Documentation Technique
K8S- challenge4 – task 3
Student: Valérie Hermans

màj le 06-02-2021

Auto-scaling Horyzontal Pods

Synthèse sur les techniques de configurations de l'API Dashboard de K8S

Expérimentation de la fonction de **Horizontal Pod Autoscaler (HPA)**
et compréhension de son fonctionnement avec *Metrics Server* et *Helm3*

Introduction aux **H**oryzontal **P**ods **A**utoscallers

Horyzontal parce que l'on va créer X pods pour répondre aux requêtes et à la charge demandées.

HPA est complémentaire avec le CA (**C**luster **A**utoscaller) qui permet d'instancier de nouvelles machines et de les ajouter à notre cluster pour pouvoir répondre à la charge au cas où les ressources du cluster seraient épuisées.

L'objectif:

- Multiplier le nombre de pods en fonction du besoin.

  En fonction de la consommation de ressources, nous allons donc avoir un nombre **min**imum et un nombre **max**imum de pods

- Nous aurons aussi un seuil de déclenchement, qui va faire que l'on instancie un nouveau pod, ou que, potentiellement, on en diminue le nombre.

On utilise le hpa dans le cadre de déploiements et non dans le cadre de replicasets.
L'objctif d'un déploiement va être d'avoir un pod, sur lequel on applique des réplicasets et, d'y ajouter une couche de hpa.

Attention, pas de HPA sans métriques, car nous avons besoins de vérifier la consommation des ressources et pouvoir forcément scaller.

Introduction à Helm3

Helm est un outil très pratique qui a rejoint la **C**loud **N**ative **C**omputing **F**oundation en 2018.

C'est un gestionnaire de paquets open-source dédié à Kubernetes.

- Il permet de fournir, de partager et d'utiliser des logiciels conçus pour Kubernetes.
- Il permet de simplifier la génération des manifest yaml ainsi que de faire du versionning (update, rollback etc.)
- Il facilite, par le biais d'un dépôt distant, le téléchargement d'autres dépôts, suivant une stack donnée.

Une charte est un stack technique, qui comprends, un ensemble de fichiers manifests, qui permettent de déployer une stack stable et cohérente, non sous forme de binaire, mais bien sous forme de fichiers de configurations.

Vous pouvez retrouver ces chartes sur leur site à l'adresse: https://artifacthub.io/

Introduction à Metrics Server

Metrics Server est un dépôt officiel. Son rôle est de collecter des métriques à l'échelle du cluster et les fournir à l'API via Kubelet, il nous offre une vue sur des ressources, et principalement celles du CPU et de la mémoire.

Nous les récupérons les métriques sur l'utilisation des ressources en cours, via cette API contenant un tableau de bord mis à disposition.

Attention, Il ne stocke pas de données, donc vous ne pouvez pas l'utiliser pour récupérer des valeurs et/ou prédire des tendances.

S'il existe une charte chez helm, utilisez-la car il s'agit d'un gage de soutiens à la communauté Kubernetes. Autrement, il vaut mieux utiliser une alternative fiable.

Attention cependant à la version utilisée: Metrics-server n'est plus compatible avec la dernière version de Kubernetes (version 1.18+)

Je l'ai installé et bien que j'ai eu un message d'erreur, il s'est quand même installé et à fait son travail.

Pré-requis:

Il nous faut un cluster qui fonctionne.

Vérifier cela avec `kubectl get all` dans la console

Il nous faut donc, créer, au minimum un déploiement et un service:

Manuellement::

- Créez un dossier, nommé par exemple *challenge-4/task3*
À l'intérieur de celui-ci, faites un click-droit: Ouvrir avec code

Créer un nouveau fichier appellé `challenge4-task03-horizontal-pod-autoscaler.yaml`

En console:

```
mkdir challenge-4      # on crée le dossier
cd challenge-4         # on se déplace dedans

#on crée le fichier
touch  challenge4-task03-horizontal-pod-autoscaler.yaml
code                   #on l'ouvre avec VScode

//revenir au dossier challenge-4 pour effectuer le reste des commandes
```

Copier-coller le script ci-dessous à l'intérieur du fichier yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
spec:
  selector:
    matchLabels:
      run: php-apache
  replicas: 1
  template:
    metadata:
      labels:
        run: php-apache
    spec:
      containers:
      - name: php-apache
        image: k8s.gcr.io/hpa-example
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 500m
          requests:
            cpu: 200m
---
apiVersion: v1
kind: Service
metadata:
  name: php-apache
  labels:
    run: php-apache
spec:
  ports:
  - port: 80
  selector:
    run: php-apache
```

→ Lancer ensuite la commande Apply -f pour lancer et déployer le script manifest:

```
PS D:\challenge-4> kubectl apply -f challenge4-task03-horizontal-pod-
autoscaler.yaml
```

→ On vérifie la présence du pod crée:

```
PS D:\challenge-4> kubectl get po
NAME                          READY   STATUS              RESTARTS   AGE
php-apache-d4cf67d68-xl5ls    0/1     ContainerCreating   0          14s
```

→ On 'saute' dans ce pod en cours, pour effectuer l'installation de helm:

```
PS D:\challenge-4> kubectl exec -it php-apache-d4cf67d68-xl5ls -- /bin/bash
```

→ On récupère le curl de helm - Sur leur site: Introduction / Installing helm, repérer et copier le **curl :**

```
root@php-apache-d4cf67d68-xl5ls:/var/www/html# curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
```

→ On donne le droit d'execution au fichier get_helm:

```
root@php-apache-d4cf67d68-xl5ls:/var/www/html# chmod 700 get_helm.sh
```

→ On le télécharge sur l'hôte:

```
root@php-apache-d4cf67d68-xl5ls:/var/www/html#  ./get_helm.sh
Downloading https://get.helm.sh/helm-v3.5.1-linux-amd64.tar.gz
Verifying checksum... Done.
Preparing to install helm into /usr/local/bin
helm installed into /usr/local/bin/helm
```

→ On affiche la liste en local:

```
root@php-apache-d4cf67d68-xl5ls:/var/www/html# helm list
NAME            NAMESPACE       REVISION        UPDATED
STATUS   CHART                   APP VERSION
my-release      default         1               2021-02-02 17:44:58.1286186
+0100 +0100
deployed metrics-server-5.5.0    0.4.1
```

→ On recherche un dépôt stable de metrics-server sur le hub:

```
root@php-apache-d4cf67d68-xl5ls:/var/www/html# helm repo add stable
https://charts.helm.sh/stable
"stable" has been added to your repositories
```

→ On mets cette version à jour:

```
root@php-apache-d4cf67d68-xl5ls:/var/www/html# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "stable" chart repository
Update Complete. ⎈Happy Helming!⎈
```

→ On liste le repo installé localement:

```
root@php-apache-d4cf67d68-xl5ls:/var/www/html# helm repo list
NAME    URL
stable  https://charts.helm.sh/stable
```

→ On copie le fichier de values en un fichier local appelé my-metrics.values:

```
root@php-apache-d4cf67d68-xl5ls:/var/www/html# helm show values
stable/metrics-server  > my-metrics.values
```

Nous allons devoir modifier ce fichier, mais avant il faut mettre le système à jour et installer vim:

```
apt-get update
apt-get install vim
...
```

→ Modification du fichier my-metrics.values:

```
root@php-apache-d4cf67d68-xl5ls:/var/www/html# vi my-metrics.values
```

Taper "i" pour entrer dans le mode Insert de Vim

Adaptez les lignes hostNetwork et args telles que ci-dessous:

```
hostNetwork
    change enabled : true

args:
- -- kubelet-insecure-tls
```

une fois que c'est fait, touche **Escape** + **:wq** pour quitter et sauvegarder le fichier

→ On installe à présent notre fichier values modifié:

```
root@php-apache-d4cf67d68-xl5ls:/var/www/html# helm install my-metrics-
server stable/metrics-server --values my-metrics.values

WARNING: This chart is deprecated
W0202 20:41:43.557189     252 warnings.go:70] apiregistration.k8s.io/v1beta1
APIService is deprecated in v1.19+, unavailable in v1.22+; use
apiregistration.k8s.io/v1 APIService
W0202 20:41:43.664452     252 warnings.go:70] apiregistration.k8s.io/v1beta1
APIService is deprecated in v1.19+, unavailable in v1.22+; use
apiregistration.k8s.io/v1 APIService
NAME: my-metrics-server
LAST DEPLOYED: Tue Feb  2 20:41:43 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
The metric server has been deployed.

In a few minutes you should be able to list metrics using the following
command:

  kubectl get --raw "/apis/metrics.k8s.io/v1beta1/nodes"
```

→ On peut à présent quitter le pod:

```
root@php-apache-d4cf67d68-xl5ls:/var/www/html# exit
exit
command terminated with exit code 127
```

→ On check à présent les noeuds:

```
PS D:\challenge-4> kubectl top nodes
NAME             CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
docker-desktop   573m         7%     1963Mi          16%
```

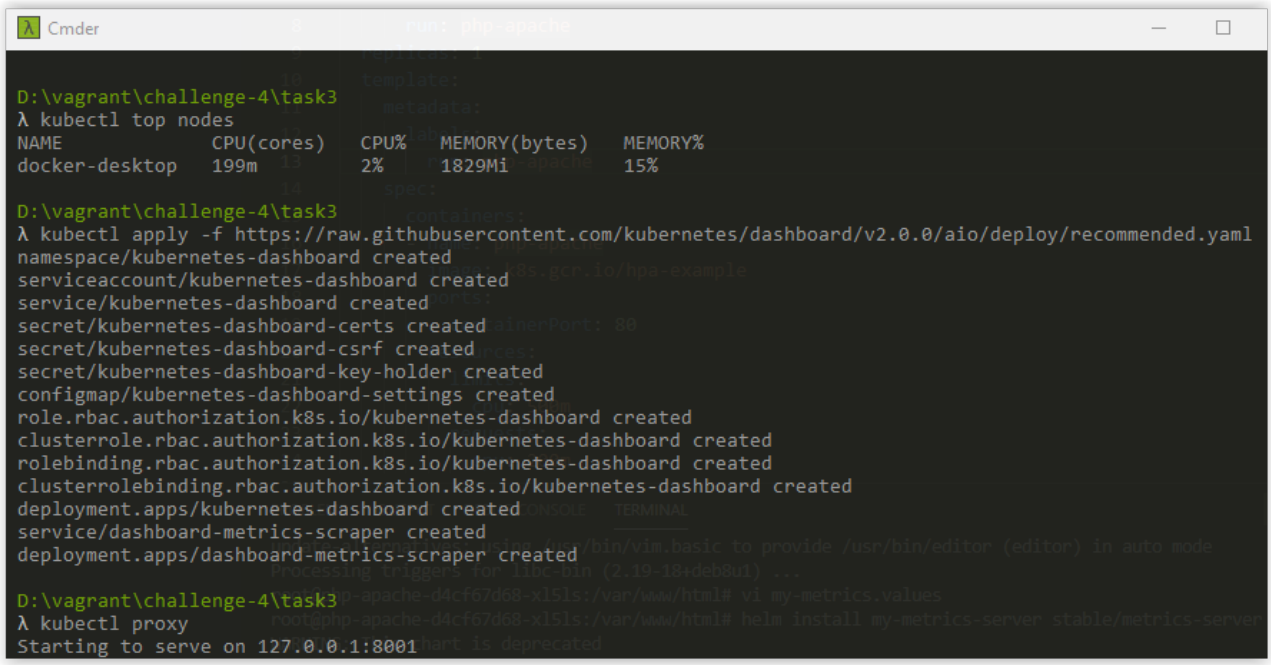
En paralèlle à cette console, nous devons utiliser un autre terminal pour ouvrir l'API et accéder au tableau de bords de métrics:

→ On déploie ensuite le dashboard du metrics-server :

```
valer@DESKTOP-L2RC63V MINGW64 /d/challenge-4/
$ kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0/aio/deploy/rec
ommended.yaml
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
deployment.apps/dashboard-metrics-scraper created
```

→ On lance un proxy pour établir une connexion avec notre tableau de bord:

```
valer@DESKTOP-L2RC63V MINGW64 /d/challenge-4/
$ kubectl proxy
```



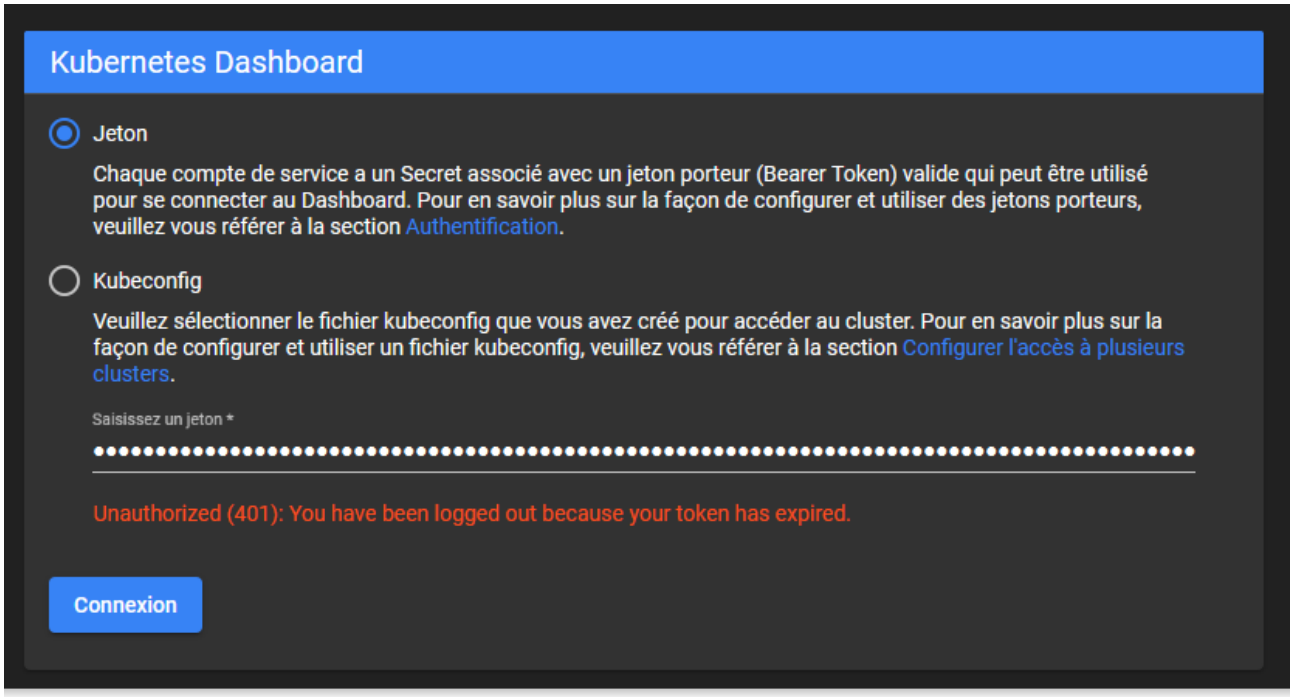
!! Attention, ne fermez pas cette console autrement votre connection se fermera elle aussi !!

→ Rendez-vous à votre tableau de bord:

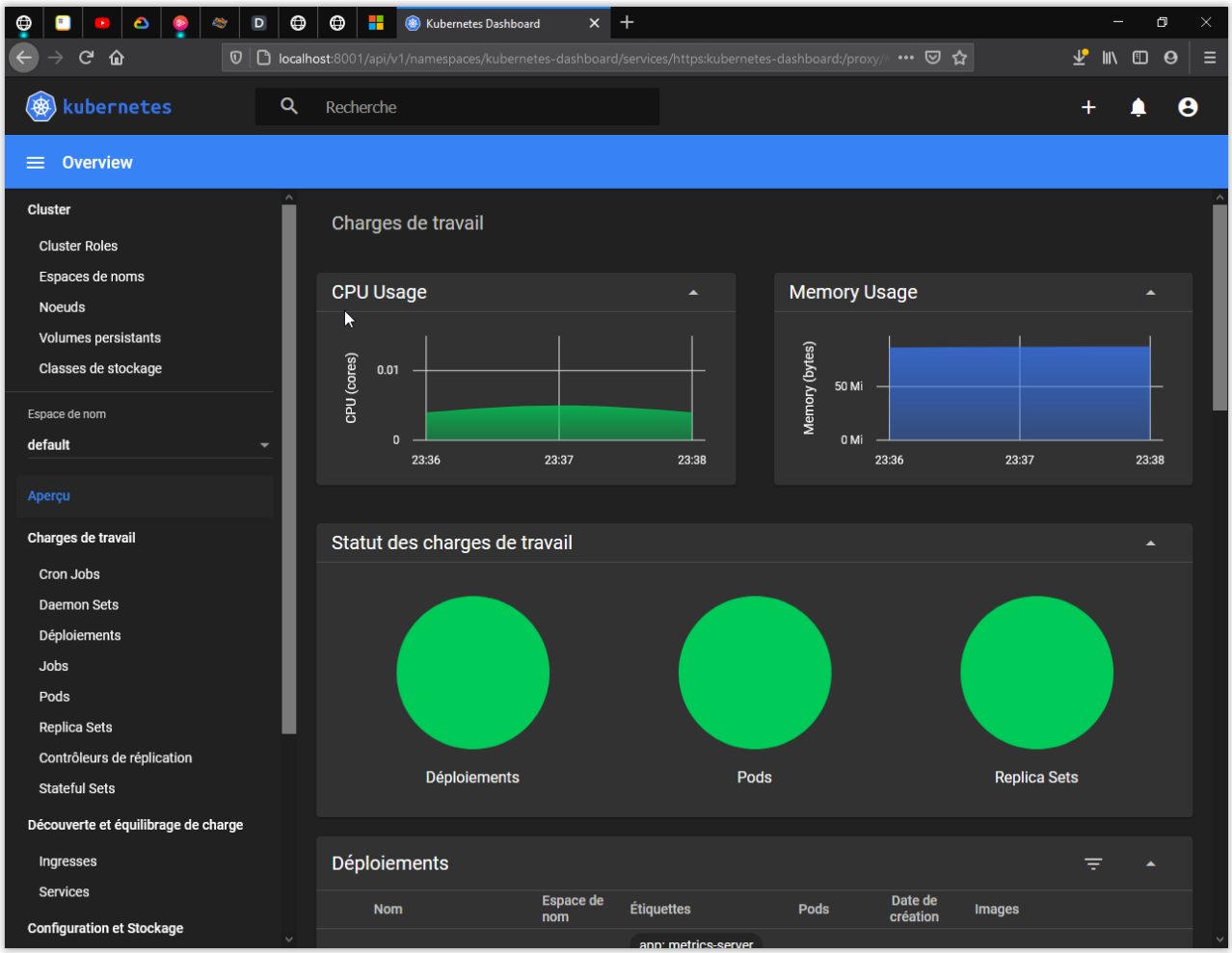
http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/

Il nous demande un jeton de connexion, dans un terminal, celui de Vscode qui est à présent disponible, entrez la commande pour générer le jeton secret:

```
PS D:\challenge-4> kubectl -n kubernetes-dashboard describe secret default
Name:         default-token-t5wn2
Namespace:    kubernetes-dashboard
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: default
              kubernetes.io/service-account.uid: 2875102a-d299-4abb-a4bf-
3098649f5e20

Type:  kubernetes.io/service-account-token

Data
====
namespace:  20 bytes
eyJhbGciOiJSUzI1NiIsImtpZCI6ImVmVnUtUEptazM3RjNLSnRNTDhXNEFCQi16Qng3NGxFZmJl
OW15UVU4ZkEifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlc
y5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIsImt1Y
mVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJkZWZhdWx0LXRva2VuLXQ1d
24yIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZSI6I
mRlZmF1bHQiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51a
WQiOiIyODc1MTAyYS1kMjk5LTRhYmItYTRiZi0zMDk4NjQ5ZjVlMjAiLCJzdWIiOiJzeXN0ZW06c
2VydmljZWFjY291bnQ6a3ViZXJuZXRlcy1kYXNoYm9hcmQ6ZGVmYXVsdCJ9.FDQkysa5dB-
FDXhHaNG4GIM5dMrC6uoIuNL250Jm_Sqh14DsKns16Ah80_FsilKFefEp5eEty1cznzh2vmYoQ2v
rjhnGQLHeeb32OxVszXC4nSFvvQP_QFJaNtlPbPP7kmogb2hhqMzrk6qwaFrKf7q0Khoh7lIUXmI
-hAtY454w2jdoFK0QyCOAc2xNCqRe0aNZnlLp9JNB4C7T-
_axxxsI7tI9miQW846T_9Sda2jlQVHEDjr0Q7qEJmFQy5CgP20kfNhBqVynX3zYgWaVh59Qw_gOM
MATIPLVXi4gO7NL5VOD5Z3kXZSNW_N6bYKqylFjTh0ALVshdVxGmAKFRQ
ca.crt:     1066 bytes

URL du dashboard:
Kubectl will make Dashboard available at
http://localhost:8001/api/v1/namespaces/kubernetes-
dashboard/services/https:kubernetes-dashboard:/proxy/
```



→ Copier le token et entrez-le dans le champs approprié, cliquez dedans: **ctrl+a** et **ctrl + v** pour coller le token dedans, cliquez ensuite sur **Connexion** pour entrer dans le dashboard.



Task 3 Instruction: *Create Horizontal Pod Autoscaler*:

Now that the server is running, we will create the autoscaler using kubectl autoscale.

The following command will create a Horizontal Pod Autoscaler that maintains between 1 and 10 replicas of the Pods controlled by the php-apache deployment we created in the first step of these instructions.

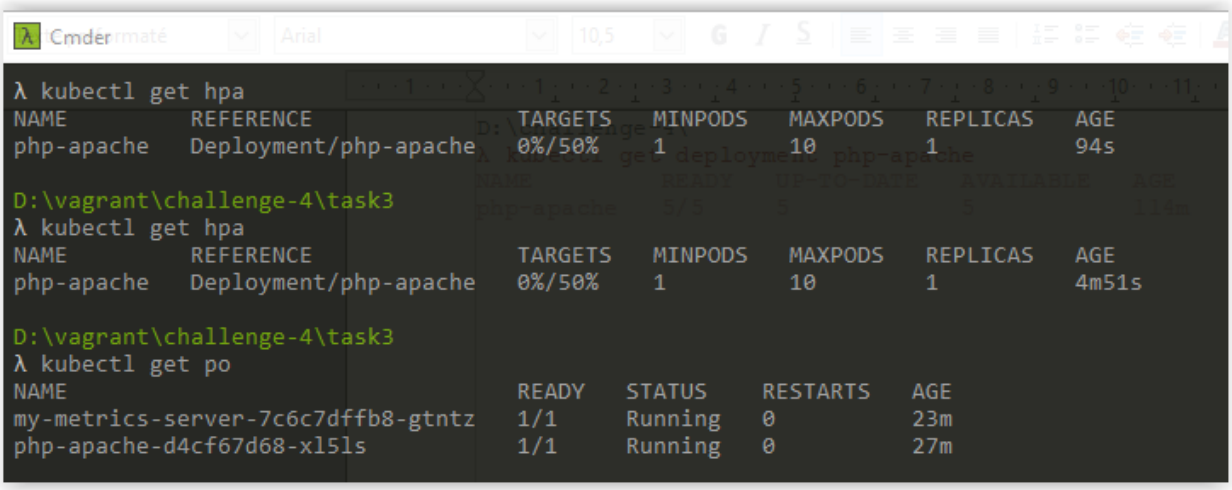
Roughly speaking, HPA will increase and decrease the number of replicas (via the deployment) to maintain an average CPU utilization across all Pods of 50% (since each pod requests 200 milli-cores by kubectl run), this means average CPU usage of 100 milli-cores).

→ On lance un déployement définissant le **min** et le **max** de pods suivant les ressources cpu/mémoire :

```
PS D:\challenge-4> kubectl autoscale deployment php-apache --cpu-percent=50
--min=1 --max=10
horizontalpodautoscaler.autoscaling/php-apache autoscaled
```

→ On vérifie notre pod, pour l'instant, rien ne bouge, tout est encore à la normale (CPU) :

