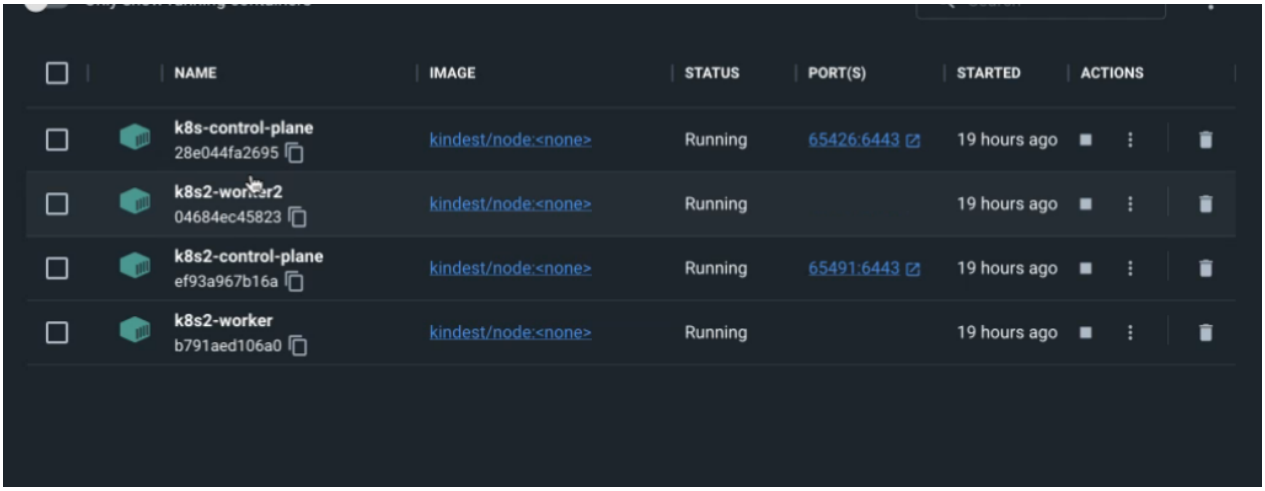


Pour vérifier que le cluster existe : **kubectl config get-contexts**





Pour voir la liste des noeuds disponible sur notre machine : **docker ps** pour voir les containers utilisé par **Kind**




```
vel@vel-Precision-3650-Tower:~$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED
     STATUS          PORTS
                                NAMES
45880b8c39dd   kindest/node:v1.25.3   "/usr/local/bin/entr…"   5 minutes a
go   Up 5 minutes    127.0.0.1:38557->6443/tcp
                                hello-k8s-control-plane
```

Pour sélectionner un cluster :

```
mohamed@MacBook-Pro-de-Mohamed ~ % kubectl config use-context kind-k8s2
Switched to context "kind-k8s2".
```

```
vel@vel-Precision-3650-Tower:~$ kubectl config use-context kind-hello-k8s
Switched to context "kind-hello-k8s".
```

Pour avoir la liste des nodes d'un cluster quand on a basculé dans son context :

```
vel@vel-Precision-3650-Tower:~$ kubectl get no
NAME                     STATUS   ROLES           AGE   VERSION
hello-k8s-control-plane   Ready    control-plane   20m   v1.25.3
```

Présentation

- Groupe de containers tournant dans un même context d'isolation
  - Linux namespaces : network, IPC, UTS, ...
- Partagent la stack réseau et le stockage (volumes)
- Adresse IP dédiée, pas de NAT pour la communication entre les Pods

Source: kubernetes.io

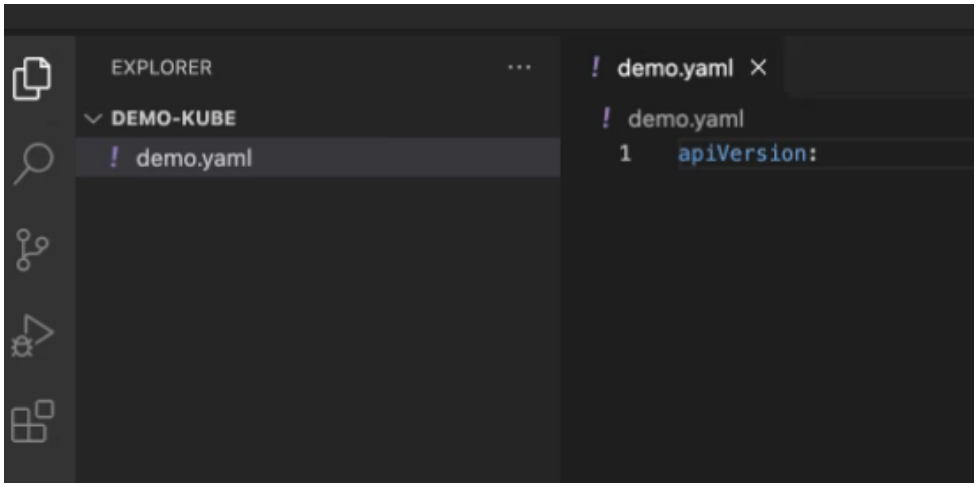
Exemple - server http

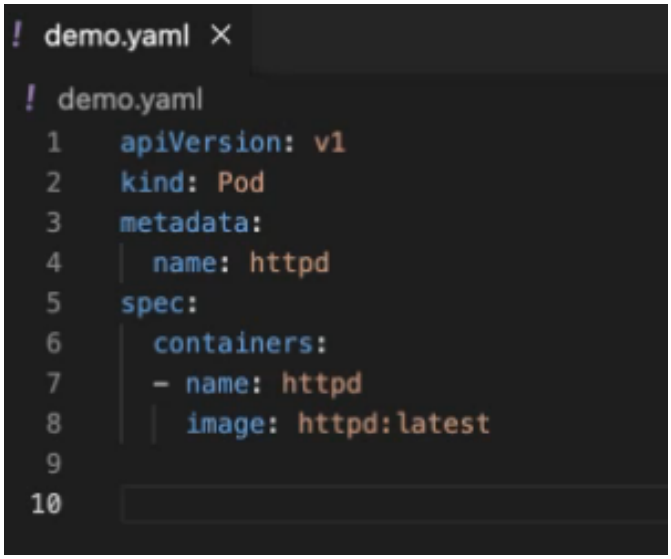
- Spécification dans un fichier texte yaml (souvent préféré au format json)

```
$ cat nginx-pod.yaml
apiVersion: v1            L'objet Pod est stable et disponible depuis la v1 de l'API
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: www
    image: nginx:1.12.2
```

Spécification d'un Pod dans lequel est lancé un container basé sur l'image nginx

Exemple - création fichier yaml pour déployer un nodes-worker de test






```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl apply -f demo.yaml
pod/httpd created
```

```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
httpd   1/1     Running   0          2m8s
```

Pour vérifier la description du pods : **kubectl describe "typde_ressource"/"nom_ressource"**
(il faut obligatoirement indiquer le type de ressource, si on indique uniquement son nom, nous aurons une erreur)

```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl describe po/httpd
Name:             httpd
Namespace:        default
Priority:         0
Service Account:  default
Node:             hello-k8s-control-plane/172.18.0.2
```

```
Start Time:       Fri, 10 Feb 2023 15:12:08 +0100
Labels:           <none>
Annotations:      <none>
Status:           Running
IP:               10.244.0.5
IPs:
  IP:  10.244.0.5
Containers:
  httpd:
    Container ID:   containerd://db8cfd774136f16ab69619646e44fcdba2843695afa699c00762c1db18de0c9e
    Image:          httpd:latest
    Image ID:       docker.io/library/httpd@sha256:db2d897cae2ad67b33435c1a5b0d6b6465137661ea7c01a5e95155f0159e1bcf
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Fri, 10 Feb 2023 15:12:14 +0100
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-ltm8w (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
```

Tout en bas du describe se trouve les différentes **events** de la vie de la ressource:

```
                    node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  9m17s  default-scheduler  Successfully assigned default/httpd to hello-k8s-control-plane
  Normal  Pulling    9m17s  kubelet            Pulling image "httpd:latest"
  Normal  Pulled     9m11s  kubelet            Successfully pulled image "httpd:latest" in 5.974278097s
  Normal  Created    9m11s  kubelet            Created container httpd
  Normal  Started    9m10s  kubelet            Started container httpd
```

Pour rentrer dans le pod en mode interactif avec notre shell actuel : **kubectl exec –i –t "nom ressource" -- /bin/bash**

```
mohamed@MacBook-Pro-de-Mohamed demo-kube % kubectl exec -i -t httpd -- /bin/bash
root@httpd:/usr/local/apache2#
```

```
root@httpd:/usr/local/apache2# cd htdocs/
root@httpd:/usr/local/apache2/htdocs# ls
index.html
root@httpd:/usr/local/apache2/htdocs# echo "Test Julien a nouveau" >> index.html
root@httpd:/usr/local/apache2/htdocs# cat index.html
<html><body><h1>It works!</h1></body></html>
Test Julien a nouveau
```

(pour windows -> slmgr /dlv pour vérifier le nombre de réarmement dispo : slmgr /rearm pour remettre une version d'essai

```
PS C:\windows\system32> slmgr /dlv
PS C:\windows\system32> slmgr /rearm
PS C:\windows\system32>
```

Port-forwarding du pods : **kubectl port-forward "nom_pods" "port_machine":"port_pods"**

```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl port-forward httpd 8091:80
Forwarding from 127.0.0.1:8091 -> 80
Forwarding from [::1]:8091 -> 80
```

Test sur la page web :

```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl port-forward httpd 8091:80
Forwarding from 127.0.0.1:8091 -> 80
Forwarding from [::1]:8091 -> 80

Handling connection for 8091
```

Nouvel onglet × localhost:8091/ +

localhost:8091

It works!

