Conteneurisation

Projet Conteneurisation et Orchestration

Paris Ynov Campus

M. LECLERC

DENIS Lucas

KHADIR Selim

MARIVINT Xavier

Introduction

Dans le cadre de ce projet de conteneurisation et orchestration, nous avons développé une solution permettant le déploiement, la maintenance et la mise à jour de différents micro-services REST et applications web dans un environnement conteneurisé, avec un orchestrateur Kubernetes afin d'assurer la haute disponibilité des applications.

Dans un premier temps, nous avons créé les images nécessaires pour l'application en utilisant plusieurs outils. Ensuite, nous avons créé un docker Hub pour stocker ces images, et nous avons également effectué un push de ces images dans le docker Hub.

Par la suite, nous avons rédigé les fichiers YAML pour Kubernetes afin de déployer l'application. Nous avons également mis en place la solution ELK pour permettre le suivi des logs. En plus de cela, nous avons créé une solution de suivi de santé pour garantir la disponibilité de l'application.

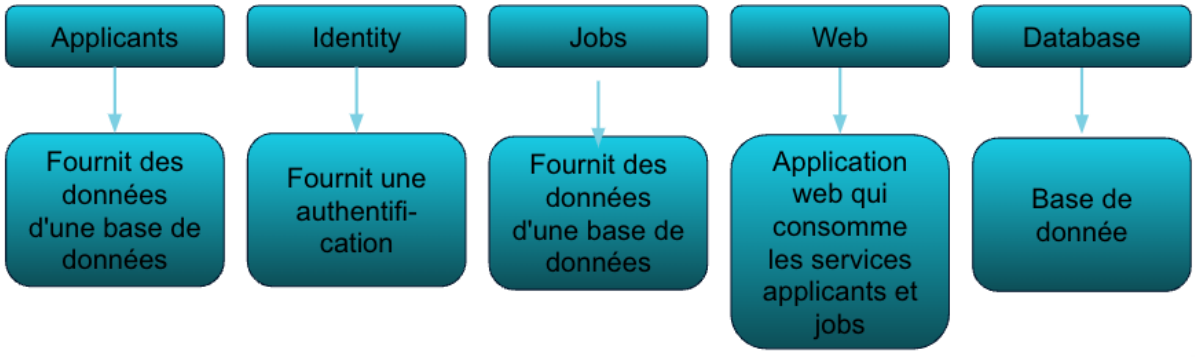
Enfin, nous avons décidé de migrer l'application vers Azure, et nous avons créé un cluster AKS pour déployer l'application. Nous avons également créé un paquet helm pour simplifier l'installation de l'application. Enfin, nous avons réalisé la connexion au cluster et l'installation du paquet helm.

En résumé, nous avons tous contribué à cette tâche en utilisant nos compétences spécifiques, ce qui a permis de mener à bien le projet dans les délais impartis.

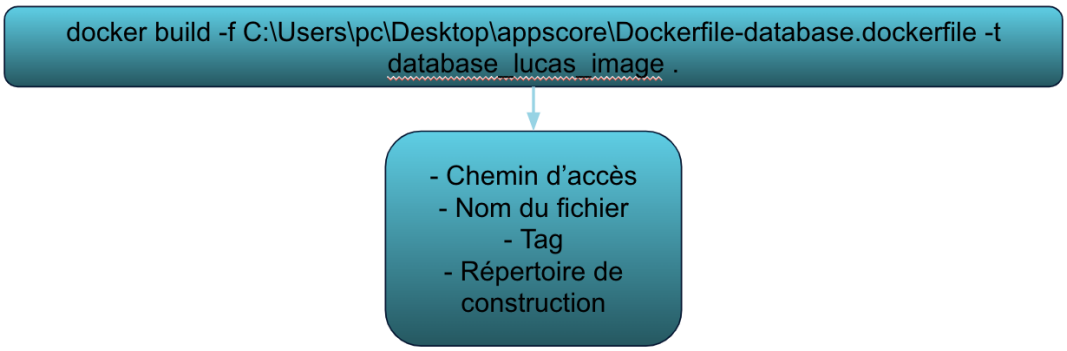
I. Création des images

Dans le cadre de notre projet, cette partie portera sur la création des images Docker correspondant aux différentes composantes de l'application : applicants, identity, jobs, web et database.

Pour ce faire, nous avons utilisé les fichiers Docker fournis au préalable et nous avons créé des images à partir de ces fichiers à l'aide des commandes Docker build. Nous avons ainsi obtenu les images correspondantes que nous avons ensuite poussées dans un Docker Hub dédié. Grâce à cette approche, nous avons pu simplifier le déploiement de notre application en permettant une mise en production rapide et facile.



