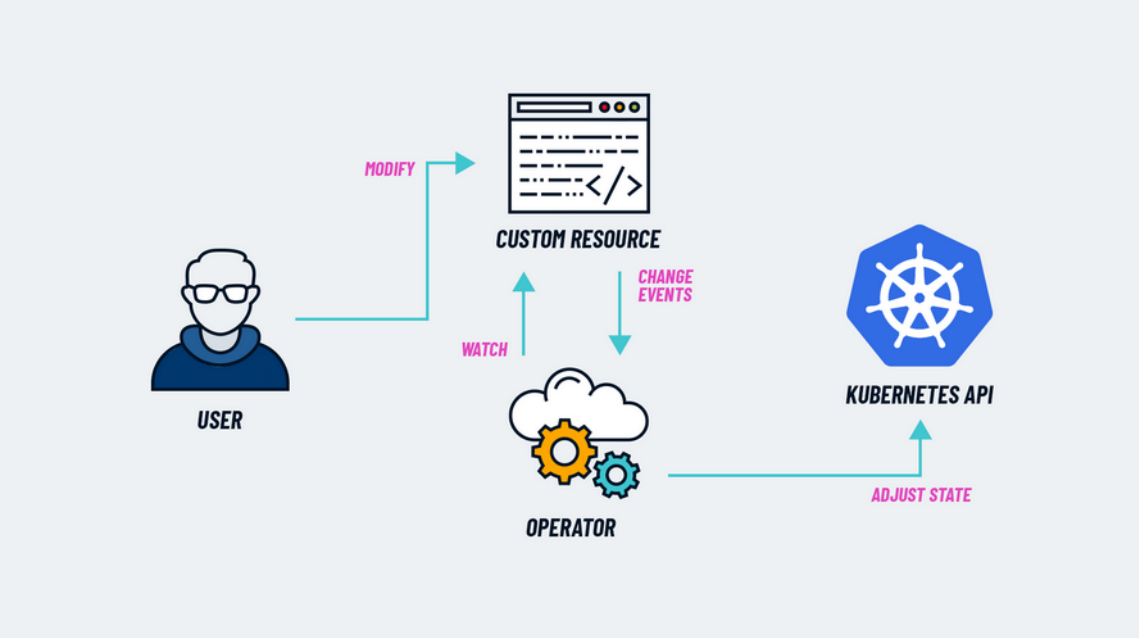
Kubernetes

mardi 7 février 2023 09:27

https://kubernetes.io/fr/

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

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

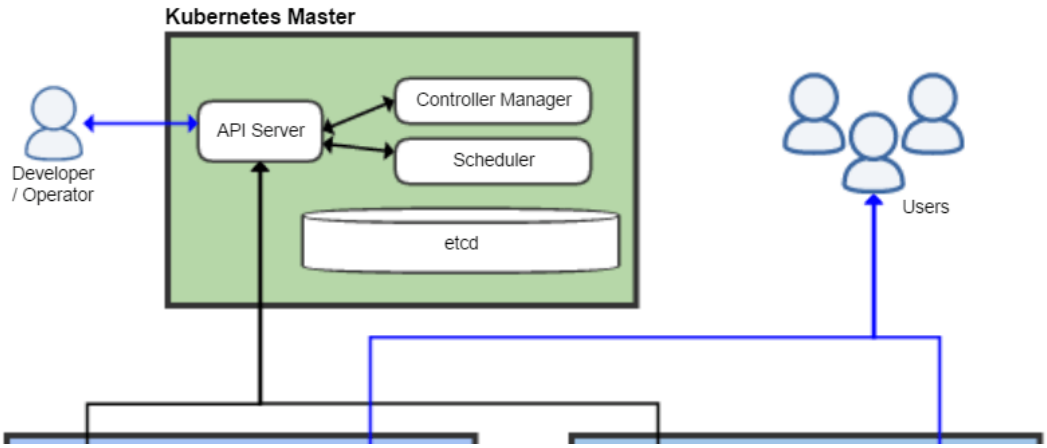


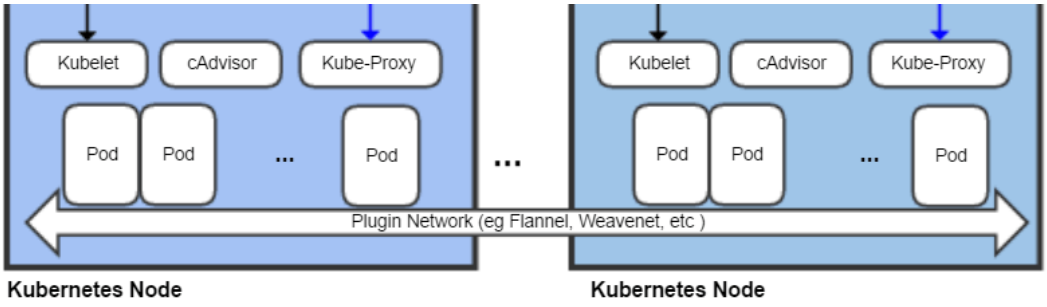
Kubernetes

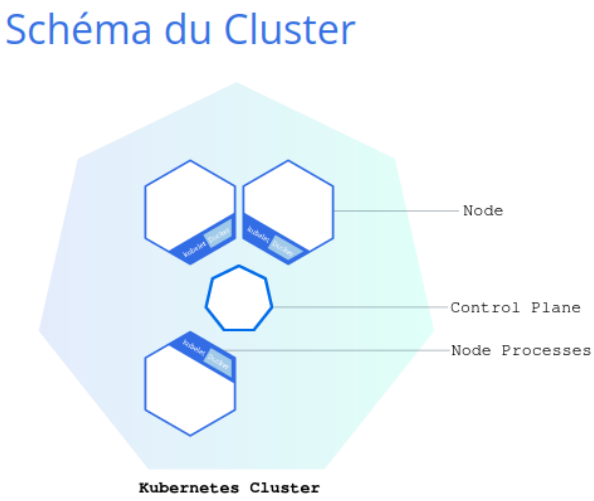
Article Discussion