Autre façon de lancer un pods : **kubectl run "nom_pods" --image="nom_image"**

```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl run demo --image=nginx
pod/demo created
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl get po
NAME    READY   STATUS    RESTARTS        AGE
demo    1/1     Running   0               8s
httpd   1/1     Running   1 (8m52s ago)   61m
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$
```

Autre façon de faire : **--dry-run**

Le dry run permet d'afficher la syntaxe d'une configuration qui pourrait être contenu dans un fichier yaml

```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl run demo2 --image=httpd --dry-run=client -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: demo2
  name: demo2
spec:
  containers:
  - image: httpd
    name: demo2
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

Tp : déploiement de deux pods avec volume partagé :

k8s > demo_kind > ! pod_multi.yaml > {} spec > [ ] containers >

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mc1
spec:
  volumes:
  - name: data
    emptyDir: {}
  containers:
  - name : cont1
    image: nginx
    volumeMounts:
    - name: data
      mountPath: /usr/share/nginx/html
  - name: cont2
    image: debian
    volumeMounts:
    - name: data
      mountPath: /html
    command: ["/bin/bash", "-c"]
    args:
      - while true; do
          date >> / /html/index.html
          sleep 1;
        done
```

```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl apply -f pod_multi.yaml
pod/mc1 created
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl get po
NAME    READY   STATUS    RESTARTS      AGE
demo    1/1     Running   0             31m
httpd   1/1     Running   1 (39m ago)   92m
mc1     2/2     Running   0             53s
```

```
                 node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  4m22s  default-scheduler  Successfully assigned default/mc1 to hello-k8s-control-plane
  Normal  Pulling    4m22s  kubelet            Pulling image "nginx"
  Normal  Pulled     4m21s  kubelet            Successfully pulled image "nginx" in 902.593083ms
  Normal  Created    4m21s  kubelet            Created container cont1
  Normal  Started    4m21s  kubelet            Started container cont1
  Normal  Pulling    4m21s  kubelet            Pulling image "debian"
  Normal  Pulled     4m17s  kubelet            Successfully pulled image "debian" in 3.947068359s
  Normal  Created    4m17s  kubelet            Created container cont2
  Normal  Started    4m17s  kubelet            Started container cont2
```

Supprimer les ressources utilisé par un script : **kubectl delete –f "nom_script"**

```
 2020  history
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl delete -f pod_multi.yaml
pod "mc1" deleted
```

Test de déploiement du container de test :

Script Kubernetes qui servira au déploiement :

pod_multi.yaml

```
k8s > demo_kind > ! pod_multi.yaml > {} spec > [ ] containers
```

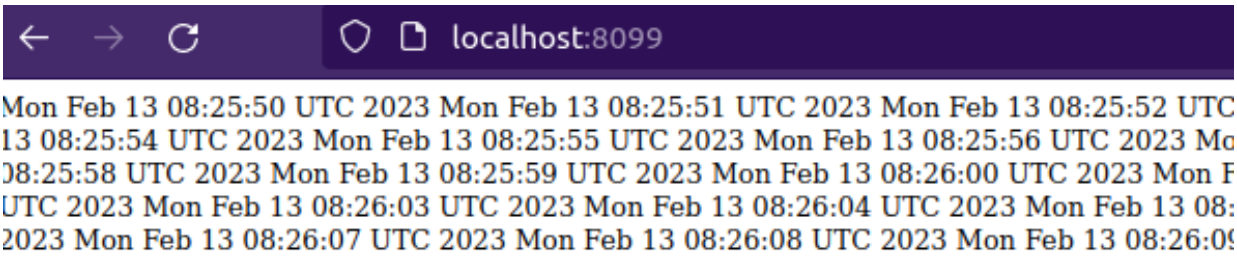
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mc1
spec:
  volumes:
  - name: data
    emptyDir: {}
  containers:
  - name : cont1
    image: nginx
    volumeMounts:
    - name: data
      mountPath: /usr/share/nginx/html
  - name: cont2
    image: debian
    volumeMounts:
    - name: data
      mountPath: /html
    command: ["/bin/bash", "-c"]
    args:
      - while true; do
          date >> /html/index.html;
          sleep 1;
        done
```

Apply du script pod_multi.yaml + port forwarding

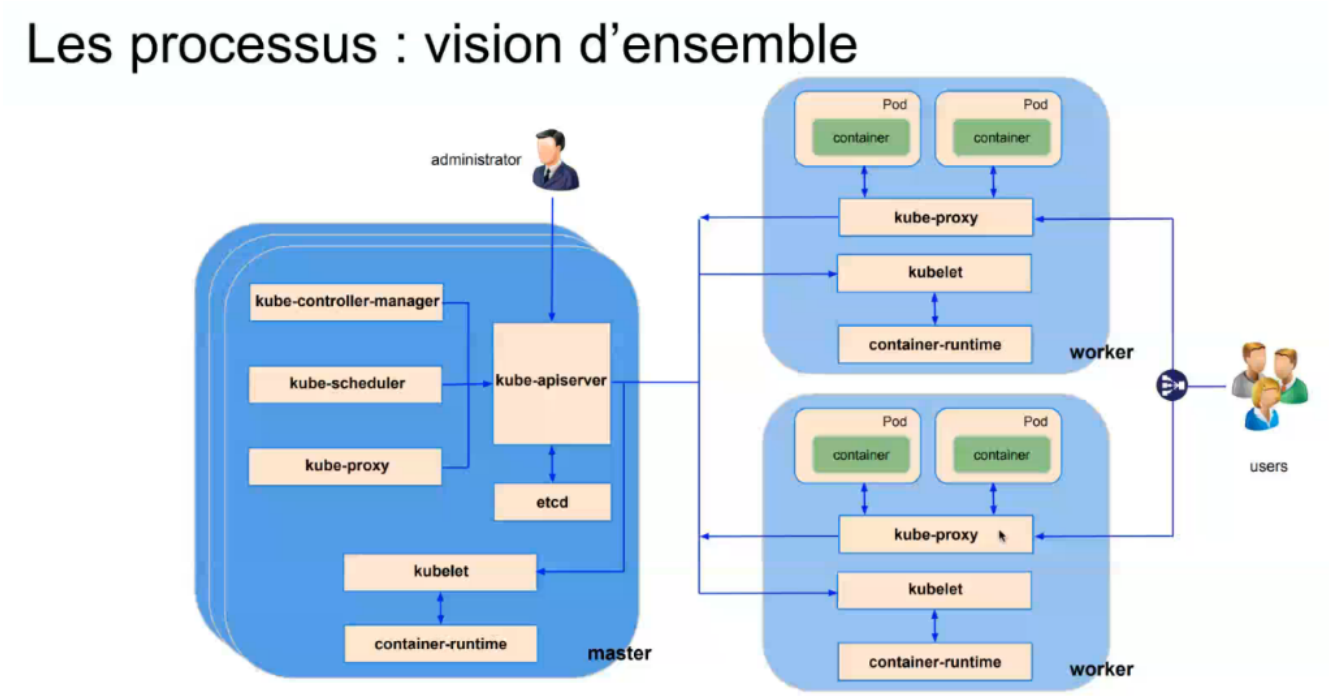
```
pod "mc1" deleted
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl apply -f pod_multi.yaml