```
PS D:\vagrant\challenge-4> kubectl get hpa
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS
AGE
php-apache   Deployment/php-apache   0%/50%    1         10        1
3m17s
```



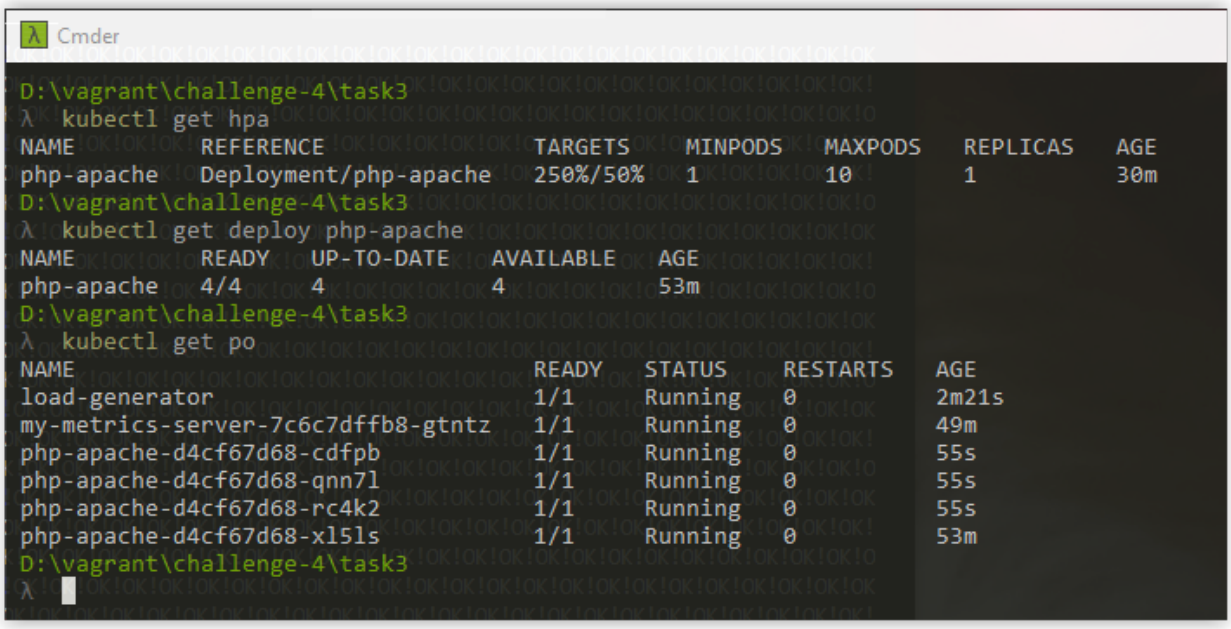
Increase load:
Now, we will see how the autoscaler reacts to increased load. We will start a container, and send an infinite loop of queries to the php-apache service (please run it in a different terminal):

```
PS D:\challenge-4> kubectl run -i --tty load-generator --rm --image=busybox
--restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-
apache; done"
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK! ......
```

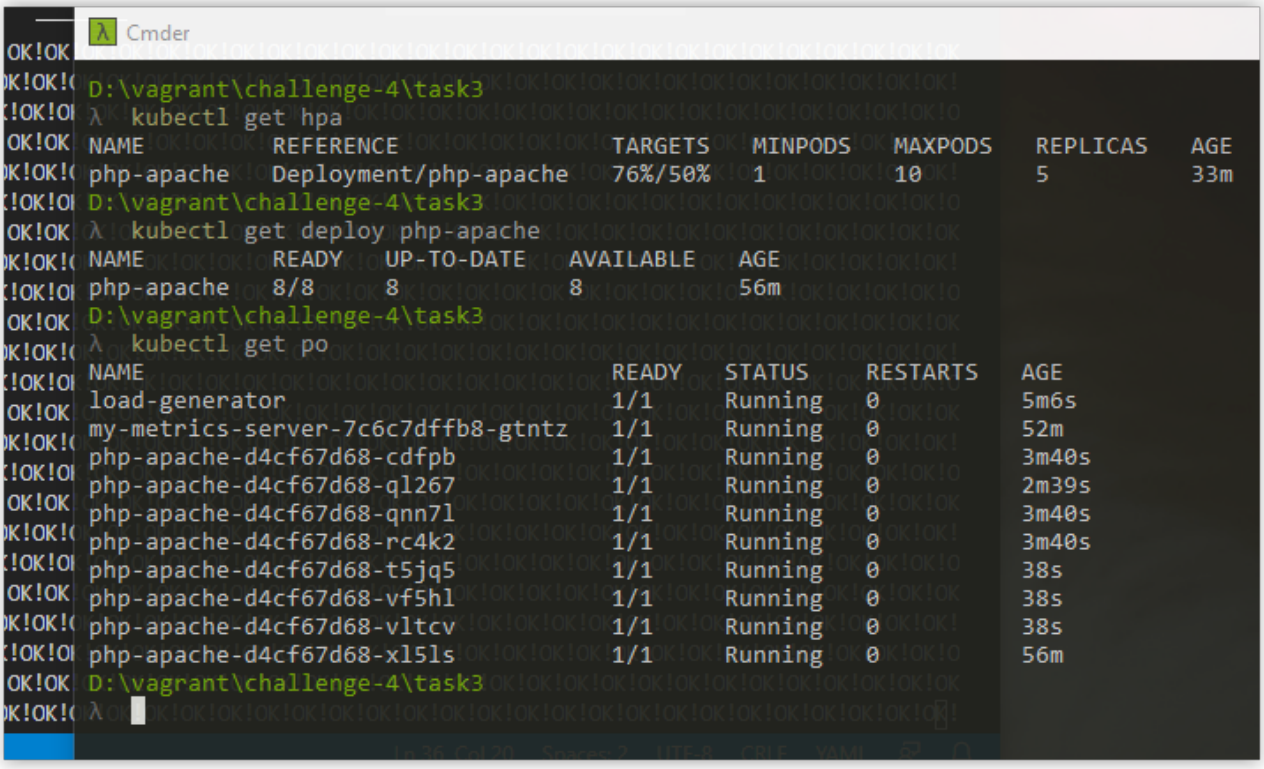
Pendant que la boucle tourne, il nous faut un autre terminal pour voir le déroulement des ressources via la cible (target) du cpu:

→ On vérifie, simultanément les pods qui se construisent et les valeurs du HPA qui grimpe (de manière fulgurante) jusqu'a minimum 250 %

```
D:\challenge-4\
λ kubectl get deployment php-apache
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
php-apache   1/1     1            1           113m

D:\challenge-4\
λ kubectl get deployment php-apache
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
php-apache   4/4     4            4           113m

D:\challenge-4\
λ kubectl get hpa
NAME         REFERENCE                  TARGETS    MINPODS   MAXPODS   REPLICAS
AGE
php-apache   Deployment/php-apache   250%/50%   1         10        1
92m