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







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 ↵
```

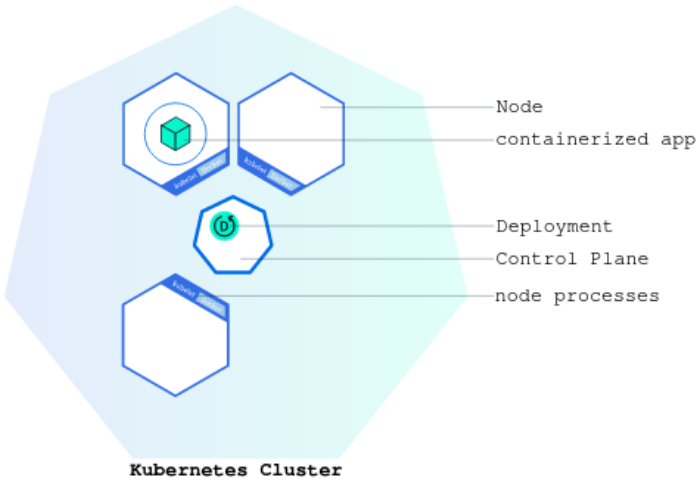
This command shows all nodes that can be used to host our applications. Now we have only one node, and we can see that its status is ready (it is ready to accept applications for deployment).