pod/mc1 created
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl port-forward mc1 8099:8
0
error: unable to forward port because pod is not running. Current status=Pending
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl port-forward mc1 8099:8
0
Forwarding from 127.0.0.1:8099 -> 80
Forwarding from [::1]:8099 -> 80
Handling connection for 8099
```

(la 1ère commande de port forwarding a échoué car je l'ai lancé trop rapidement après avoir apply le script Kubernetes)

Test de connexion sur la page localhost:8099

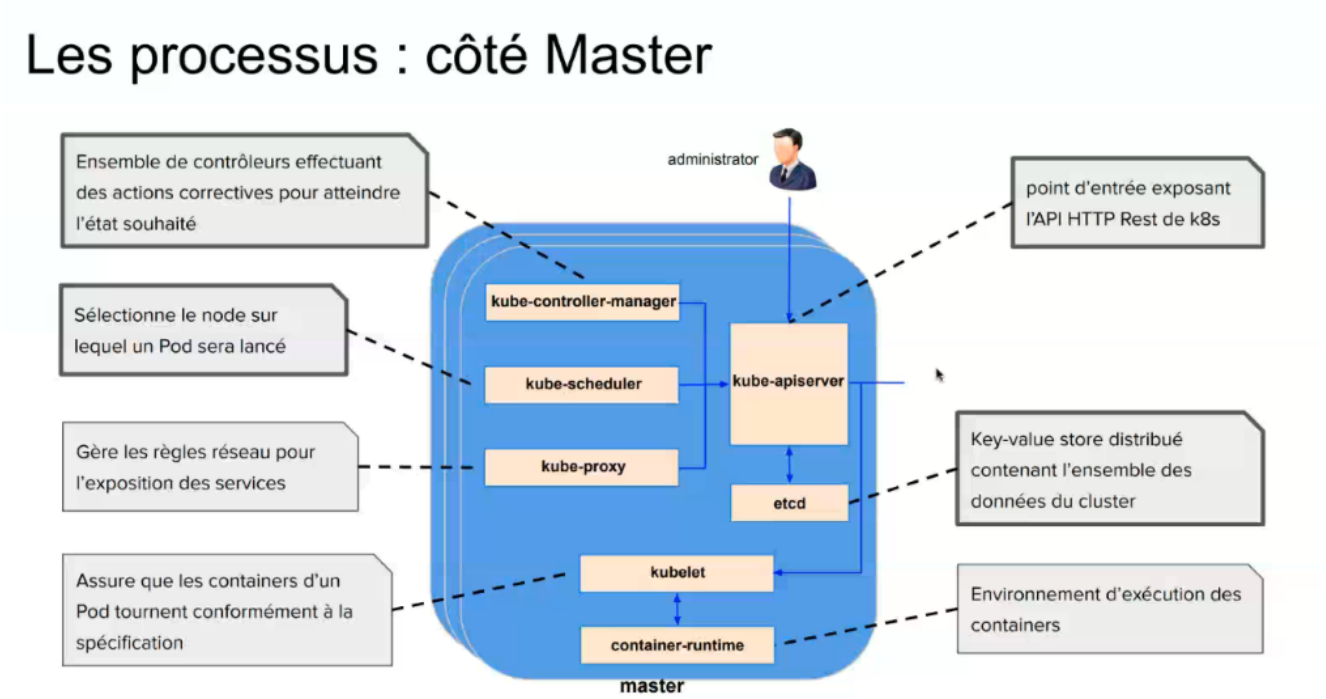


Vision d'ensemble : Kubernetes

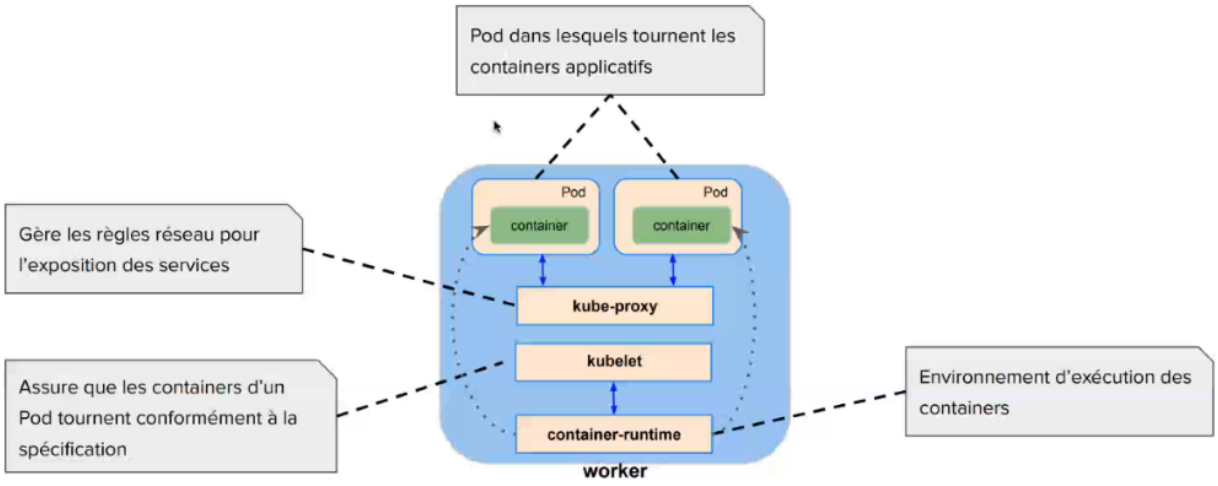


etcd : garde les informations sur les états des clusters

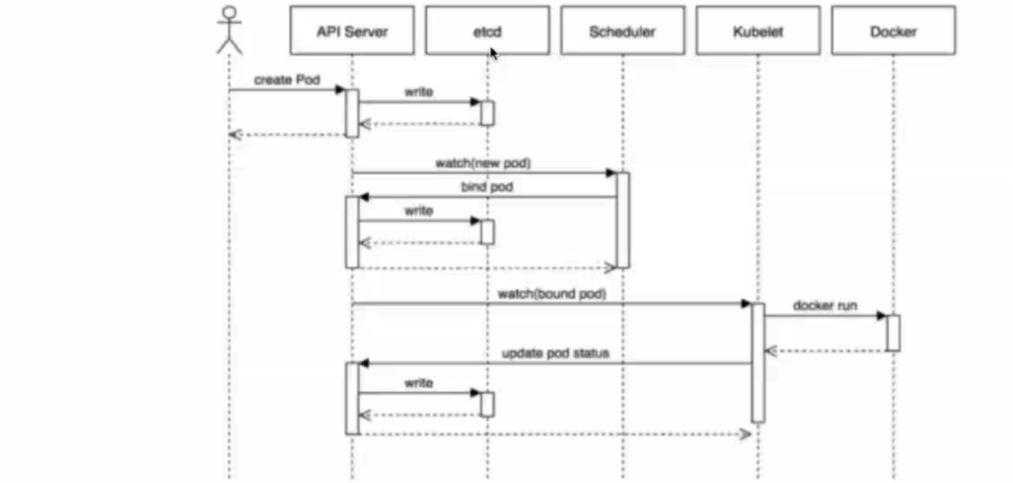
Container-runtime : endroit ou s'exécute les containers



Les processus : côté Worker



Les processus : workflow de création d'un Pod



Cycle de vie

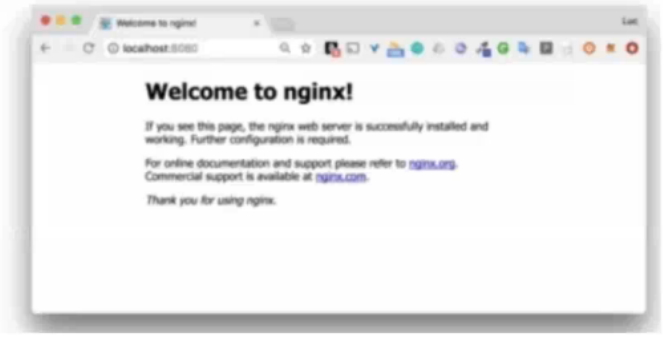
```
# Détails du Pod
$ kubectl describe po/www
Name:         www
Namespace:    default
Node:         minikube/192.168.99.100
...
Events:
  Type    Reason                 Age   From               Message
  ----    ------                 ----  ----               -------
  Normal  Scheduled              18s   default-scheduler  Successfully assigned nginx to minikube
  Normal  SuccessfulMountVolume  18s   kubelet, minikube  MountVolume.SetUp succeeded for volume "default-token-brp4l"
  Normal  Pulled                 18s   kubelet, minikube  Container image "nginx:1.12.2" already present on machine
  Normal  Created                18s   kubelet, minikube  Created container
  Normal  Started                18s   kubelet, minikube  Started container


# Suppression du Pod
$ kubectl delete pod www
pod "www" deleted
```

Ensemble des évènements survenus lors du lancement du Pod

Forward de port

- Commande utilisée pour le développement et debugging
- Permet de publier le port d'un Pod sur la machine hôte
- *$ kubectl port-forward POD_NAME HOST_PORT:CONTAINER_PORT*

```
$ kubectl port-forward www 8080:80
Forwarding from 127.0.0.1:8080 -> 80
```



Pod avec plusieurs containers

```
apiVersion: v1
kind: Pod
metadata:
  name: wp
spec:
  containers:
  - image: wordpress:4.9-apache
    name: wordpress
    env:
    - name: WORDPRESS_DB_PASSWORD
      value: mysqlpwd
    - name: WORDPRESS_DB_HOST
      value: 127.0.0.1
  - image: mysql:5.7
    name: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: mysqlpwd
    volumeMounts:
    - name: data
      mountPath: /var/lib/mysql
  volumes:
  - name: data
    emptyDir: {}
```

Pod avec plusieurs containers

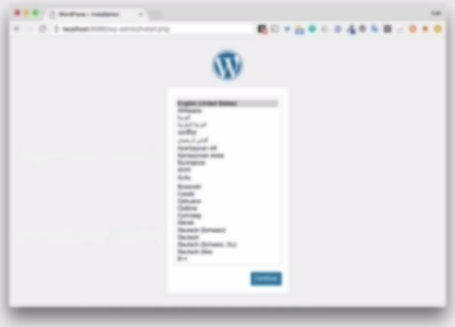
```
apiVersion: v1
kind: Pod
metadata:
  name: wp
spec:
  containers:
  - image: wordpress:4.9-apache
    name: wordpress
    env:
    - name: WORDPRESS_DB_PASSWORD
      value: mysqlpwd
    - name: WORDPRESS_DB_HOST
      value: 127.0.0.1
  - image: mysql:5.7
    name: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: mysqlpwd
    volumeMounts:
    - name: data
      mountPath: /var/lib/mysql
  volumes:
```

Montage du volume dans le container mysql

```
- name: data                    # Définition d'un volume: répertoire sur la machine hôte
  emptyDir: {}
```

Pod avec plusieurs containers

```
# Création du Pod
$ kubectl create -f wordpress-pod.yaml
Pod "wp" created

# Liste des Pod présent
$ kubectl get pods
NAME                    READY     STATUS    RESTARTS   AGE
wp                      2/2       Running   0          18s

# Exposition du port 80 du container wordpress
$ kubectl port-forward wp 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Handling connection for 8080
```



Les services :

https://fr.wikipedia.org/wiki/Microservices

> En informatique, les **microservices** sont une technique de développement logiciel — une variante du style architectural de l'architecture orientée services (SOA) — qui structure une application comme un ensemble de services faiblement couplés. Les microservices indépendants communiquent les uns avec les autres en utilisant des API indépendantes du langage de programmation.
>
> Des API REST sont souvent employées pour relier chaque microservice aux autres. Un avantage avancé est que lors d'un besoin critique de mise à jour d'une ressource, seul le microservice contenant cette ressource sera mis à jour, l'ensemble de l'application restant compatible avec la modification, contrairement à la totalité de l'application dans une architecture classique, par exemple une architecture trois tiers. Cependant, le coût de mise en place, en raison des compétences requises, est parfois plus élevé.

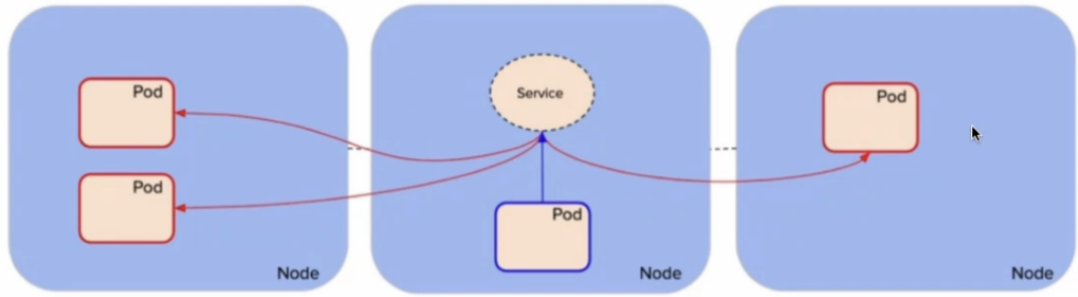
Rôle

- Expose les Pods d'une application via des règles réseaux
- Utilise des labels pour grouper les Pods
- Adresse IP persistante (VIP : virtual IP address)
- kube-proxy en charge du load balancing sur les Pods
  - userspaces / iptables / **IPVS**
- Différents types
  - ClusterIP (défaut) : exposition à l'intérieur du cluster
  - NodePort : exposition vers l'extérieur
  - LoadBalancer : intégration avec un Cloud Provider
  - ExternalName: associe le service à un nom DNS

Faire très attention à la portée des micro-services : il ne faut pas laisser des micro-services sensible en "**front**" (exposé à l'extérieur) -> faille de sécurité

On les fait communiquer en interne avec le service **ClusterIP**

Service de type ClusterIP



Service de type ClusterIP : exemple