D:\challenge-4\
λ kubectl get deployment php-apache
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
php-apache   5/5     5            5           114m
```

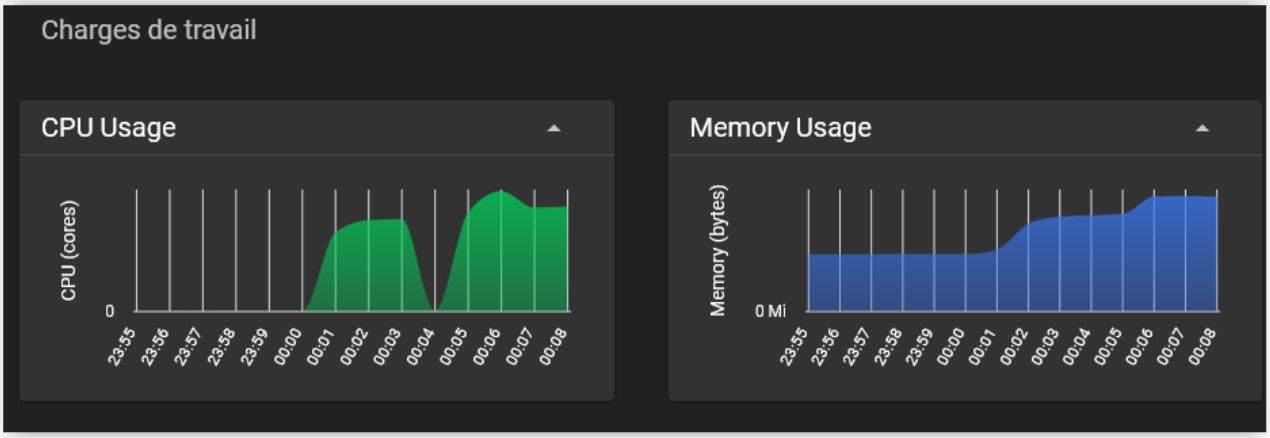


Le niveau de CPU peut monter jusqu'a + de 250% de CPU sur la moitié des capacités de mémoire (50%) et ce, pour créer une charge de maximum 10pods.

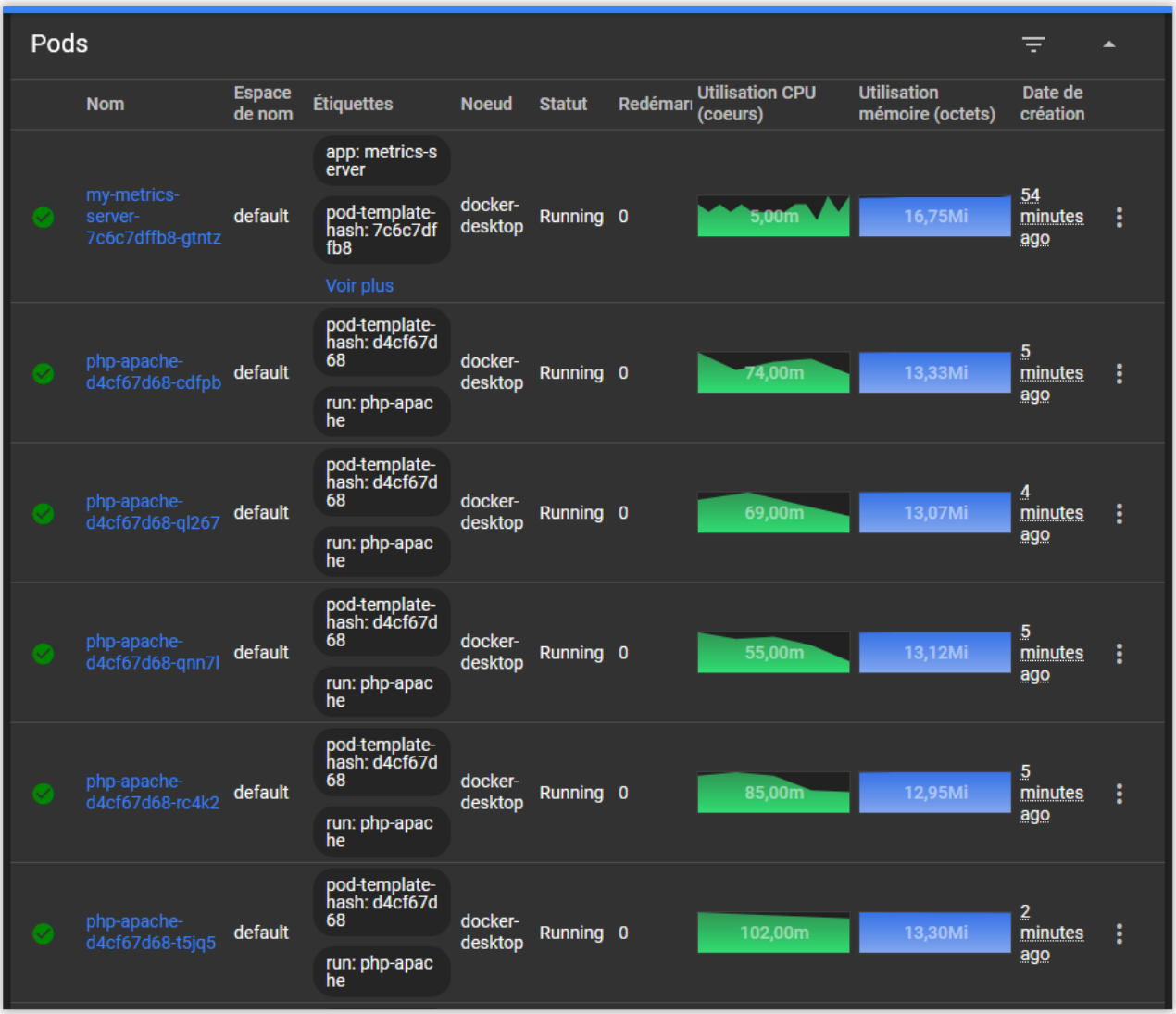


Avec `kubectl get po`, vous voyez les pods se créer.

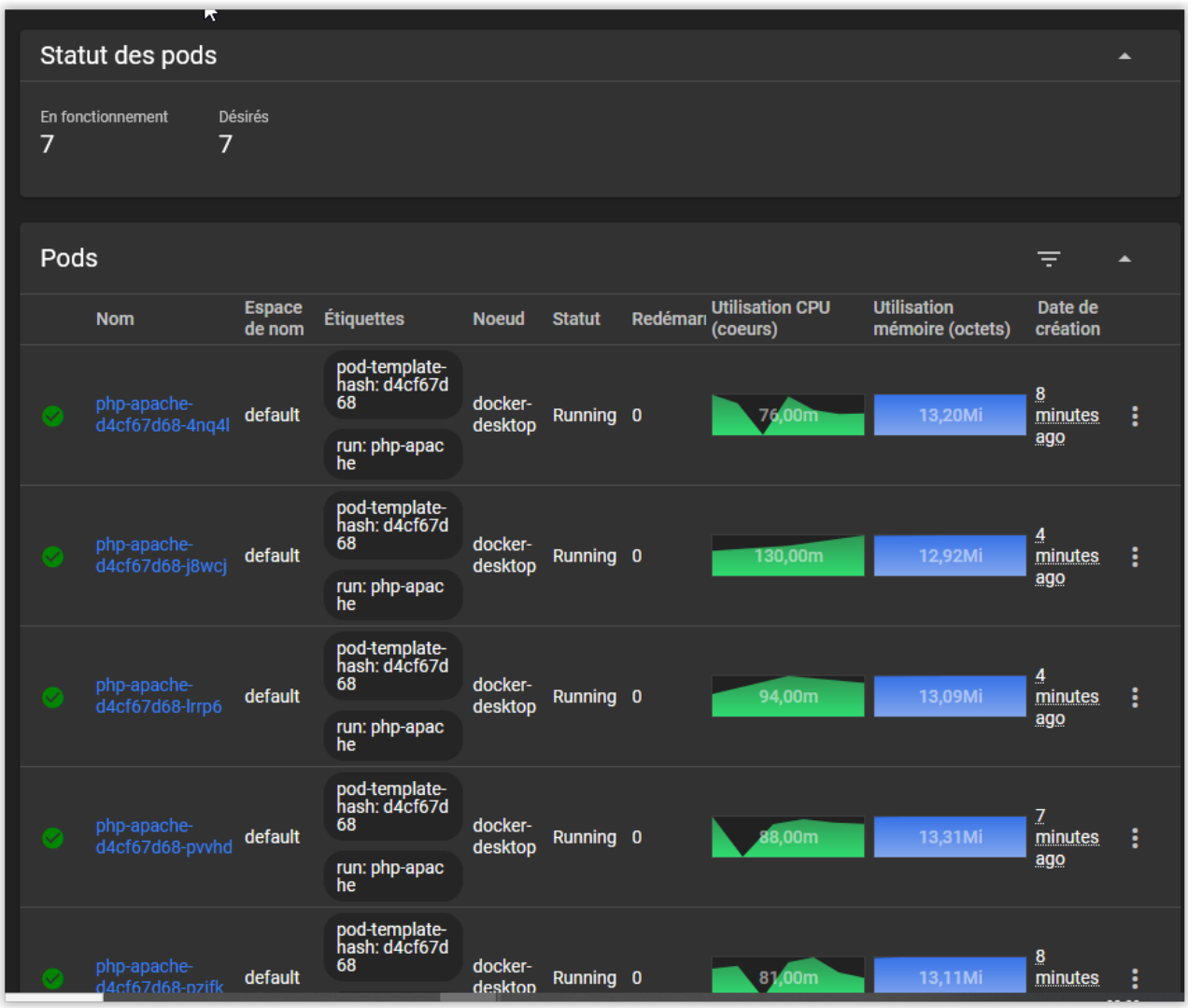