```
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

Step 1 of 3

kubectl basics

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 need to provide the deployment name and app image location (include the full repository url for

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/kubernetes-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"
}$
```

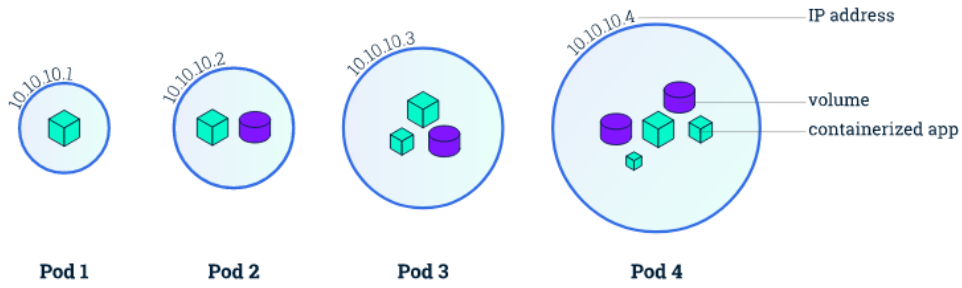
Viewing Pods and Nodes

Objectives

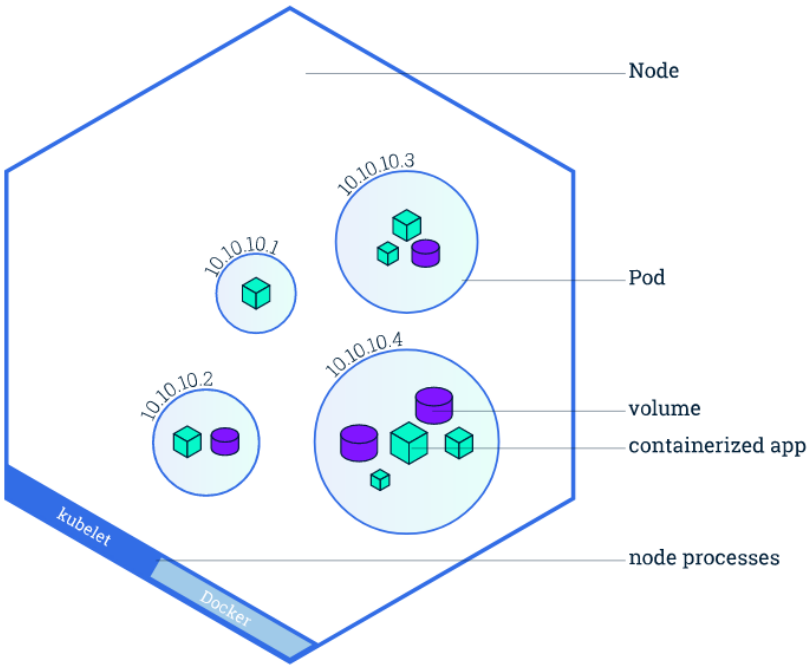
- Learn about Kubernetes Pods.
- Learn about Kubernetes Nodes

- 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:

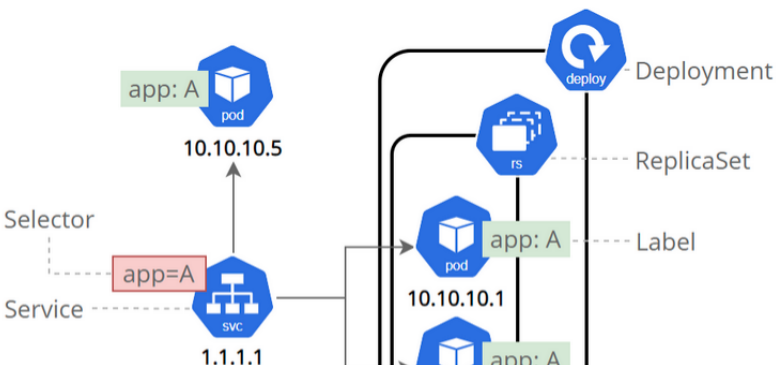
- *ClusterIP* (default) - Exposes the Service on an internal IP in the cluster. This type makes the Service only reachable from within the cluster.
- *NodePort* - Exposes the Service on the same port of each selected Node in the cluster using NAT. Makes a Service accessible from outside the cluster using `<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.

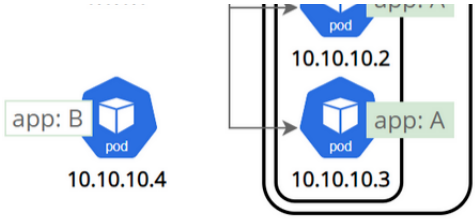
Services and Labels

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:

```
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

Step 5: 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

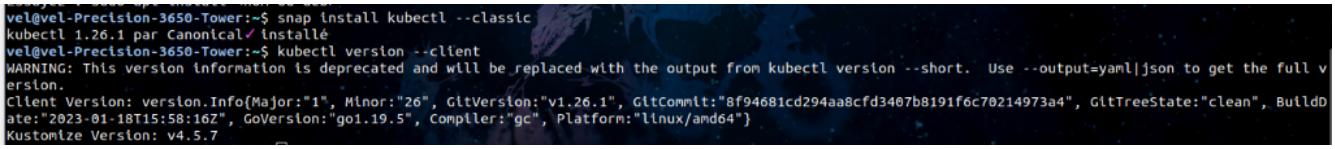
```
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

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

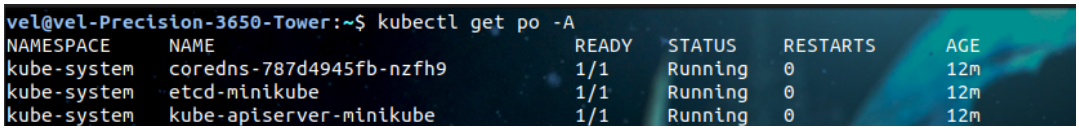
```
minikube kubectl -- get po -A
```

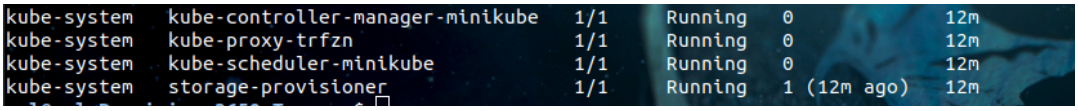
You can also make your life easier by adding the

```
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
```



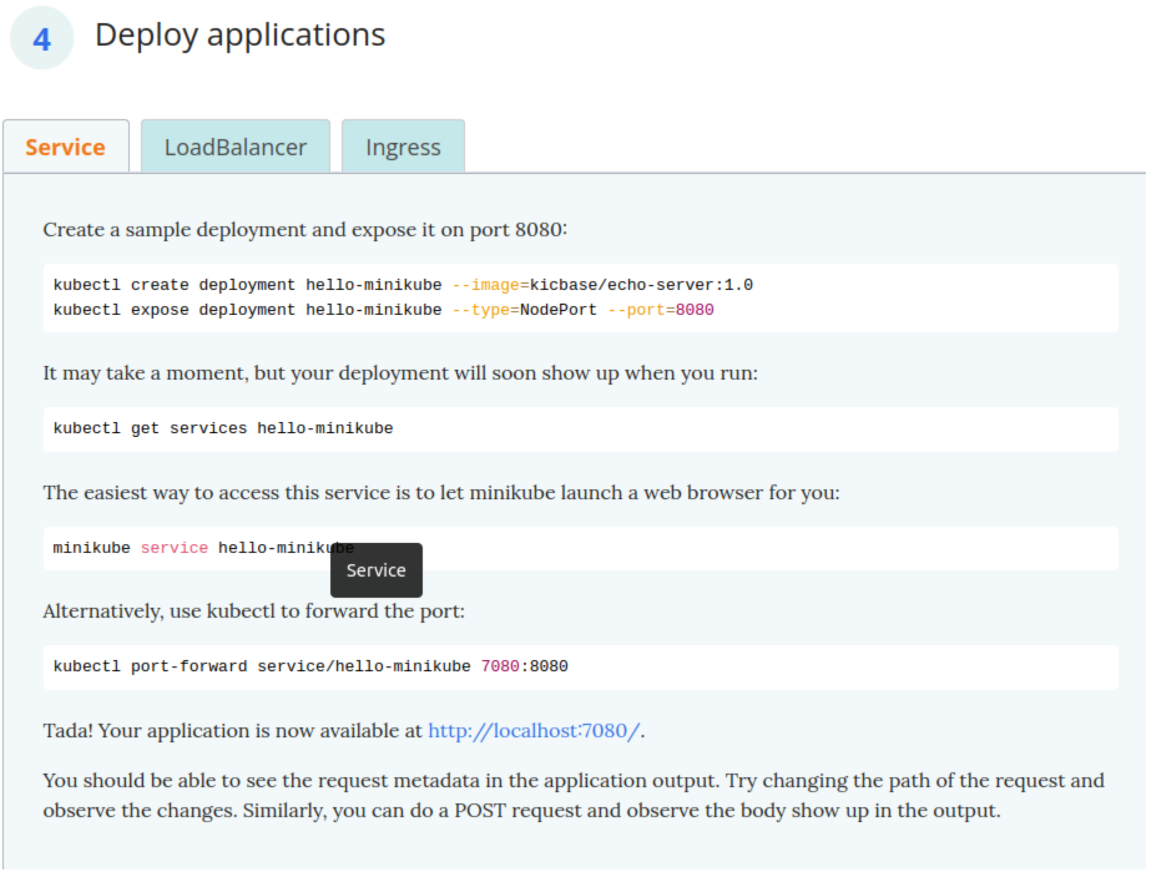


Pour ce tp, je ne ferais pas l'alias

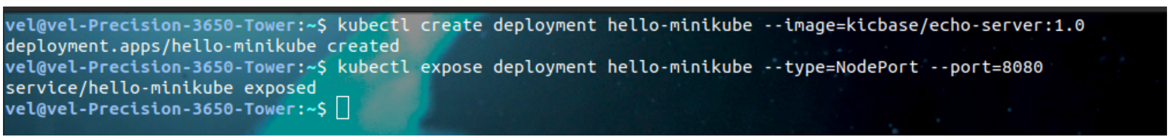
Déploiement du dashboard :

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

Déploiement application :



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



Vérification des services lancés

