Lab 6 : Déploiement K8s d’un système MLOps Churn

Étape 1 : Préparer l’environnement Kubernetes

Instructions :

Minikube instaling

```
PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> minikube version
minikube version: v1.37.0
commit: 65318f4cfff9c12cc87ec9eb8f4cdd57b25047f3
```

1. Démarrer Minikube (driver Docker) :

```
PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> minikube start --driver=docker --kubernetes-version=v1.28.3
😄  minikube v1.37.0 on Microsoft Windows 11 Pro 10.0.26200.7462 Build 26200.7462
✨  Using the docker driver based on user configuration
📌  Using Docker Desktop driver with root privileges
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.48 ...
💾  Downloading Kubernetes v1.28.3 preload ...
    > gcr.io/k8s-minikube/kicbase...:  561.71 KiB / 488.52 MiB  0.11% 51.72 KiB
```

2. Créer un namespace dédié au lab :

```
PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl create namespace churn-mlops
namespace/churn-mlops created
PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

3. Basculer le contexte courant sur ce namespace :

```
PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl config set-context --current --namespace=churn-mlops
Context "minikube" modified.
PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

4. Vérifier :

```
PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl get ns
NAME              STATUS   AGE
churn-mlops       Active   0s
default           Active   19m
kube-node-lease   Active   19m
kube-public       Active   19m
kube-system       Active   19m
PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl get pods
No resources found in churn-mlops namespace.
PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Étape 2 : Préparer l'image Docker de l'API churn

Créer un environnement virtuel avec Python 3.12

Windows / PowerShell et Mettre pip à jour :

```
PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> python --version
Python 3.12.10
PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> py -3.12 -m venv venv_mlops
PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> .\venv_mlops\Scripts\activate
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> python -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\anoua\projects\mlops\mlops-lab-01\venv_mlops\lib\site-packages (25.0.1)
```

Préciser les dépendances dans `requirements.txt`

(afin d'éviter toute incompatibilité entre entraînement et prédiction)

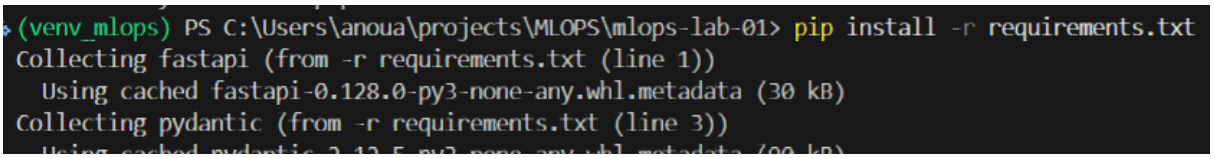
Créer ou éditer le fichier

et y mettre :

```
fastapi          # Framework web rapide pour créer des APIs Python
uvicorn[standard]  # Serveur ASGI pour exécuter FastAPI
pydantic         # Validation et typage des données
scikit-learn==1.7.2 # Entraînement et inférence des modèles ML
pandas==2.2.3      # Manipulation de données tabulaires
```