Dans le tableau de bord, on voit la charge de travail qui se mets tout doucement en route...



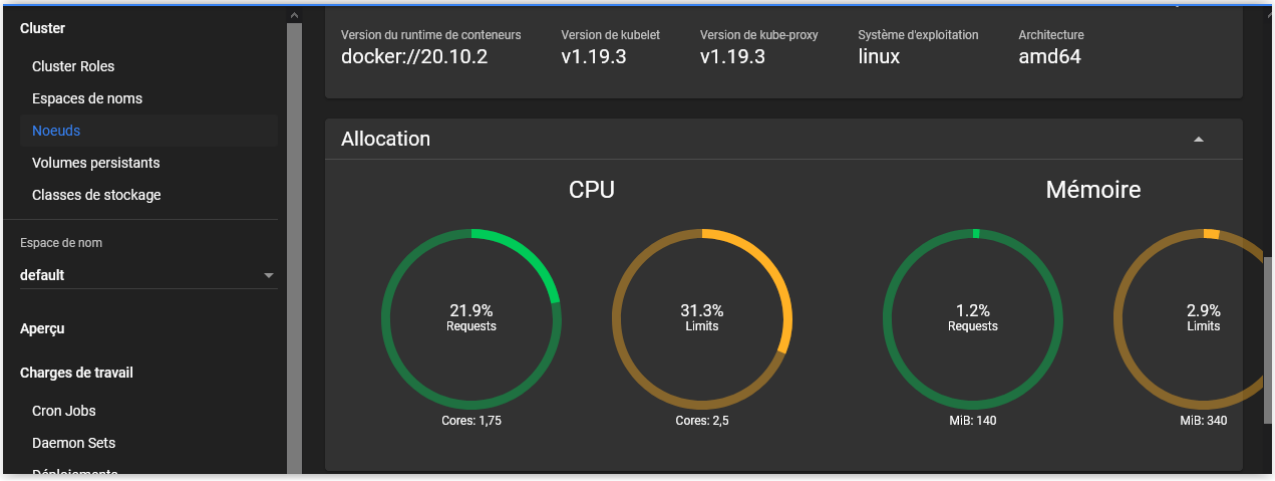
Les pics des pods (de déployements) sont aussi flagrants:



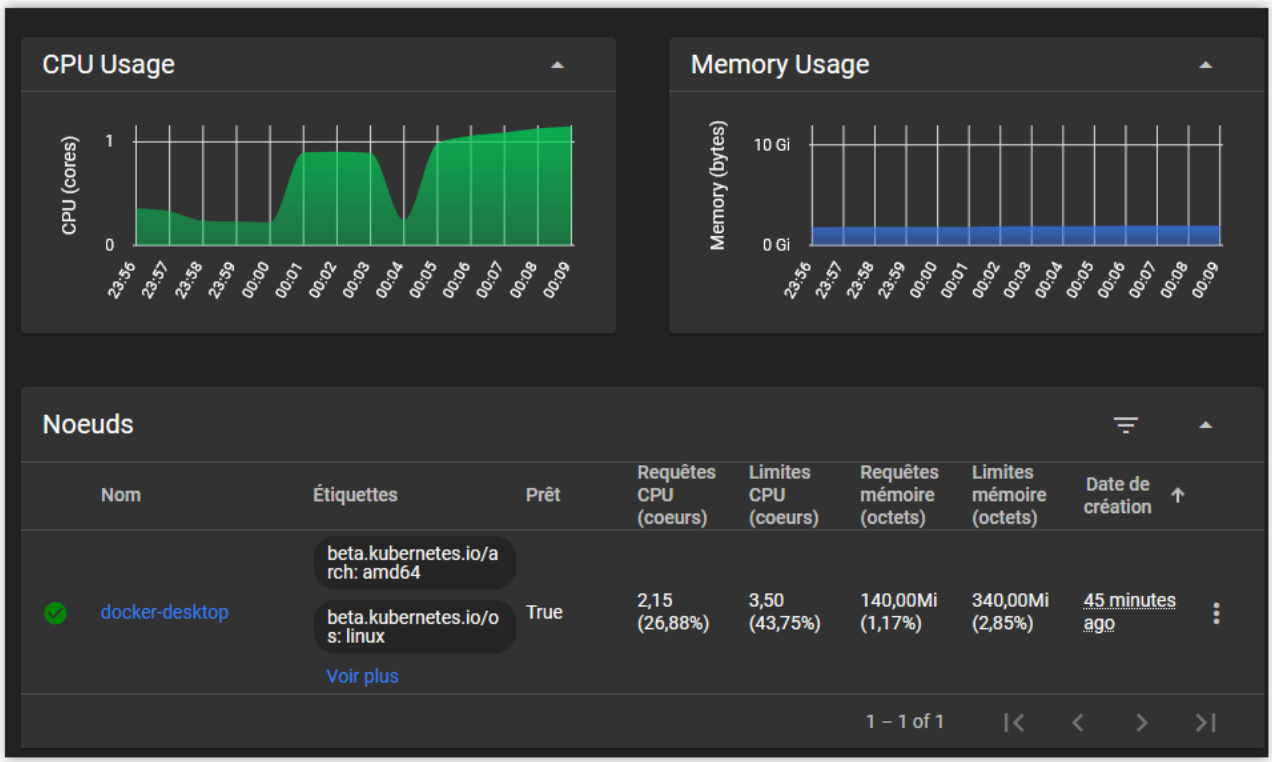
Ainsi que la vue sur les status des pods (réplicas crées):



Aperçu des ressouces sur le noeud Docker-desktop:



L'État du CPU et de la mémoire du noeud Docker Desktop:



→ Retour dans notre boucle, que l'on peut stopper avec **ctrl** + **c**