La commande "docker build" est utilisée pour construire des images Docker à partir de fichiers Dockerfile. Elle utilise plusieurs drapeaux pour spécifier des informations telles que le chemin d'accès et le nom du fichier Dockerfile à utiliser, le tag à attribuer à l'image construite, et le répertoire de construction. Une fois les images construites, elles peuvent être poussées dans le Docker Hub pour être partagées et utilisées par d'autres utilisateurs.



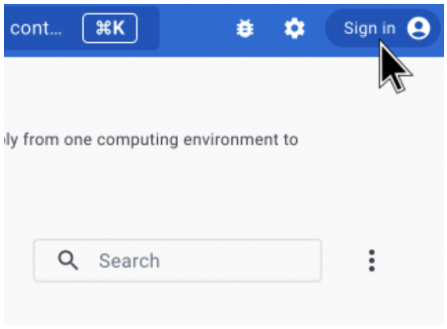
II. Création d'un docker Hub

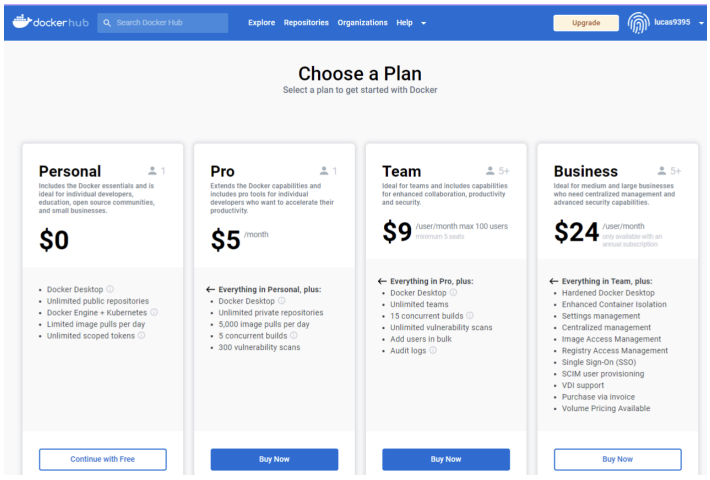
Dans cette étape, nous allons mettre en place le Docker Hub pour stocker nos images.

La création d'un Docker Hub est une étape essentielle dans le déploiement et la gestion d'images Docker. Docker Hub est une plateforme de partage d'images Docker qui permet de stocker, partager et gérer des images Docker.

Le principe de Docker Hub est simple : une fois que vous avez créé une image Docker, vous pouvez la télécharger sur Docker Hub pour la partager avec d'autres personnes. Cela vous permet de stocker et de partager vos images Docker en un seul endroit, et de faciliter le déploiement de vos applications dans différents environnements.

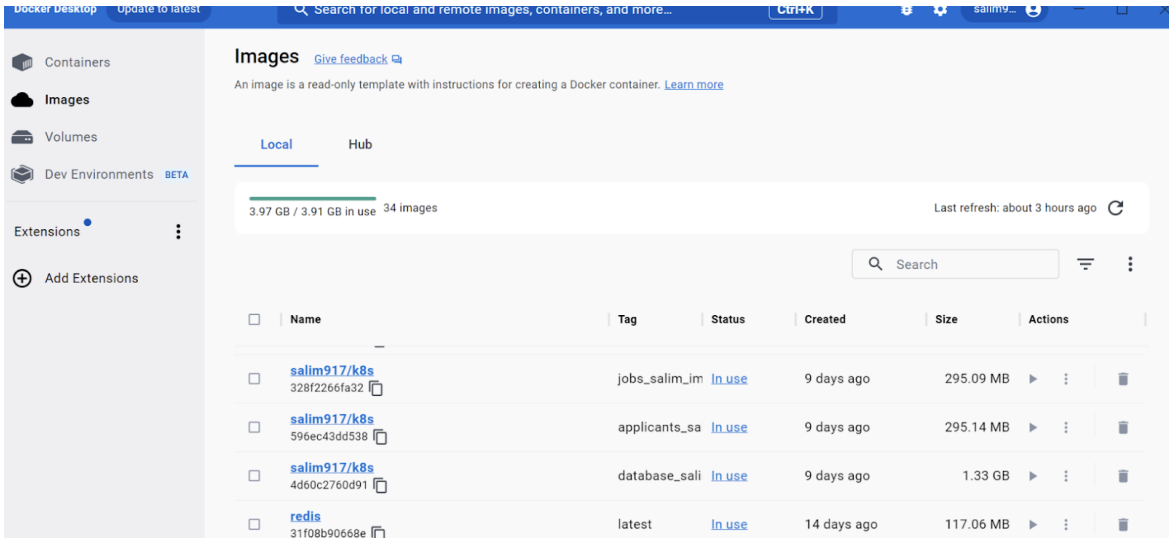
Pour créer un compte sur Docker Hub, il suffit de se rendre sur le site web de Docker Hub, de saisir ses informations personnelles, notamment son adresse e-mail, de choisir un nom d'utilisateur et un mot de passe. Ensuite, il suffit de valider son compte et de choisir le forfait gratuit. Une fois votre compte créé, vous pouvez créer un référentiel pour stocker vos images Docker.