```
numpy==2.1.3      # Calcul numérique
joblib==1.4.2     # Sérialisation et chargement des modèles
```

Installer les dépendances :



Cette étape est obligatoire pour :

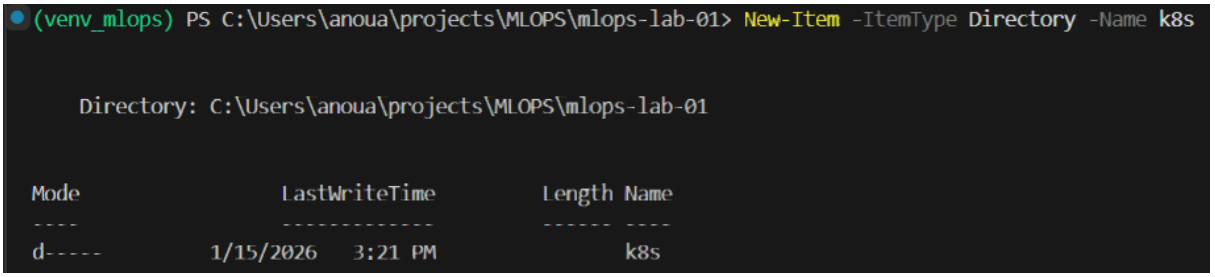
éviter les erreurs de compatibilité `scikit-learn` ,garantir que le modèle chargé par l'API correspond exactement aux versions installées.

- **éviter les erreurs de compatibilité `scikit-learn` ,**
- **garantir que le modèle chargé par l'API correspond exactement aux versions installées.**

Étape 3 : Créer le dossier des manifests Kubernetes

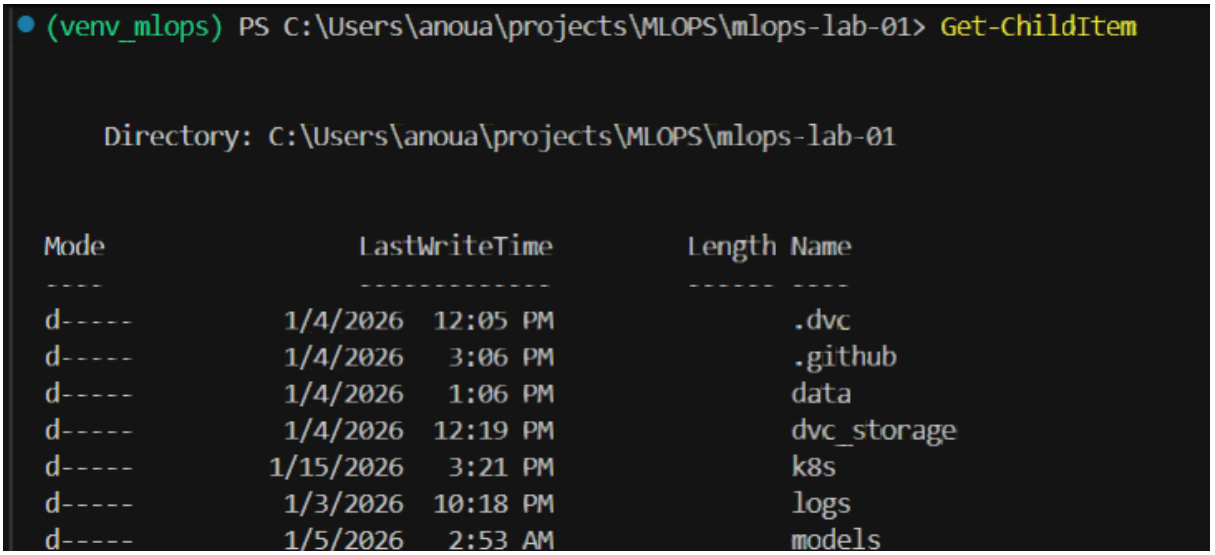
Instructions :Sous PowerShell :

`New-Item -ItemType Directory -Name k8s`



Vérifier :

`Get-ChildItem`



Tu dois voir :

```
mlops-lab-01/
├── src/
├── data/
├── models/
├── logs/
├── registry/
├── k8s/
└── ...
```

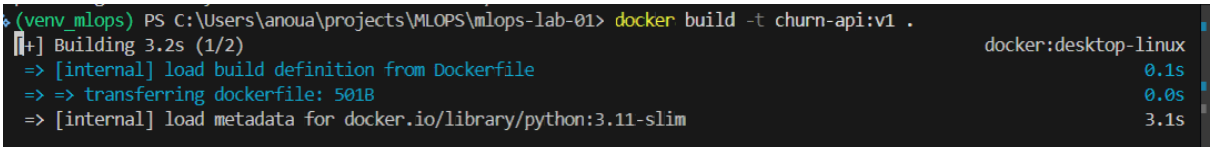
Étape 4 : Construire l'image Docker (tag versionné)

Instructions :Se placer dans le dossier du projet :

```
cd ./mlops-lab-01
```

Construire l'image Docker avec un tag versionné (jamais `latest`) :

```
# Construire l'image du projet avec un tag de version (v1, v2, ...)
docker build -t churn-api:v1 .
```



Vérifier l'image localement :

```
# PowerShell
docker images | Select-String churn-api
```

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> docker images | Select-String churn-api
WARNING: This output is designed for human readability. For machine-readable output, please use --format.

churn-api:latest                  15c3c801b7f6       516MB             0B
churn-api:v1                      78dd3690aa4a       852MB             0B
```

Étape 5 : Charger explicitement l'image dans Minikube

Instructions :Sauvegarder l'image Docker :

```
docker save churn-api:v1 -o churn-api_v1.tar
```

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> docker save churn-api:v1 -o churn-api_v1.tar
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Charger l'image dans Minikube :

```
minikube image load churn-api_v1.tar
```

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> minikube image load churn-api_v1.tar
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Vérifier que l'image est disponible dans Minikube :

```
# PowerShell
minikube image ls | Select-String churn-api
```

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> minikube image ls | Select-String churn-api

  docker.io/library/churn-api:v1


(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Étape 6 : Deployment Kubernetes pour l'API churn

Instructions :

1. Créer le fichier :

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> New-Item k8s/deployment.yaml