```yaml
apiVersion: v1
kind: Service
metadata:
  name: vote
spec:
  selector:
    app: vote
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
```

Spécification du type de l'objet

service.yaml

Service de type ClusterIP : exemple

Chaque requête reçue par le service est envoyée sur l'un des Pods ayant le label spécifié

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

pod.yaml

kubectl apply -f pod.yaml

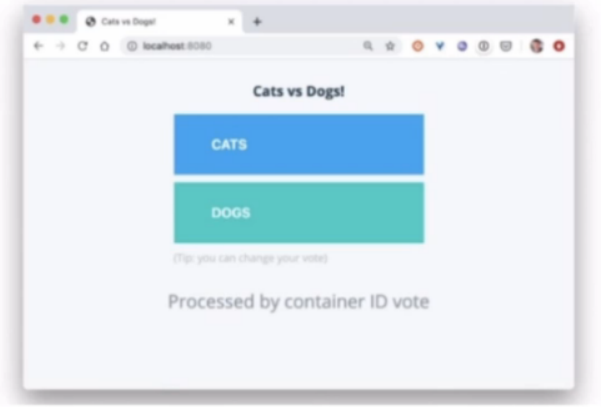
```yaml
apiVersion: v1
kind: Service
metadata:
  name: vote
spec:
  selector:
    app: vote
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
```

service.yaml

kubectl apply -f service.yaml

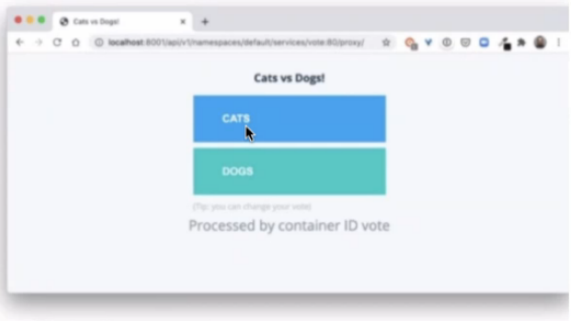
Service de type ClusterIP : accès via le port-forward

```
# Accès au service depuis l'extérieur
$ kubectl port-forward svc/vote 8080:80
```



Service de type ClusterIP : accès via le proxy

```
# Lancement du proxy (port 8001)
$ kubectl proxy
```

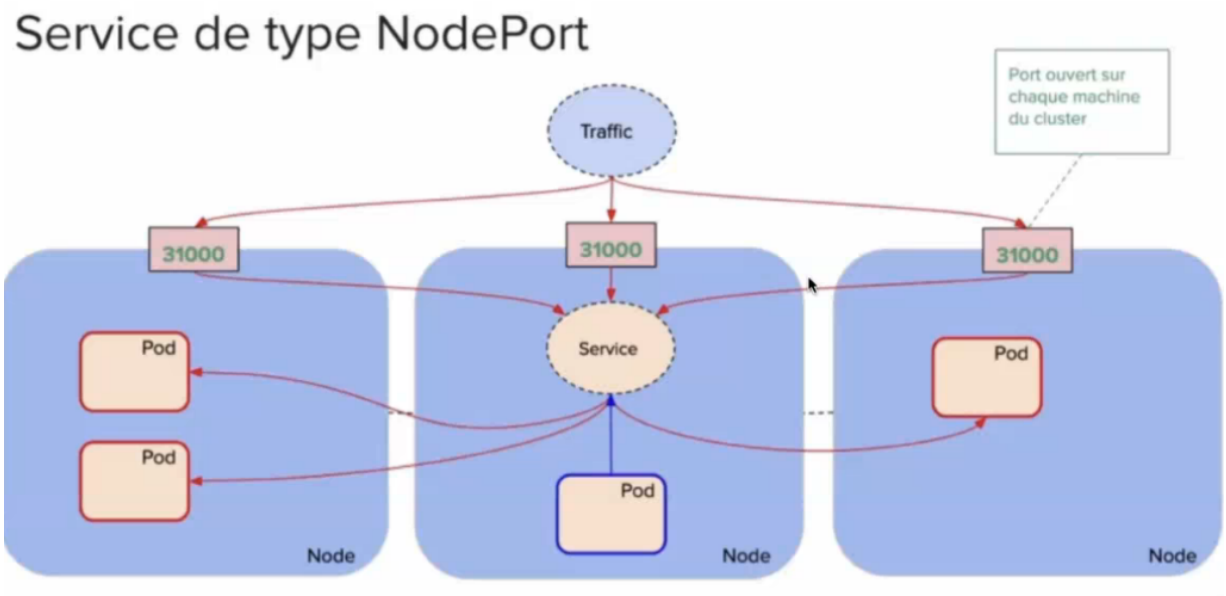


Service accessible au travers du proxy via l'URL:

http://localhost:8001/api/v1/namespaces/default/services/vote:80/proxy

Exposer service vers l'extérieur : service **NodePort**

Service de type NodePort



Par défaut, si on ne précise pas que le **type du service** sera un **NodePort** -> K8S va créer un **ClusterIP**

Service de type NodePort : exemple

```yaml
apiVersion: v1
kind: Service
metadata:
  name: vote-np
spec:
  selector:
    app: vote
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    nodePort: 31000
```

Service de type NodePort, exposé sur chaque node du cluster

service-np.yaml

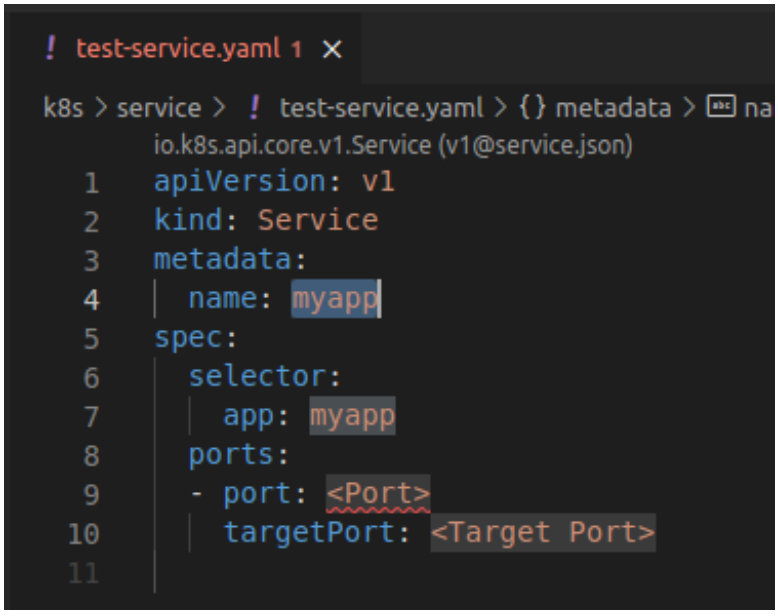
Service de type NodePort : exemple

```yaml
apiVersion: v1
kind: Service
metadata:
  name: vote-np
spec:
  selector:
    app: vote
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    nodePort: 31000
```

service-np.yaml

```
$ kubectl apply -f service-np.yaml
```

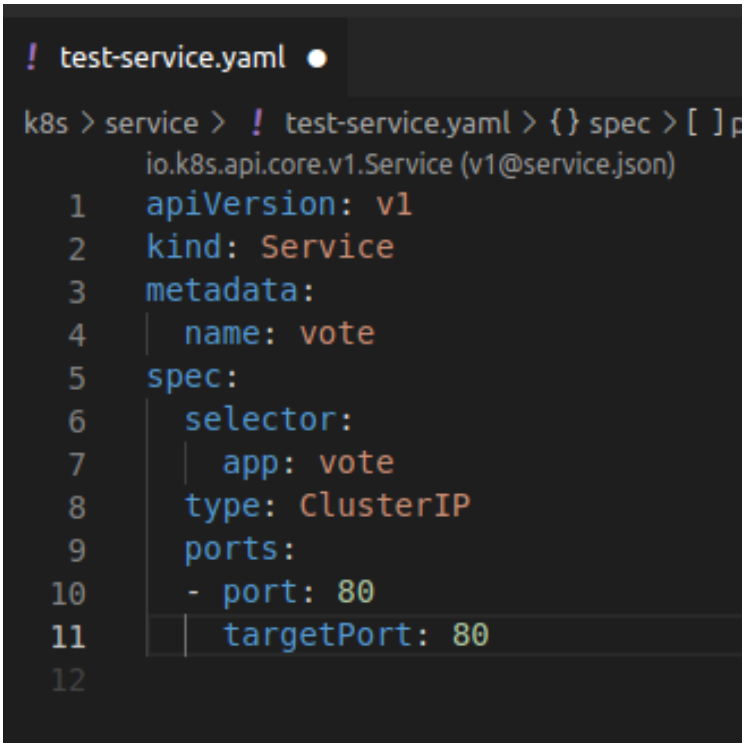
Démo :



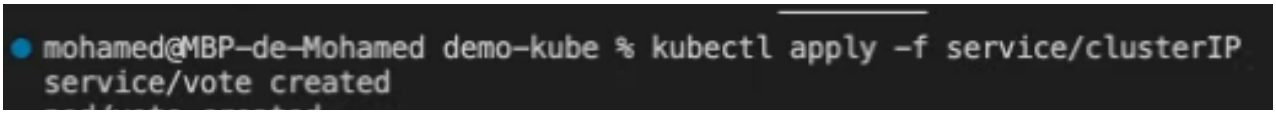
Dans un service, ne pas oublier le **type**

Ports:

- Port : "port du service"
- TargetPort: "port du containers"



Création d'un pod lié à ce service :