Le fonctionnement de Docker Hub est très simple : il suffit de pousser (push) les images Docker créées sur votre ordinateur vers le référentiel de Docker Hub. Cela permet de stocker vos images en toute sécurité et de les partager avec d'autres utilisateurs.

En somme, Docker Hub est un outil de gestion et de partage d'images Docker qui facilite le déploiement de vos applications. Il permet de stocker et de partager des images Docker en toute sécurité et de faciliter la collaboration entre développeurs.



III. Push des images dans le docker Hub

Une fois que les images Docker ont été créées, la prochaine étape consiste à les pousser vers un registre de conteneurs tel que Docker Hub. La commande "docker push" est utilisée pour pousser les images Docker locales vers le registre de conteneurs.

Cette étape est cruciale pour rendre les images accessibles à partir d'autres systèmes et pour que Kubernetes puisse les utiliser lorsqu'il déploie les différents services de l'application. Le registre de conteneurs permet de stocker et de distribuer les images Docker de manière centralisée et sécurisée, et il peut être utilisé pour déployer des conteneurs sur différents environnements, y compris les environnements de production.

Dans notre cas, nous avons utilisé Docker Hub comme registre de conteneurs pour stocker nos images Docker. Pour pousser les images vers Docker Hub, nous avons utilisé la commande "docker push" suivie du nom de l'image avec le tag approprié. Par exemple, "docker push salim917/k8s:identity_salim_image" pousse l'image nommée "identity_salim_image" avec le tag "salim917/k8s" vers le registre de conteneurs Docker Hub.

Une fois que les images ont été poussées vers Docker Hub, elles peuvent être récupérées à partir d'autres systèmes et utilisées pour déployer des conteneurs en utilisant Kubernetes ou toute autre plate-forme de conteneurs compatible.

IV. Rédaction des fichiers YAML pour kubernetes :

Ci-joints le contenu des fichiers values

Rabbitmq:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rabitmq-deployment
  labels:
    app: rabitmq
spec:
  replicas: 1
  selector:
    matchLabels:
      app: rabitmq-pod
  template:
    metadata:
      labels:
        app: rabitmq-pod
    spec:
      containers:
      - name: rabitmq
        image: rabbitmq:3-management
        ports:
        - containerPort: 5672
        - containerPort: 15672
        env:
          - name: RABBIMQ_DEFAULT_USER
            value: guest
          - name: RABBIMQ_DEFAULT_PASSWORD
            value: guest

---
apiVersion: v1
kind: Service
metadata:
  name: service-rabittmq
spec:
  selector:
    app: rabitmq-pod
  ports:
    - name: port5672
      protocol: TCP
      port: 5672
```

```
      targetPort: 5672
    - name: port15672
      protocol: TCP
      port: 15672
      targetPort: 15672
  type: ClusterIP
```

Applicants :

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-applicants-deployment
  labels:
    app: api-applicants
spec:
  replicas: 1
  selector:
    matchLabels:
      app: api-applicants-pod
  template:
    metadata:
      labels:
        app: api-applicants-pod
    spec:
      containers:
      - name: applicants-container
        image: salim917/k8s:applicants_salim_image
        env:
        - name: ConnectionString
          value:
"Server=service-database;User=sa;Password=Pass@word;Database=dotnetgigs.applica
nts;"
        - name: HostRabbitmq
          value: service-rabittmq.default
        - name: RABBIMQ_USERNAME
          value: guest
        - name: RABBIMQ_PASSWORD
          value: guest
        ports:
        - containerPort: 80
        resources:
          requests:
            memory: "500Mi"
```

```
            cpu: "0.5"
          limits:
            memory: "1500Mi"
            cpu: "1"
      restartPolicy: Always
    env:

---
apiVersion: v1
kind: Service
metadata:
  name: service-api-applicants
spec:
  selector:
    app: api-applicants-pod
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: ClusterIP
```

Database:

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: database-deployment
  labels:
    app: database
spec:
  replicas: 1
  selector:
    matchLabels:
      app: database-pod
  template:
    metadata:
      labels:
        app: database-pod
    spec:
      containers:
      - name: sql-container
        image: salim917/k8s:database_salim_image
        ports:
```

```
        - containerPort: 1433
        volumeMounts:
            - mountPath: /var/opt/mssql/lib/
              name: database-volume
      volumes:
        - name: database-volume
          persistentVolumeClaim:
            claimName: mysql-volume-claim



---
apiVersion: v1
kind: Service
metadata:
  name: service-database
spec:
  selector:
    app: database-pod
  ports:
    - protocol: TCP
      port: 1433
      targetPort: 1433
  type: ClusterIP
---
#volume
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-volume-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

Identity:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-identity-deployment
  labels:
```

```
    app: api-identity
spec:
  replicas: 1
  selector:
    matchLabels:
      app: api-identity-pod
  template:
    metadata:
      labels:
        app: api-identity-pod
    spec:
      containers:
      - name: service-api-identity
        image: salim917/k8s:identity_salim_image
        ports:
        - containerPort: 80
        env:
        - name: RedisHost
          value: "service-user-data:6379"
        - name: HostRabbitmq
          value: service-rabittmq
        resources:
          requests:
            memory: "500Mi"
            cpu: "0.5"
          limits:
            memory: "1500Mi"
            cpu: "1"
      restartPolicy: Always

---

apiVersion: v1
kind: Service
metadata:
  name: service-api-identity
spec:
  selector:
    app: api-identity-pod
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: ClusterIP
```

Jobs:

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-jobs-deployment
  labels:
    app: api-jobs
spec:
  replicas: 1
  selector:
    matchLabels:
      app: api-jobs-pod
  template:
    metadata:
      labels:
        app: api-jobs-pod
    spec:
      containers:
      - name: service-api-jobs
        image: salim917/k8s:jobs_salim_image
        env:
        - name: ConnectionString
          value:
"Server=service-database.default;User=sa;Password=Pass@word;Database=dotnetgigs
.jobs;"
        - name: HostRabbitmq
          value: service-rabittmq


        ports:
        - containerPort: 80
        resources:
          requests:
            memory: "500Mi"
            cpu: "0.5"
          limits:
            memory: "1500Mi"
            cpu: "1"
      restartPolicy: Always


---
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: service-api-jobs
spec:
  selector:
    app: api-jobs-pod
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: ClusterIP
```

User-Data
:

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: user-data-deployment
  labels:
    app: user-data
spec:
  replicas: 1
  selector:
    matchLabels:
      app: user-data-pod
  template:
    metadata:
      labels:
        app: user-data-pod
    spec:
      containers:
      - name: user-data
        image: redis:latest
        ports:
        - containerPort: 6379
        resources:
          requests:
            memory: "100Mi"
            cpu: "4m"

---
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: service-user-data
spec:
  selector:
    app: user-data-pod
  ports:
    - protocol: TCP
      port: 6379
      targetPort: 6379
  type: ClusterIP
```

Web:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-deployment
  labels:
    app: web
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web-pod
  template:
    metadata:
      labels:
        app: web-pod
    spec:
      containers:
      - name: web-container
        image: salim917/k8s:web_salim_image
        ports:
        - containerPort: 80
        resources:
          requests:
            memory: "500Mi"
            cpu: "0.5"
          limits:
            memory: "1500Mi"
            cpu: "1"
        env:
```

```
          - name: APSNETCORE_ENVIRONMENT
            value: Development
        # - name:  identity
        #   value: srv-api-identity
        # - name: jobs
        #   value: srv-api-jobs
---

apiVersion: v1
kind: Service
metadata:
  name: service-web
spec:
  selector:
    app: web-pod
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: ClusterIP
```