```
vel@vel-Precision-3650-Tower:~$ kubectl get services hello-minikube
NAME             TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
hello-minikube   NodePort   10.100.22.94    <none>        8080:31193/TCP   64s
```

```
vel@vel-Precision-3650-Tower:~$ minikube service hello-minikube
|-----------|----------------|-------------|---------------------------|
| NAMESPACE |      NAME      | TARGET PORT |            URL            |
|-----------|----------------|-------------|---------------------------|
| default   | hello-minikube |        8080 | http://192.168.49.2:31193 |
|-----------|----------------|-------------|---------------------------|
🎉  Ouverture du service default/hello-minikube dans le navigateur par défaut...
vel@vel-Precision-3650-Tower:~$ Gtk-Message: 12:23:42.857: Not loading module "atk-bridge": The functionality is provided by
 GTK natively. Please try to not load it.
```

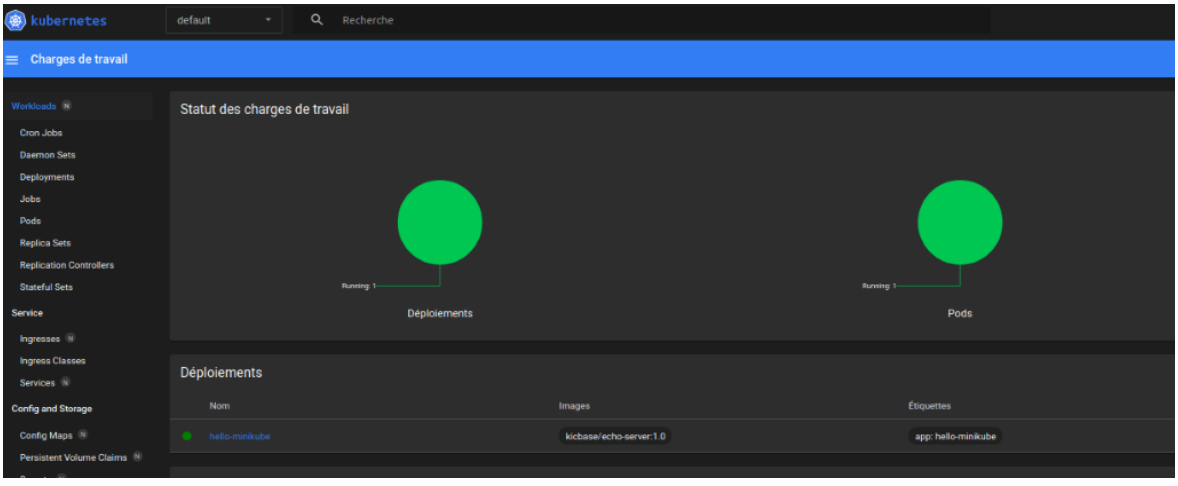
192.168.49.2:31193