```
pod/vote created
mohamed@MBP-de-Mohamed demo-kube % kubectl get po
NAME    READY   STATUS              RESTARTS   AGE
mc1     2/2     Running             0          141m
nginx   1/1     Running             0          13h
vote    0/1     ContainerCreating   0          4s
```

Voir les services en cours sur k8s : **kubectl get svc**

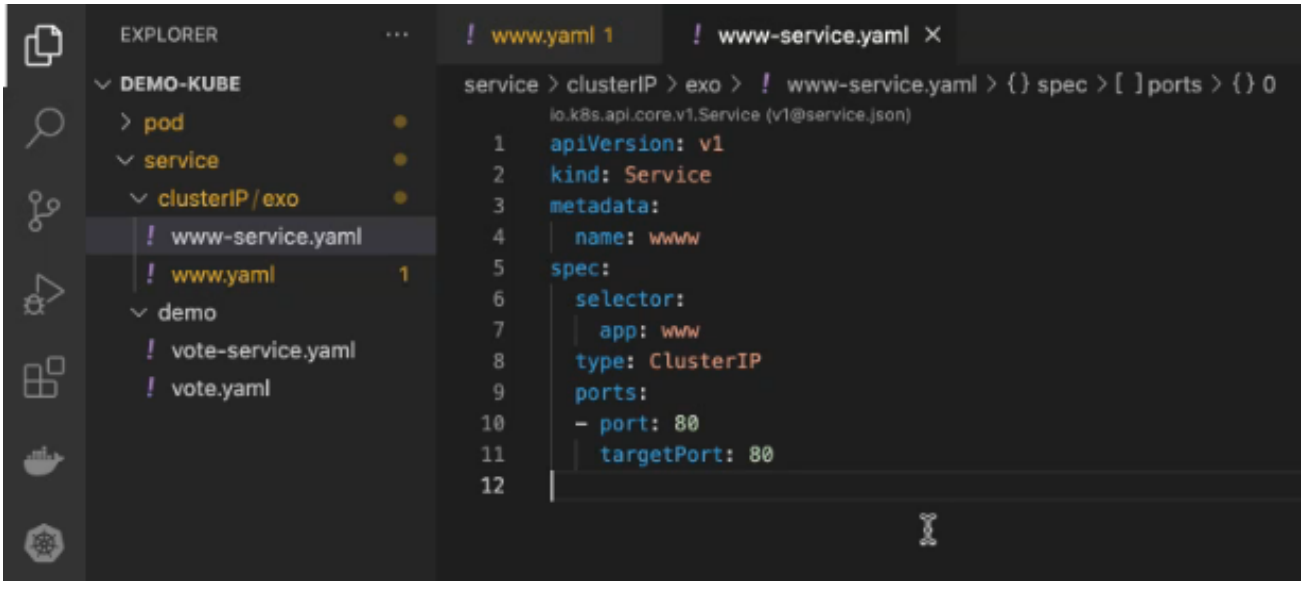
```
vel@vel-Precision-3650-Tower:~/Atelier/k8s$ kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP   2d21h
vote         ClusterIP   10.96.127.2    <none>        80/TCP    2m17s
```

TP : création ressource **cluster IP** + **pod** + **wget** depuis un **pod** pour récupérer l'**index.html**

www_service_clusterIP.yaml | www_pod.yaml | www_debug.yaml 1

k8s > pod > tp_www > ! www_pod.yaml > {} metadata > name

```yaml
3   metadata:
4     name: www
5     labels:
6       name: www
7   spec:
8     containers:
9     - name: nginx
10      image: nginx
11      resources:
12        limits:
13          memory: "128Mi"
14          cpu: "500m"
15      ports:
16        - containerPort: 80
17
```



! www.yaml 1 ! debug.yaml 1 × ! www-service.yaml

service > clusterIP > exo > ! debug.yaml > {} spec > [ ] containers > {} 0 > [ ] command > 1

io.k8s.api.core.v1.Pod (v1@pod.json)

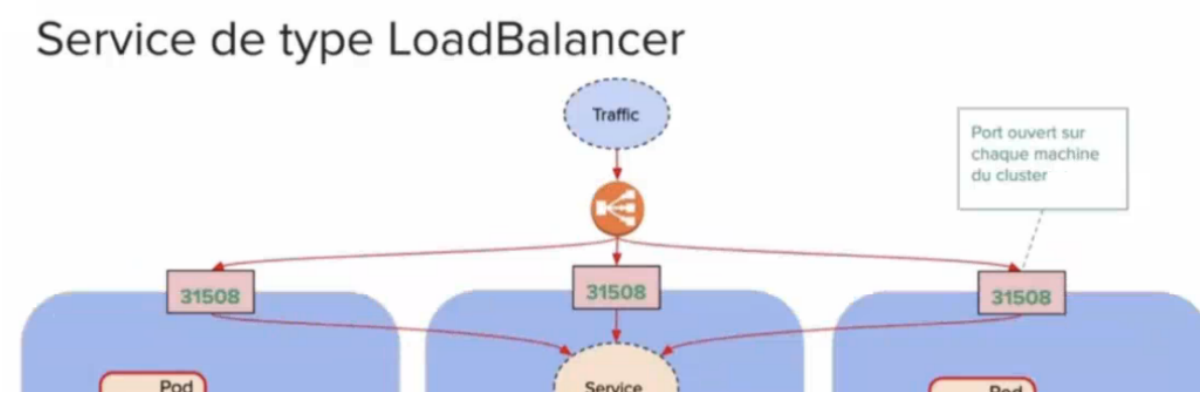
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: debug
  labels:
    name: debug
spec:
  containers:
  - name: debug
    image: alpine:3.15
    command:
      - "sleep"
      - "10000"

```

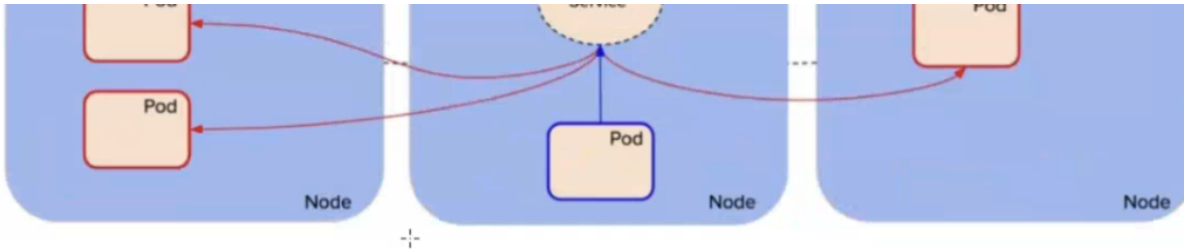
```
mohamed@MBP-de-Mohamed demo-kube % kubectl get svc,po
NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP   3d19h
service/vote         ClusterIP   10.96.124.139   <none>        80/TCP    153m
service/wwww         ClusterIP   10.96.225.58    <none>        80/TCP    16s

NAME         READY   STATUS    RESTARTS   AGE
pod/debug    1/1     Running   0          16s
pod/mc1      2/2     Running   0          4h54m
pod/nginx    1/1     Running   0          16h
pod/vote     1/1     Running   0          153m
pod/www      1/1     Running   0          16s
mohamed@MBP-de-Mohamed demo-kube % kubectl exec -it debug -- sh
/ #
```

```
mohamed@MBP-de-Mohamed demo-kube % kubectl get no -o wide
NAME                STATUS   ROLES           AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION     CONTAINER-RUNTIME
k8s-control-plane   Ready    control-plane   3d19h   v1.25.3   172.31.0.5    <none>        Ubuntu 22.04.1 LTS   5.15.49-linuxkit   containerd://1.6.9
mohamed@MBP-de-Mohamed demo-kube % export KUBECONFIG=k8s-1-25-4-do-0-ams3-1676230768828-kubeconfig.yaml
mohamed@MBP-de-Mohamed demo-kube % kubectl config get-contexts
CURRENT   NAME                                          CLUSTER                                       AUTHINFO                                            NAMESPACE
*         do-ams3-k8s-1-25-4-do-0-ams3-1676230768828   do-ams3-k8s-1-25-4-do-0-ams3-1676230768828   do-ams3-k8s-1-25-4-do-0-ams3-1676230768828-admin
mohamed@MBP-de-Mohamed demo-kube % kubectl get nodes
NAME                STATUS   ROLES    AGE   VERSION
pool-r6acqqffg-qqkj7   Ready    <none>   18h   v1.25.4
pool-r6acqqffg-qqkjm   Ready    <none>   18h   v1.25.4
mohamed@MBP-de-Mohamed demo-kube % kubectl get node -w o
Error from server (NotFound): nodes "o" not found
mohamed@MBP-de-Mohamed demo-kube % kubectl get node -o wide
NAME                STATUS   ROLES    AGE   VERSION   INTERNAL-IP   EXTERNAL-IP      OS-IMAGE                         KERNEL-VERSION          CONTAINER-RUNTIME
pool-r6acqqffg-qqkj7   Ready    <none>   18h   v1.25.4   10.110.0.3    174.138.10.202   Debian GNU/Linux 11 (bullseye)   5.18.0-0.deb11.4-amd64   containerd://1.4.
pool-r6acqqffg-qqkjm   Ready    <none>   18h   v1.25.4   10.110.0.2    167.99.41.161    Debian GNU/Linux 11 (bullseye)   5.18.0-0.deb11.4-amd64   containerd://1.4.
mohamed@MBP-de-Mohamed demo-kube %
```

Fin du tp

Service : **loadbalancer**



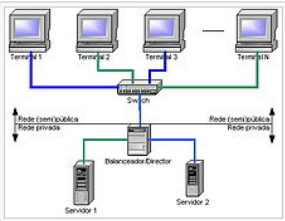


Pour les articles homonymes, voir Load et balance.

En informatique, la **répartition de charge** (en anglais : *load balancing*) désigne le processus de répartition d'un ensemble de tâches sur un ensemble de ressources, dans le but d'en rendre le traitement global plus efficace. Les techniques de répartition de charge permettent à la fois d'optimiser le temps de réponse pour chaque tâche, tout en évitant de surcharger de manière inégale les nœuds de calcul.

La *répartition de charge* est issue de la recherche dans le domaine des ordinateurs parallèles. Deux principales approches coexistent : les algorithmes statiques, qui ne tiennent pas compte de l'état des différentes machines, et les algorithmes dynamiques, qui sont en général plus généraux et performants, mais nécessitent des échanges d'information entre les différentes unités de calculs, au risque d'une perte d'efficacité.

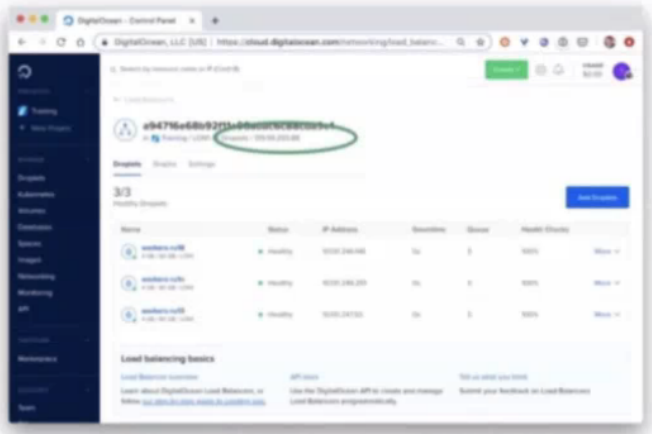
Ces techniques sont par exemple très utilisées dans le domaine des services HTTP où un site à forte audience doit pouvoir gérer des centaines de milliers de requêtes par seconde.