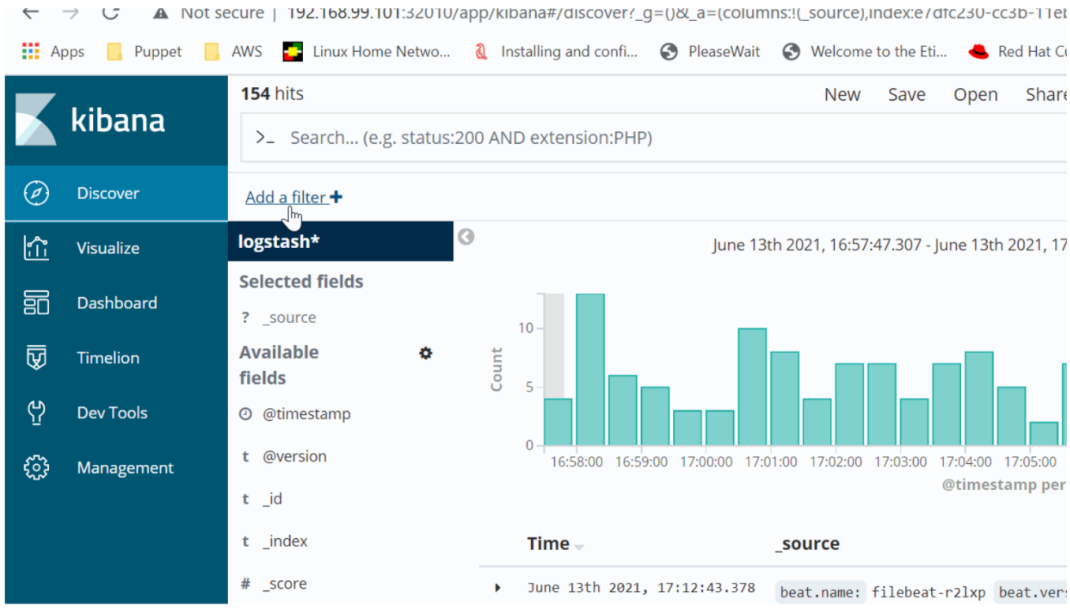
Ingress:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: service-web-ingress
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - host: localhost
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: service-web
            port:
              number: 80
```

V. Mise en place de la solution ELK

On exécute les commandes suivantes:

```
# git clone https://github.com/hussainaphroj/ELK-kubernetes.git
# kubectl update -f rbac.yml -f elastic-service.yml -f elastic.yml -f logstash-config.yml -f logstash-deployment.yml -f kibana.yml -f web-deployment.yml
```



VI. Mise en place de la solution de suivi de santé

On exécute les commandes suivantes:

```
# helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
# helm repo update
# helm install monitoring prometheus-community/kube-prometheus-stack --set
prometheus-node-exporter.hostRootFsMount.enabled=false -f values.yml
```

Contennu du fichier values:

applicants.yml | elastic.yml | values.yml

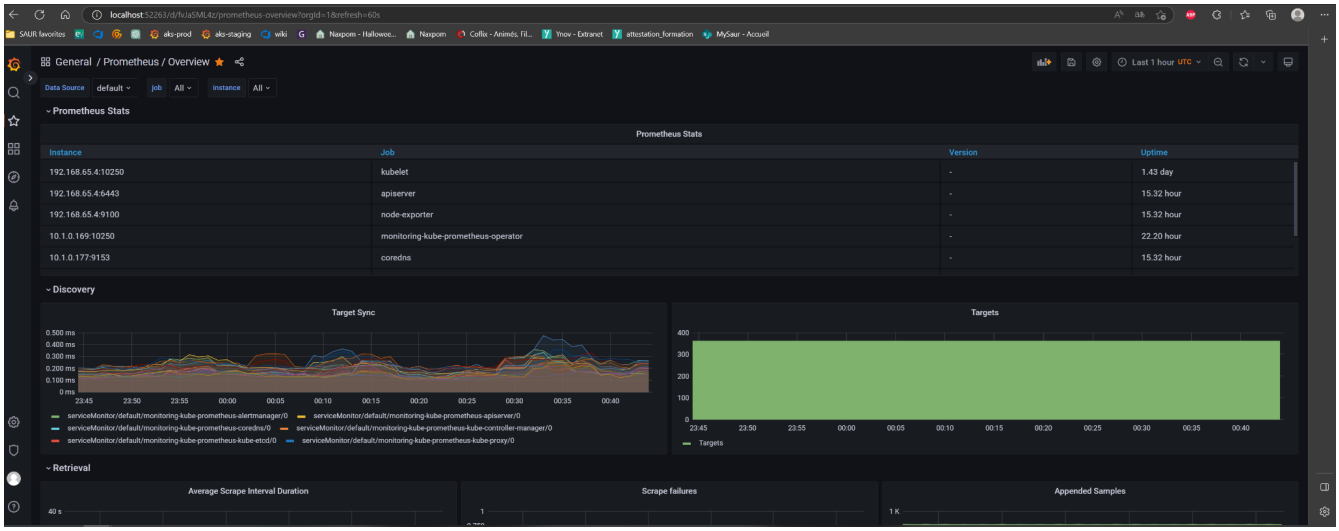
C: > Users > ksalim > Downloads > projects_kube > k8s > values.yml > {} kube-state-metrics

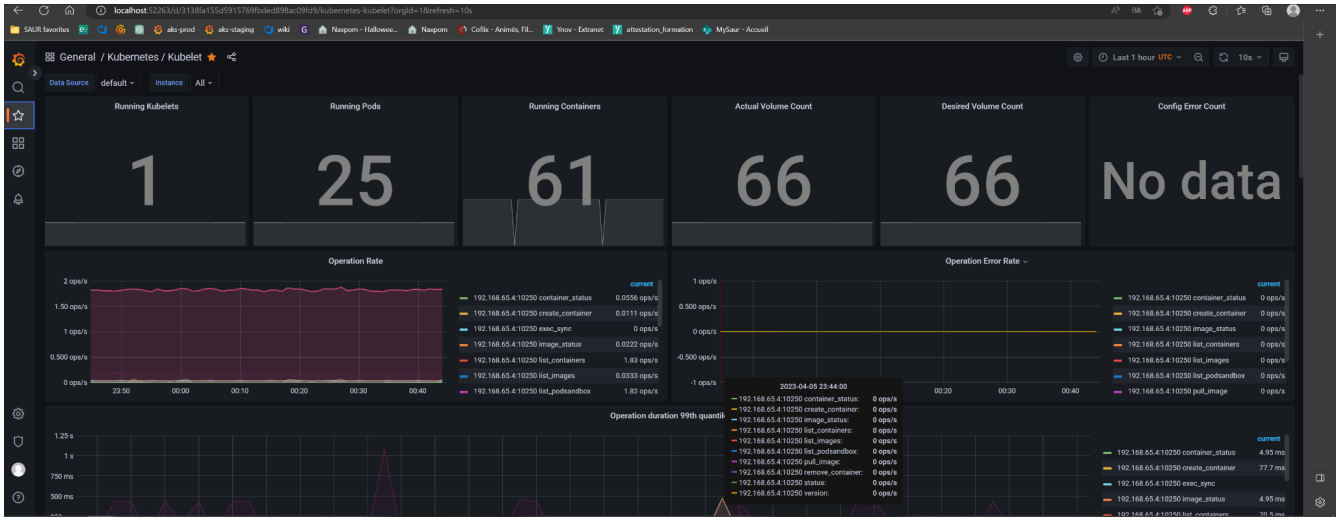
```yaml
kube-state-metrics:
  tolerations:
  - key: CriticalAddonsOnly
    operator: Exists