```
PS D:\challenge-4>kubectl run -i --tty load-generator --rm --image=busybox
--restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-
apache; done"
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!//..
OK!OK!OK!OK!OK!
^C
```

Dans notre console cmder, le retour à la normale se refait doucement:

```
D:\challenge-4\
λ kubectl get hpa
NAME         REFERENCE                  TARGETS    MINPODS   MAXPODS   REPLICAS
AGE
php-apache   Deployment/php-apache   0%/50%     1         10        1
102m
```

→ Clean-up et suppression des ressources:

```
PS D:\challenge-4> kubectl delete -f challenge4-task03-horizontal-pod-autoscaler.yaml
deployment.apps "php-apache" deleted
service "php-apache" deleted

PS D:\challenge-4> kubectl get  hpa
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   0%/50%    1         10        1          106m

PS D:\challenge-4> kubectl delete  hpa php-apache
horizontalpodautoscaler.autoscaling "php-apache" deleted

PS D:\challenge-4> kubectl get deploy
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
my-metrics-server   1/1     1            1           120m

PS D:\challenge-4> kubectl delete deploy my-metrics-server
deployment.apps "my-metrics-server" deleted
```

& Suppression des derniers pods en process dans doker desktop:

```
PS D:\challenge-4> kubectl get svc
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes          ClusterIP   10.96.0.1       <none>        443/TCP   140m
my-metrics-server   ClusterIP   10.100.212.29   <none>        443/TCP   122m

PS D:\challenge-4> kubectl delete svc my-metrics-server
service "my-metrics-server" deleted

PS D:\challenge-4> kubectl get ns
NAME                   STATUS   AGE
default                Active   144m
kube-node-lease        Active   144m
kube-public            Active   144m
kube-system            Active   144m
kubernetes-dashboard   Active   120m

PS D:\challenge-4> kubectl delete  ns kubernetes-dashboard
namespace "kubernetes-dashboard" deleted


#helm et my-metrics server on généré des secrets

PS D:\challenge-4> kubectl get secrets
NAME                                  TYPE
DATA   AGE
default-token-7dmqr                   kubernetes.io/service-account-
token   3      168m
my-metrics-server-token-cq45v         kubernetes.io/service-account-
token   3      150m
sh.helm.release.v1.my-metrics-server.v1   helm.sh/release.v1
1      150m

PS D:\challenge-4> kubectl delete secret my-metrics-server-token-cq45v
secret "my-metrics-server-token-cq45v" deleted
```

```
PS D:\challenge-4> kubectl delete secret sh.helm.release.v1.my-metrics-server.v1
secret "sh.helm.release.v1.my-metrics-server.v1" deleted
```

Informations complémentaires

Au sujet de helm:

Liste des commandes helm:

```
helm list
helm repo list              #liste les dépots
helm search hub nomCharte    #cherche des charts sur le hub
helm search repo nomCharte   #cherche des dépots avec mot clé dans charts
```

Des fichiers sont générés lorsque l'on fait un `helm create`

```
nomCharte/
      Chart.yaml        #description du chart
      values.yaml       #Variables injectées et disponibles pour le template
      templates /       #templates de manifests
      charts /          #sous-charts (optionnel: imbrications de Charts)
      helmignore        #ignore des fichiers pour le dépot
```

Attention, helm peut masquer l'utilisation de volumes persistants donc il faut toujours regarder ce que l'on installe avec helm et si il y a de la persistance de données etc.

Au sujet de Metrics-server:

Pour trouver le repository de metrics-server:

Entrez "*incubateur kubernetes/metrics server*" dans Google pour le retrouver
ou directement depuis cette url:

https://github.com/kubernetes-sigs/metrics-server pour pouvoir le cloner:

```
git clone https://github.com/kubernetes-sigs/metrics-server.git
```

dans votre répertoire de travail.

Il n'est pas nécessaire dans cet exemple de cloner le repository mais je sais que d'autres le clone, modifie le fichier values et le charge directement dans le cluster

Au sujet de la ressource hpa:

La resource hpa s'utilise avec :

```
apiVersion: autoscaling/v1
kind: HoryzontalPodAutoscaller
```

Pour voir toutes les apiVersions et leurs kind qui en dépendent: `kubectl get apiVersion`