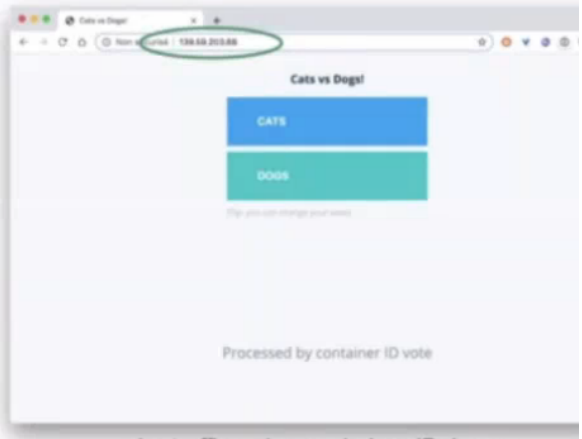
Répartition de charge entre deux serveurs accédés par 4 postes clients, ici le switch et le répartiteur sont deux points de panne potentiels, aucun des deux n'est doublé.

Service de type LoadBalancer : exemple

```
$ kubectl get svc
NAME         TYPE           CLUSTER-IP       EXTERNAL-IP      PORT(S)        AGE
vote-lb      LoadBalancer   10.245.186.209   139.59.203.88    80:31243/TCP   3m4s
```



Création un LoadBalancer sur DigitalOcean



Le traffic arrive sur le LoadBalancer

Création avec l'approche impérative : exemple (1/3)

```
# Création d'un Pod
$ kubectl run whoami --image containous/whoami
```

Rappel: le label run:NOM_DU_POD est automatiquement défini dans la spécification du Pod

```
# Exposition via un Service NodePort
(dry run)
$ kubectl expose pod whoami \
  --type=NodePort \
```

```
apiVersion: v1
kind: Service
metadata:
  labels:
    run: whoami
  name: whoami
spec:
```

```
--port=8080 \
--target-port=80 \
--dry-run=client \
-o yaml
```

Le label du Pod est utilisé dans le selector du Service

```
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 80
  selector:
    run: whoami
  type: NodePort
```

Création avec l'approche impérative : exemple (3/3)

```
# Création d'un Pod + Service (dry run)
$ kubectl run db \
  --image=mongo:4.2 \
  --port=27017 \
  --expose \
  --dry-run=client \
  -o yaml
```

2 ressources en une seule commande

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: db
  name: db
spec:
  containers:
  - image: mongo:4.2
    name: db
    ports:
    - containerPort: 27017
  dnsPolicy: ClusterFirst
  restartPolicy: Always
```

```
apiVersion: v1
kind: Service
metadata:
  name: db
spec:
  ports:
  - port: 27017
    protocol: TCP
    targetPort: 270
  selector:
    run: db
```

Les commandes de base

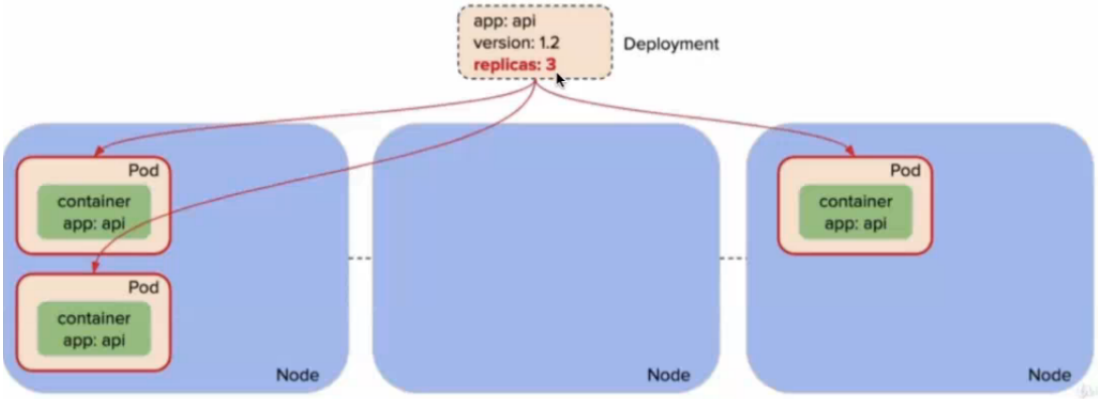
```
# Création d'un Service
$ kubectl apply -f PATH_SPECIFICATION
$ kubectl expose …
$ kubectl create service ...

# Liste de l'ensemble des services
$ kubectl get service / kubectl get svc

# Principales informations concernant un service
$ kubectl get svc/SERVICE_NAME

# Informations détaillées d'un service
$ kubectl describe svc SERVICE_NAME

# Suppression d'un service
$ kubectl delete svc/SERVICE_NAME
```

Les Deployments

Rôle

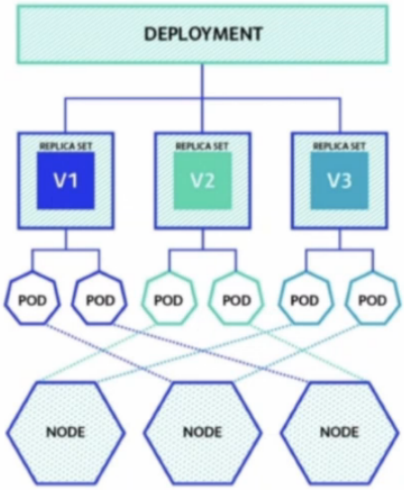
Un **Deployment** permet de gérer un ensemble de **Pods** identiques (mise à jour / rollback)

Un **Deployment** permet de gérer un ensemble de **Pods** identiques (mise à jour / rollback)



Utilisation

- Gestion du cycle de vie de Pods
  - Création / Suppression
  - Scaling
  - Rollout / Rollback
- Différents niveaux d'abstraction
  - Deployment
  - ReplicaSet
  - Pod



Rollout : revenir sur ancienne version du **replicaSet**

Spécification d'un Deployment

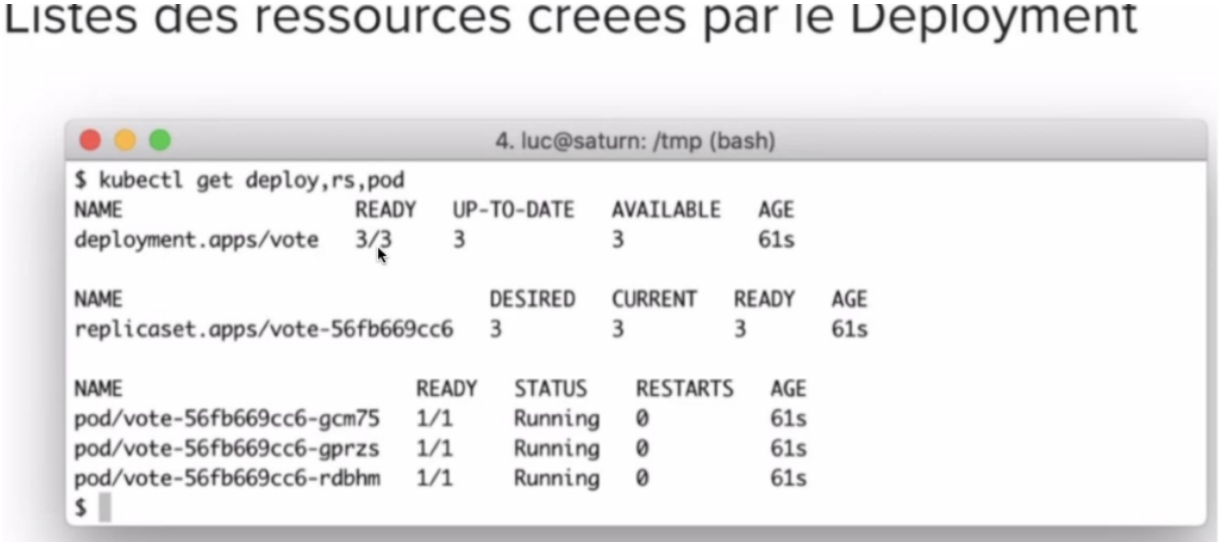
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vote
spec:
  replicas: 3
  selector:
    matchLabels:
      app: vote
  template:
    metadata:
      labels:
        app: vote
    spec:
      containers:
      - name: vote
        image: instavote/vote
        ports:
        - containerPort: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

Deployment

ReplicaSet

Pod

Listes des ressources creees par le Deployment



Création avec l'approche impérative

```
$ kubectl create deploy vote --image instavote/vote
```

- Plusieurs limitations ⇒ il n'est pas possible de :
  - spécifier le nombre de réplicas (1 par défaut)
  - spécifier plusieurs containers
  - ...
- Pratique mais beaucoup moins flexible qu'une spécification yaml

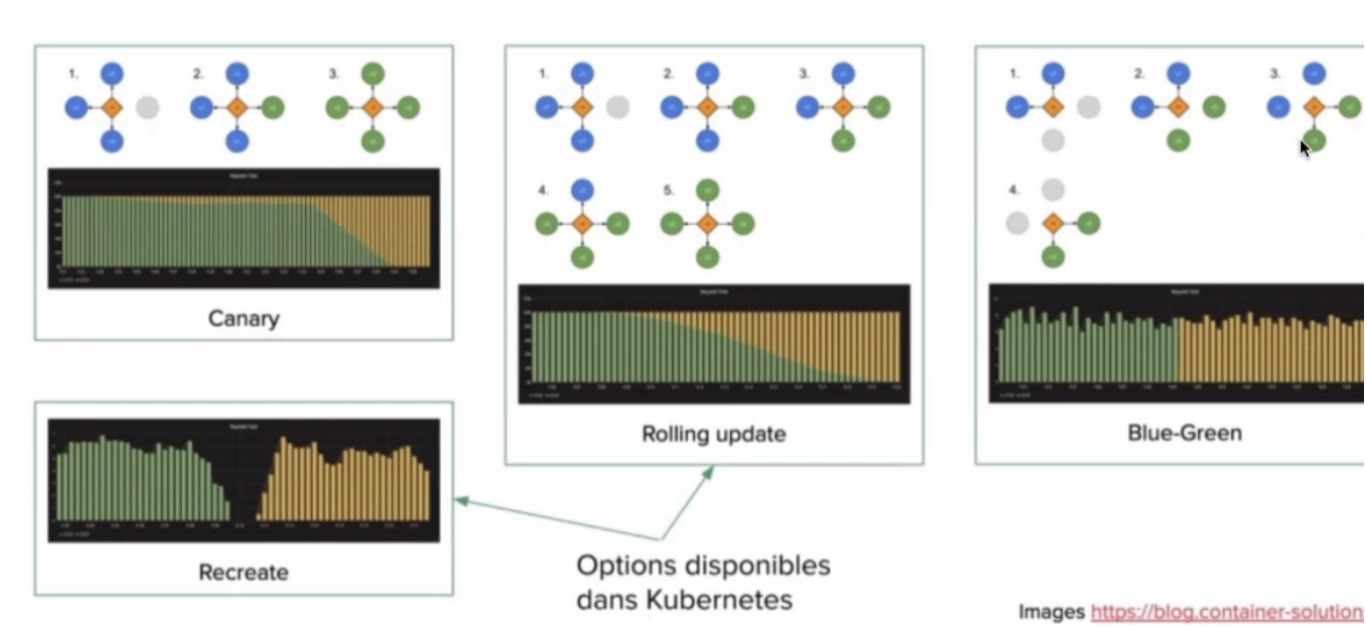
Génération de la spécification d'un Deployment