prometheus-node-exporter:
  tolerations:
  - key: CriticalAddonsOnly
    operator: Exists

prometheus:
  prometheusSpec:
    tolerations:
      - key: CriticalAddonsOnly
        operator: Exists
grafana:
  sidecar:
    dashboards:
      provider:
        allowUiUpdates: true
        disableDelete: false
```

Le résultat:

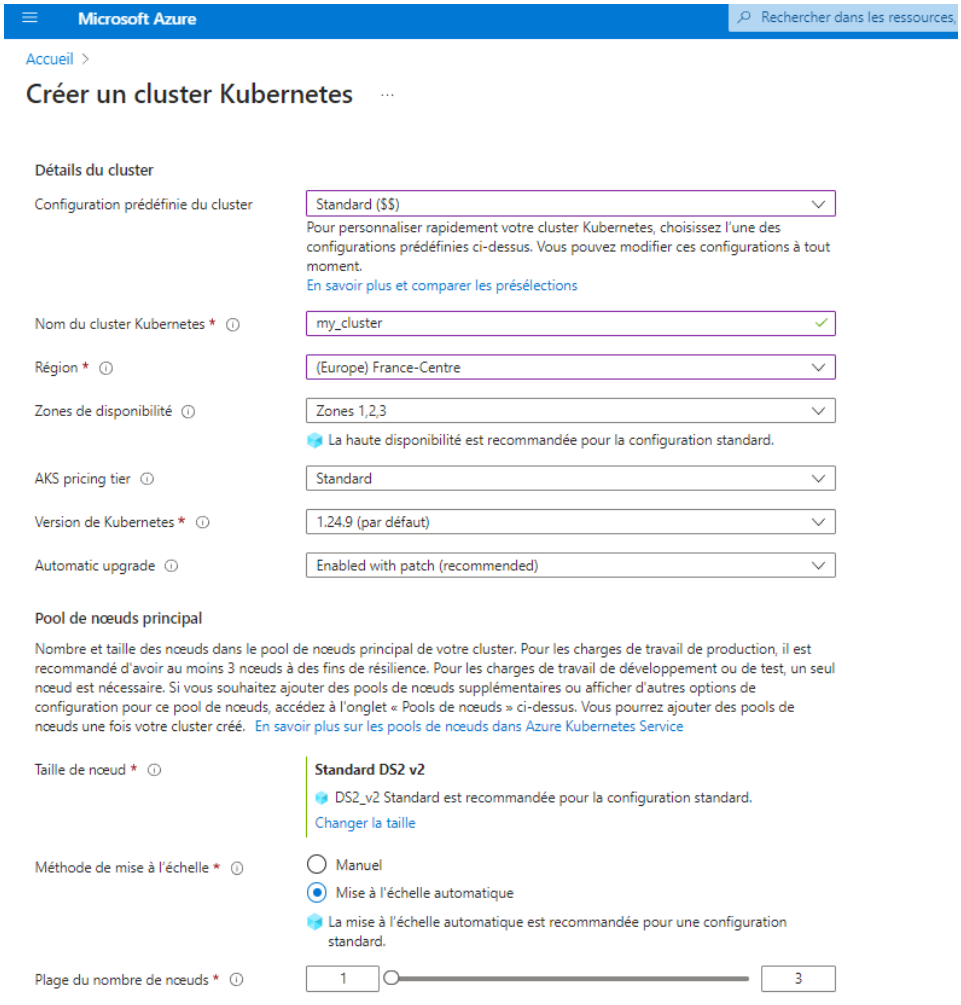




VII. Migration vers Azure Création cluster aks

A. Création du cluster

Création du groupe de ressource, choix de la zone de disponibilité, choix de l'hébergement France - Centre, choix de la configuration du nœud & choix du nombre de nœuds.
Nous allons utilisé Azure Kubernetes Service qui est un système open source permettant automatiser le déploiement, la mise à l'échelle et la gestion de logiciels. Conçu à l'origine par Google,le projet est maintenant maintenu par la Cloud Native Computing Foundation



B. Création du paquet HELM (salim)

On exécute les commandes suivantes:

helm package C:\Users\ksalim\Downloads\projects_kube\k8s\deploy-chart\cluster_deployment\deploy-chart