```
Request served by hello-minikube-77b6f68484-68w67

HTTP/1.1 GET /

Host: 192.168.49.2:31193
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/109.0
```

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



On utilise maintenant kubectl pour ouvrir les ports

```
vel@vel-Precision-3650-Tower:~$ kubectl port-forward service/hello-minikube 7080:808
Forwarding from 127.0.0.1:7080 -> 8080
Forwarding from [::1]:7080 -> 8080
```

localhost:7080

```
Request served by hello-minikube-77b6f68484-68w67

HTTP/1.1 GET /

Host: localhost:7080
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
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
vel@vel-Precision-3650-Tower:~$
```

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
...
```

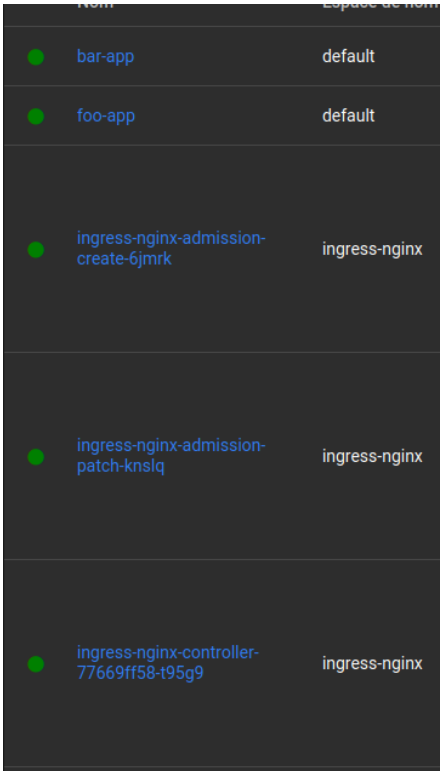
```
vel@vel-Precision-3650-Tower:~$ minikube addons enable ingress
💡  ingress est un addon maintenu par Kubernetes. Pour toute question, contactez minik
Vous pouvez consulter la liste des mainteneurs de minikube sur : https://github.com/ku
    ▪ Utilisation de l'image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20220
    ▪ 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
```

```
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

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.

```
vel@vel-Precision-3650-Tower:~$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | su
do apt-key add -
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (see apt-key(8)).
OK
vel@vel-Precision-3650-Tower:~$ cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
> deb https://apt.kubernetes.io/ kubernetes-xenial main
> EOF
deb https://apt.kubernetes.io/ kubernetes-xenial main
vel@vel-Precision-3650-Tower:~$ sudo apt-get update
Atteint :2 https://ppa.launchpadcontent.net/ansible/ansible/ubuntu jammy InRelease
Atteint :3 https://ppa.launchpadcontent.net/openjdk-r/ppa/ubuntu jammy InRelease
Réception de :1 https://packages.cloud.google.com/apt kubernetes-xenial InRelease [8 993 B]
Réception de :4 https://ngrok-agent.s3.amazonaws.com buster InRelease [20,3 kB]
Réception de :5 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 Packages [63,2
kB]
```

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.