```
$ kubectl create deploy vote \
  --image instavote/vote \
  --dry-run=client \
  -o yaml
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: vote
  name: vote
spec:
  replicas: 1
  selector:
    matchLabels:
      app: vote
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: vote
    spec:
      containers:
      - image: instavote/vote
        name: vote
...
```

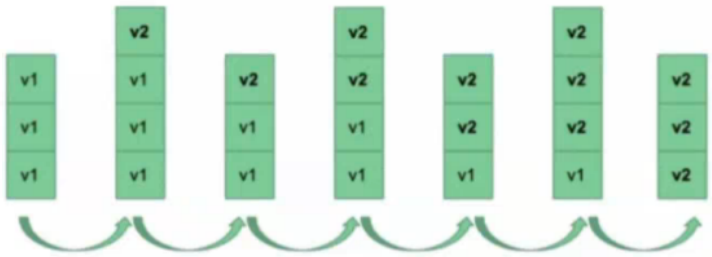
L'option **--dry-run** simule la création d'une ressource
- version < 1.18 ⇒ elle s'utilise sans valeur
- version >= 1.18 ⇒ 2 valeurs possibles
  * client: la ressource n'est pas envoyée à l'API Server
  * server: traitée par l'API Server mais non persistée

Mise à jour d'une application (général)



Mise à jour d'un Deployment : rolling update

- Mise à jour graduelle de l'ensemble des Pods
- Paramètres pour contrôler la mise à jour
  - maxUnavailable
  - maxSurge




```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: www
spec:
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  ...
```

MaxSurge : nombre de ressources **en plus** toléré

MaxUnavailable : nombre de ressource **en moins** toléré

Mise à jour à partir de la spécification

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vote
spec:
  replicas: 3
  selector:
    matchLabels:
      app: vote
  template:
    metadata:
      labels:
        app: vote
    spec:
```

```
$ kubectl apply -f deploy.yaml
```

```
containers:
- name: vote
  image: instavote/vote:indent
  ports:
  - containerPort: 80
```

deploy.yaml

Mises à jour avec l'approche impérative

```
$ kubectl set image deploy/vote vote=instavote/vote:movies --record
```

Note: le flag **--record** est à **false** par défaut, il enregistre la commande dans une annotation de la ressource

Historique des mises à jour

```
$ kubectl rollout history deploy/vote
deployments "vote-deploy"
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
3         kubectl set image deploy/vote vote=instavote/vote:movies --record=true
```

- CHANGE-CAUSE contient la commande qui a amené à la version correspondante si le flag --record a été spécifié
- 10 révisions par défaut ⇒ modifiable via la propriété ***.spec.revisionHistoryLimit*** du Deployment

Rollback

Retour vers la révision précédente ou une révision ultérieure

```
$ kubectl rollout undo deploy/vote

$ kubectl rollout undo deploy/vote --to-revision=X
```

Forcer la mise à jour

```
$ kubectl rollout restart deploy/www
```

```
$ kubectl get pods -w
NAME                   READY   STATUS              RESTARTS   AGE
www-567cb66955-6b4nw   1/1     Running             0          73s
www-567cb66955-rznp7   1/1     Running             0          69s
www-5588bfccff-d79g8   0/1     Pending             0          0s
www-5588bfccff-d79g8   0/1     ContainerCreating   0          0s
www-5588bfccff-d79g8   1/1     Running             0          2s
www-567cb66955-rznp7   1/1     Terminating         0          73s
www-5588bfccff-fmrh8   0/1     Pending             0          0s
www-5588bfccff-fmrh8   0/1     ContainerCreating   0          0s
www-5588bfccff-fmrh8   1/1     Running             0          2s
www-567cb66955-6b4nw   1/1     Terminating         0          79s
...
```

Scaling horizontal

Modification du nombre de Pods gérés par un Deployment / ReplicaSet / StatefulSet

```
# Création d'un Deployment
$ kubectl create deploy www --image=nginx:1.16

# Augmentation du nombre de réplicas
$ kubectl scale deploy/www --replicas=5
```

HorizontalPodAutoscaler (½)

Modification du nombre de Pods en fonction de l'utilisation du CPU

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: www
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: www
  minReplicas: 2
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

Création depuis une spécification hpa.yaml

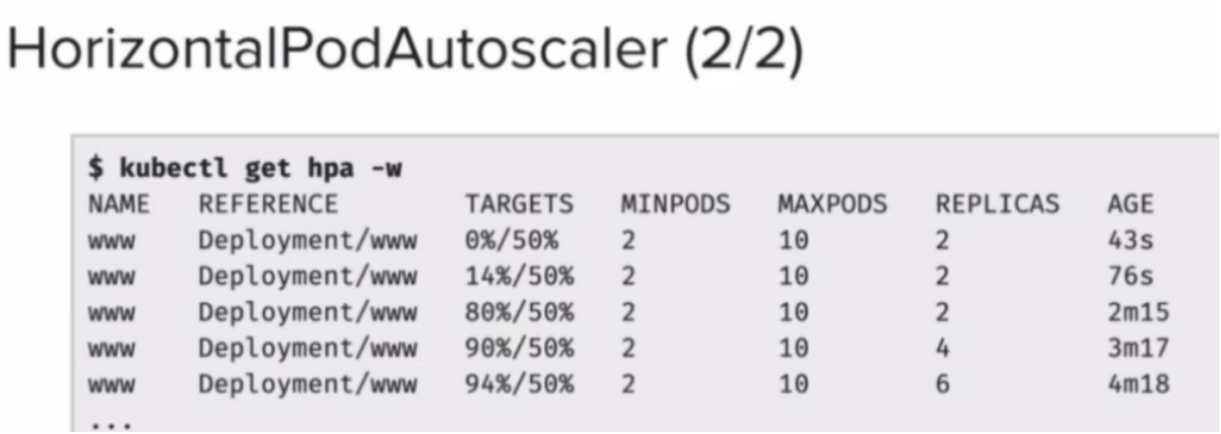
```
$ kubectl apply -f hpa.yaml
```

=

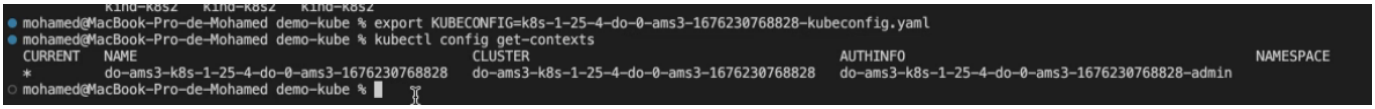
```
$ kubectl autoscale \
  deploy www \
  --min=2 \
  --max=10 \
  --cpu-percent=50
```

Création avec une commande impérative

Kind: **HorizontalPodAutoscaler** permet de provisionner automatiquement un nouveau pod dès qu'une alerte est détecté : ici une **alerte CPU**

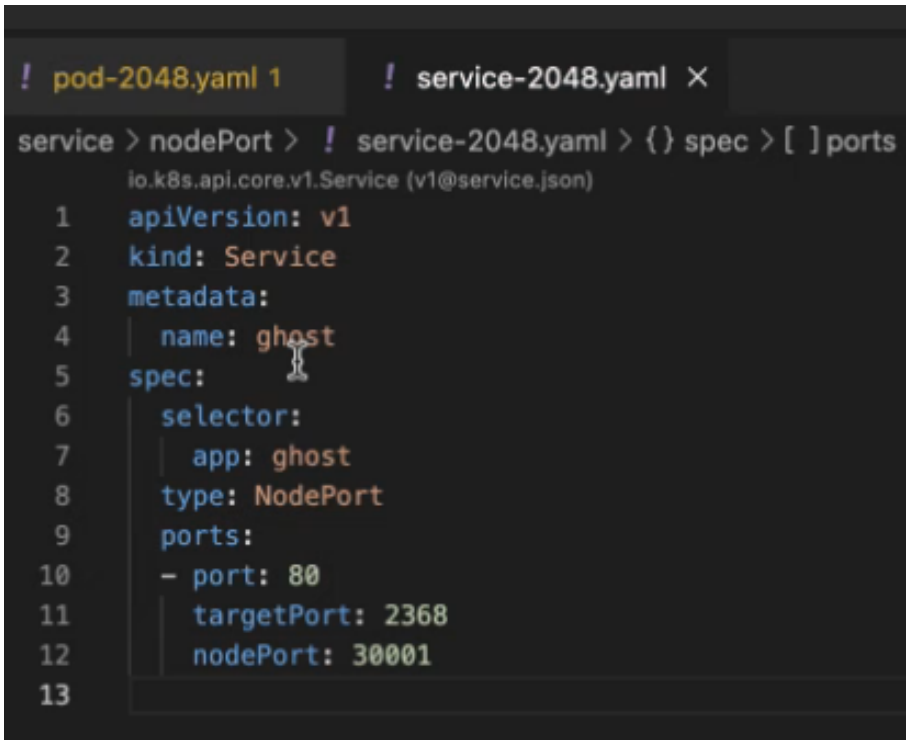


Exemple avec un Cloud Providers :

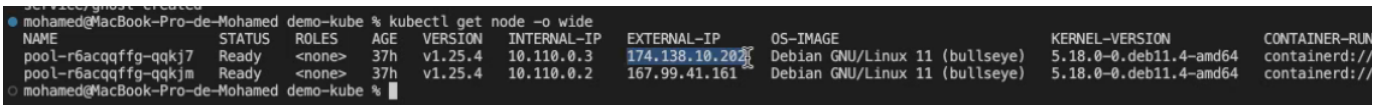


Le fichier est fournis par le cloud providers pour faire le lien entre la machine et les ressources du cloud providers