```
PS C:\Users\ksalim\Downloads\projects_kube\k8s\deploy-chart\cluster_deployment\deploy-chart> helm package C:\Users\ksalim\Downloads\projects_kube\k8s\deploy-chart\cluster_deployment\deploy-chart
Successfully packaged chart and saved it to: C:\Users\ksalim\Downloads\projects_kube\k8s\deploy-chart\cluster_deployment\deploy-chart\deploy-chart-0.1.0.tgz
PS C:\Users\ksalim\Downloads\projects_kube\k8s\deploy-chart\cluster_deployment\deploy-chart>
```

helm repo index C:\Users\ksalim\Downloads\projects_kube\k8s\deploy-chart\cluster_deployment\deploy-chart

```
PS C:\Users\ksalim\Downloads\projects_kube\k8s\deploy-chart\cluster_deployment\deploy-chart> helm repo index C:\Users\ksalim\Downloads\projects_kube\k8s\deploy-chart\cluster_deployment\deploy-chart
PS C:\Users\ksalim\Downloads\projects_kube\k8s\deploy-chart\cluster_deployment\deploy-chart>
```

Puis on push les fichiers:

git add .\deploy-chart-0.1.0.tgz
git commit -m deploy-chart-0.1.0.tgz
git push

git add .\index.yaml
git commit -m .\index.yaml
git push

C. Connexion au cluster

Pour se connecter au cluster, il faut utiliser la commande “Az login” afin de choisir le compte Microsoft avec lequel nous allons nous connecter.

```
PS C:\Users\mariv\Desktop\Projet_Docker\appscore> az login
A web browser has been opened at https://login.microsoftonline.com/organizations/oauth2/v2.0/authorize. Please continue the login in the web browser. If no web browser is available or if the web browser fails to open, use dev
ice code flow with `az login --use-device-code`.
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "38e72bba-3c22-4382-9323-ac1612931297",
    "id": "98ad727c-0eac-4c3e-90cb-0011453f5107",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Azure for Students",
    "state": "Enabled",
    "tenantId": "38e72bba-3c22-4382-9323-ac1612931297",
    "user": {
      "name": "xavier.marivint@ynov.com",
      "type": "user"
    }
  }
]
```

You have logged into Microsoft Azure!

You can close this window, or we will redirect you to the Azure CLI documentation in 1 minute.

Announcements

[Windows only] Starting in May 2023, Azure CLI will authenticate using the Web Account Manager (WAM) broker by default.

To help us collect feedback on the new login experience, you may opt-in to use WAM by running the following commands:

```
az config set core.allow_broker=true
az account clear
az login
```

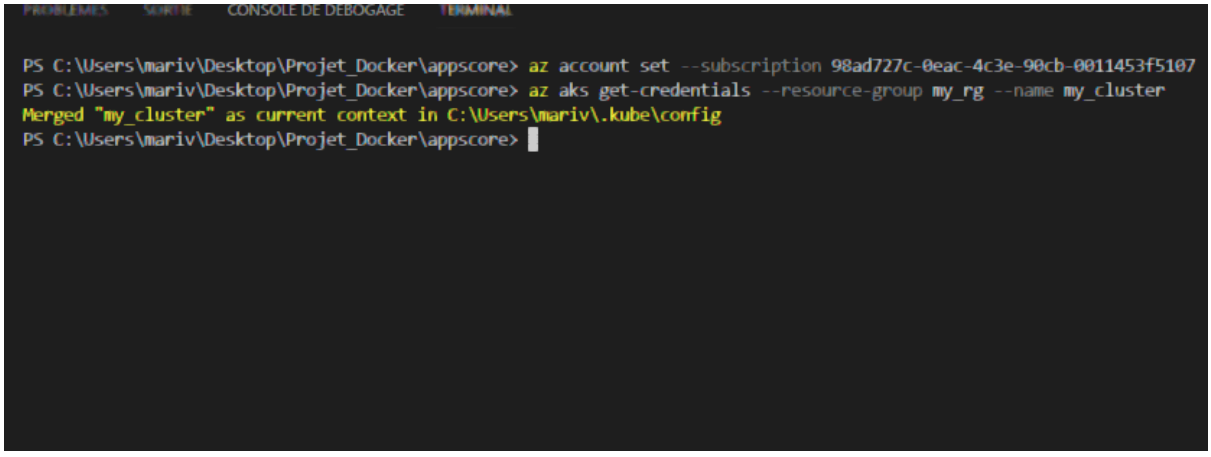
Pour accéder au cluster sur Azure il faut écrire ces 2 commandes :

1. Ouvrir Cloud Shell ou Azure CLI
2. Exécuter les commandes suivantes

```
az account set --subscription 98ad727c-0eac-4c3e-90cb-0011453f5107
```

```
az aks get-credentials --resource-group my_rg --name my_cluster
```

Résultat après avoir écrit la commande



D. Installation du paquet helm

Après la création du paquet, il nous faut maintenant l'installer. Voici comment nous allons procéder :
Il faut utiliser la commande "Helm install az-deploy".

& ajout du repo ou est installé le paquet helm :

helm repo add deployment https://github.com/salim791213/cluster_deployment/blob/main

Lister les services :

Utiliser la commande :
“*Kebectl get svc*”

```
PS C:\Users\mariv\Desktop\Projet_Docker\appscore> kubectl get svc
NAME                    TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)              AGE
kubernetes              ClusterIP   10.0.0.1       <none>        443/TCP              26h
service-api-applicants  ClusterIP   10.0.137.149   <none>        80/TCP               9h
service-api-identity    ClusterIP   10.0.3.134     <none>        80/TCP               9h
service-api-jobs        ClusterIP   10.0.249.32    <none>        80/TCP               9h
service-database        ClusterIP   10.0.211.247   <none>        1433/TCP             9h
service-rabittmq        ClusterIP   10.0.132.226   <none>        5672/TCP,15672/TCP   9h
service-user-data       ClusterIP   10.0.9.218     <none>        6379/TCP             9h
service-web             ClusterIP   10.0.178.190   <none>        80/TCP               9h
PS C:\Users\mariv\Desktop\Projet_Docker\appscore>
```

La commande "kubectl get svc" est une commande de l'outil en ligne de commande Kubernetes (kubectl) qui permet d'afficher une liste des services (services Kubernetes) actuellement déployés dans un cluster Kubernetes.
Lorsque vous exécutez la commande "kubectl get svc", elle renvoie une liste de tous les services Kubernetes dans le cluster actuellement configuré pour kubectl, avec des informations telles que le nom du service, le type de service, l'adresse IP cluster (ClusterIP), l'adresse IP externe (ExternalIP) et le port utilisé pour exposer l'application.

Cette commande est utile pour vérifier si les services sont déployés et configurés correctement, pour obtenir des informations sur les services tels que les ports et les adresses IP, et pour vérifier si les services sont accessibles à l'extérieur du cluster Kubernetes.

Lister les PODS :

Les pods sont les objets déployables qui constituent les plus petits composants essentiels de Kubernetes. Un pod représente une instance unique d'un processus en cours d'exécution dans votre cluster. Les pods contiennent un ou plusieurs conteneurs tels que des conteneurs Docker.
Utiliser la commande :

“*Kebectl get pods*”