Création d'un pod ghost + un service nodePort



https://devopssec.fr/article/gerer-manipuler-services-kubernetes



On se référe à l'external IP pour accéder à notre page web

Information concernant les ports à utiliser pour un service de type nodePort :

- `NodePort` : Il expose le service vers l'extérieur du cluster à l'aide du NAT (la plage de ports autorisés est entre 30000 et 32767).

https://devopssec.fr/article/gerer-manipuler-services-kubernetes

Test avec un fichier deployement :

deployment-ghost.yaml 1

deployment > deployment-ghost.yaml > { } spec > { } templat

io.k8s.api.apps.v1.Deployment (v1@deployment.json)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ghost
spec:
  replicas: 3
  selector:
    matchLabels:
      app: ghost
  template:
    metadata:
      labels:
        app: ghost
    spec:
      containers:
      - name: ghost
        image: ghost:4
        ports:
        - containerPort: 2368
```

Apply du fichier deployement :

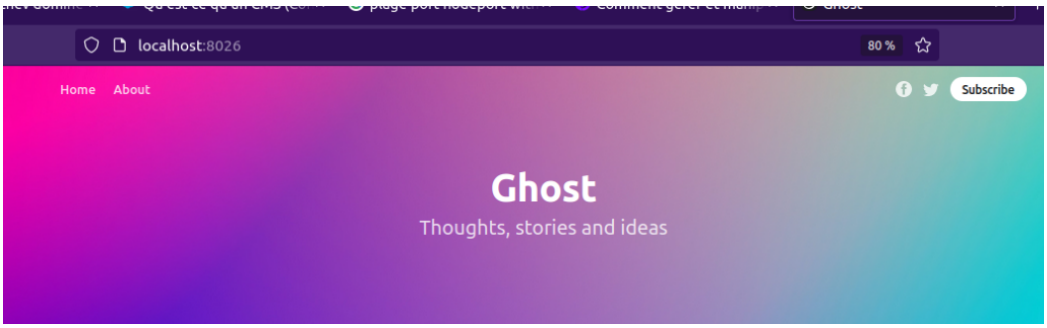
```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/deployment$ kubectl apply -f deployment-gho
st.yaml
deployment.apps/ghost created
vel@vel-Precision-3650-Tower:~/Atelier/k8s/deployment$ kubectl get po
NAME                     READY   STATUS              RESTARTS      AGE
debug                    1/1     Running             1 (70m ago)   21h
demo                     1/1     Running             2 (70m ago)   3d17h
ghost-65bbfbf5dd-bvtvb   0/1     ContainerCreating   0             10s
ghost-65bbfbf5dd-hnfct   0/1     ContainerCreating   0             10s
ghost-65bbfbf5dd-phsv4   0/1     ContainerCreating   0             10s
httpd                    1/1     Running             3 (70m ago)   3d18h
vote                     1/1     Running             1 (70m ago)   22h
www                      1/1     Running             1 (70m ago)   21h
```

```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/deployment$ kubectl get po,deploy,rs
NAME                         READY   STATUS    RESTARTS      AGE
pod/debug                    1/1     Running   1 (71m ago)   21h
pod/demo                     1/1     Running   2 (71m ago)   3d17h
pod/ghost-65bbfbf5dd-bvtvb   1/1     Running   0             75s
pod/ghost-65bbfbf5dd-hnfct   1/1     Running   0             75s
pod/ghost-65bbfbf5dd-phsv4   1/1     Running   0             75s
pod/httpd                    1/1     Running   3 (71m ago)   3d18h
pod/vote                     1/1     Running   1 (71m ago)   22h
pod/www                      1/1     Running   1 (71m ago)   21h

NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ghost   3/3     3            3           75s

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/ghost-65bbfbf5dd   3         3         3       75s
```

Port-forwarding du ghost : il faut obligatoirement utiliser **le nom d'un pod** pour faire le port forwarding

```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/deployment$ kubectl port-forward pod/ghost-
65bbfbf5dd-bvtvb 8026:2368
Forwarding from 127.0.0.1:8026 -> 2368
Forwarding from [::1]:8026 -> 2368
Handling connection for 8026
Handling connection for 8026
```

```
Handling connection for 8026
Handling connection for 8026
```



Si on supprime un pod :

```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl delete po/ghost-65bbfbf5dd-bvtvb
pod "ghost-65bbfbf5dd-bvtvb" deleted
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl get po
NAME                     READY   STATUS    RESTARTS      AGE
debug                    1/1     Running   1 (79m ago)   21h
demo                     1/1     Running   2 (79m ago)   3d18h
ghost-65bbfbf5dd-hnfct   1/1     Running   0             9m57s
ghost-65bbfbf5dd-p8xt2   1/1     Running   0             8s
ghost-65bbfbf5dd-phsv4   1/1     Running   0             9m57s
httpd                    1/1     Running   3 (79m ago)   3d19h
vote                     1/1     Running   1 (79m ago)   22h
www                      1/1     Running   1 (79m ago)   21h
```

Pour supprimer des ressources deploiement : kubectl delete –f **"nom du dossier/du fichier"**

```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/deployment$ kubectl delete -f deployment-ghost.yam

deployment.apps "ghost" deleted
```

Création Déploiement de manière **impérative** : **kubectl create deploy**

```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/deployment$ kubectl create deploy www --image nginx
:1.16
deployment.apps/www created
vel@vel-Precision-3650-Tower:~/Atelier/k8s/deployment$ kubectl get po
NAME                   READY   STATUS    RESTARTS      AGE
debug                  1/1     Running   1 (89m ago)   21h
demo                   1/1     Running   2 (89m ago)   3d18h
httpd                  1/1     Running   3 (89m ago)   3d19h
vote                   1/1     Running   1 (89m ago)   22h
www                    1/1     Running   1 (89m ago)   21h
www-86b79cddd-cc5k5    1/1     Running   0             6s
vel@vel-Precision-3650-Tower:~/Atelier/k8s/deployment$ kubectl get po,deploy,rs
NAME                       READY   STATUS    RESTARTS      AGE
pod/debug                  1/1     Running   1 (90m ago)   21h
pod/demo                   1/1     Running   2 (90m ago)   3d18h
pod/httpd                  1/1     Running   3 (90m ago)   3d19h
pod/vote                   1/1     Running   1 (90m ago)   22h
pod/www                    1/1     Running   1 (90m ago)   21h
pod/www-86b79cddd-cc5k5    1/1     Running   0             20s

NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/www   1/1     1            1           20s

NAME                             DESIRED   CURRENT   READY   AGE
replicaset.apps/www-86b79cddd    1         1         1       20s
vel@vel-Precision-3650-Tower:~/Atelier/k8s/deployment$
```

Pour scale un deployement : **kubectl scale "nom_deployment" --replicas nombre_deployment"**

```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl scale deploy/www --replicas 3
deployment.apps/www scaled
vel@vel-Precision-3650-Tower:~/Atelier/k8s/demo_kind$ kubectl get po,deploy,rs
NAME                       READY   STATUS    RESTARTS      AGE
pod/debug                  1/1     Running   1 (92m ago)   21h
pod/demo                   1/1     Running   2 (92m ago)   3d18h
pod/httpd                  1/1     Running   3 (92m ago)   3d19h
pod/vote                   1/1     Running   1 (92m ago)   22h
pod/www                    1/1     Running   1 (92m ago)   21h
pod/www-86b79cddd-cc5k5    1/1     Running   0             2m28s
```

```
pod/www-86b79cddd-w8njg   1/1     Running   0          15s
pod/www-86b79cddd-xqs22   1/1     Running   0          15s

NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/www   3/3     3            3           2m28s

NAME                             DESIRED   CURRENT   READY   AGE
replicaset.apps/www-86b79cddd    3         3         3       2m28s
```

Pour mettre à jour image utilisé dans les pods : **kubectl set image "nom_deployment" "nom_image_dockerhub" --record**

```
www-86b79cddd-d5xqg          1/1     Running   0          9m4s
mohamed@MacBook-Pro-de-Mohamed demo-kube % kubectl set image deploy/www nginx=nginx:1.16-alpine --record
Flag --record has been deprecated, --record will be removed in the future
deployment.apps/www image updated
mohamed@MacBook-Pro-de-Mohamed demo-kube %
```

Le –record nous permet d'avoir des informations sur les modifications effectué :

```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/deployment$ kubectl set image deplo
y/www nginx=nginx:1.16-alpine --record
Flag --record has been deprecated, --record will be removed in the future
deployment.apps/www image updated
vel@vel-Precision-3650-Tower:~/Atelier/k8s/deployment$ kubectl get po,deploy,r
s
NAME                        READY   STATUS    RESTARTS       AGE
pod/debug                   1/1     Running   1 (104m ago)   22h
pod/demo                    1/1     Running   2 (104m ago)   3d18h
pod/httpd                   1/1     Running   3 (104m ago)   3d19h
pod/vote                    1/1     Running   1 (104m ago)   22h
pod/www                     1/1     Running   1 (104m ago)   22h
pod/www-7b757d86f8-8b4md    1/1     Running   0              11s
pod/www-7b757d86f8-pfs2r    1/1     Running   0              10s
pod/www-7b757d86f8-t596b    1/1     Running   0              16s

NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/www   3/3     3            3           14m

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/www-7b757d86f8      3         3         3       16s
replicaset.apps/www-86b79cddd       0         0         0       14m
```

Quand on fait notre **kubectl get po** -> on voit bien les modifications apporté avec le set de l'image **dockerhub**

Pour vérifier l'**historique des différentes versions du deployment** : **kubectl rollout history "nom_deploiment"**

```
See 'kubectl rollout -h' for help and examples
vel@vel-Precision-3650-Tower:~/Atelier/k8s/deployment$ kubectl rollout history
 deploy/www
deployment.apps/www
REVISION  CHANGE-CAUSE
1         <none>
2         kubectl set image deploy/www nginx=nginx:1.16-alpine --record=true
```

Pour faire un rollback sur une version spécifique : **kubectl rollout undo deploy/www**

```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/deployment$ kubectl rollout undo de
ploy/www
deployment.apps/www rolled back
```

Pour voir la 1ère image d'un node : **kubectl get deploy/www -o jsonpath='{.spec.template.spec.containers[0].image}'**

```
vel@vel-Precision-3650-Tower:~/Atelier/k8s/deployment$ kubectl get deploy/ww
w -o jsonpath='{.spec.template.spec.containers[0].image}'
nginx:1.16vel@vel-Precision-3650-Tower:~/Atelier/k8s/deployment$
```