    Directory: C:\Users\anoua\projects\MLOPS\mlops-lab-01\k8s


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         1/15/2026   3:44 PM              0 deployment.yaml
```

2. Coller le contenu suivant dans `k8s/deployment.yaml` :

README.md | requirements.txt M | deployment.yaml U

k8s > deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: churn-api
spec:
  replicas: 2
  selector:
    matchLabels:
      app: churn-api
  template:
    metadata:
      labels:
        app: churn-api
    spec:
      containers:
        - name: api
          image: churn-api:v1 # v1 presente le tag de l'image cible
          ports:
            - containerPort: 8000
```

3. Appliquer le manifest :

`kubectl apply -f k8s/deployment.yaml`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl apply -f k8s/deployment.yaml
deployment.apps/churn-api created
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Suivre le rollout :

`kubectl rollout status deployment churn-api`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl rollout status deployment churn-api
deployment "churn-api" successfully rolled out
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Vérifier :

`kubectl get pods -l app=churn-api -o wide`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl get pods -l app=churn-api -o wide
NAME                         READY   STATUS    RESTARTS   AGE   IP           NODE       NOMINATED NODE   RE
ADINESS GATES
churn-api-7d686dfbf7-lc7d4   1/1     Running   0          53s   10.244.0.4   minikube   <none>           <r
one>
churn-api-7d686dfbf7-z278p   1/1     Running   0          53s   10.244.0.5   minikube   <none>           <r
one>
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Étape 7 : Exposer l'API via un Service NodePort

Instructions :Créer le fichier :

`New-Item k8s/service.yaml`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> New-Item k8s/service.yaml


    Directory: C:\Users\anoua\projects\MLOPS\mlops-lab-01\k8s


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         1/15/2026   3:57 PM              0 service.yaml
```

Coller le contenu suivant dans `k8s/service.yaml` :

README.md | requirements.txt M | deployment.yaml U | service.yaml U

k8s > service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: churn-api-service
spec:
  type: NodePort
  selector:
    app: churn-api
  ports:
    - port: 80
      targetPort: 8000
      nodePort: 30080
```

Appliquer :

`kubectl apply -f k8s/service.yaml`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl apply -f k8s/service.yaml
>>
service/churn-api-service created
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Visualiser la création du Service :

`kubectl get svc churn-api-service`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl get svc churn-api-service
NAME                TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
churn-api-service   NodePort   10.96.57.157    <none>        80:30080/TCP   19s
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Afficher les détails du Service :

`kubectl describe svc churn-api-service`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl describe svc churn-api-service
Name:                     churn-api-service
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=churn-api
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.96.57.157
IPs:                      10.96.57.157
Port:                     <unset>  80/TCP
TargetPort:               8000/TCP
NodePort:                 <unset>  30080/TCP
Endpoints:                10.244.0.5:8000,10.244.0.4:8000
Session Affinity:         None
External Traffic Policy:  Cluster
Internal Traffic Policy:  Cluster
Events:                   <none>
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Ouvrir l'accès à la communication avec le cluster (Windows / Minikube driver Docker) :

`kubectl port-forward svc/churn-api-service 30080:80`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl port-forward svc/churn-api-service 30080:80
Forwarding from 127.0.0.1:30080 -> 8000
Forwarding from [::1]:30080 -> 8000
```

Tester l'API, via Postman :

POST http://127.0.0.1:30080/predict

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

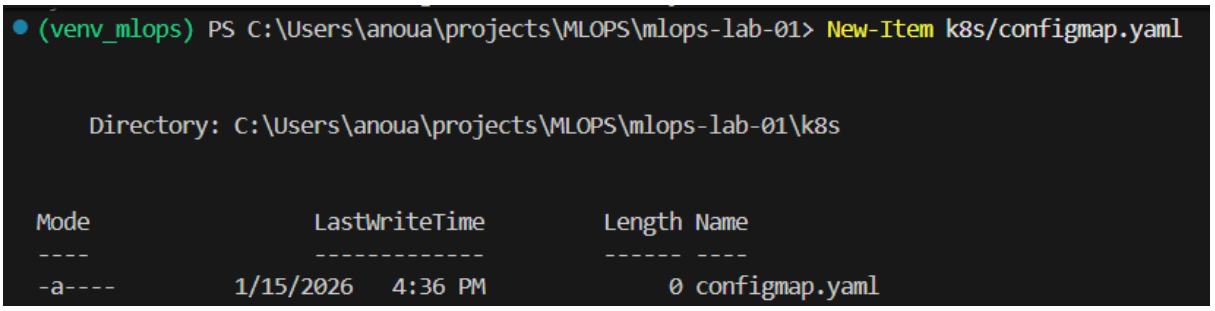
```
{
  "tenure_months": 48,
  "num_complaints": 0,
  "avg_session_minutes": 60,
  "plan_type": "premium",
  "region": "EU",
  "request_id": "req-safe"
}
```

résultat :

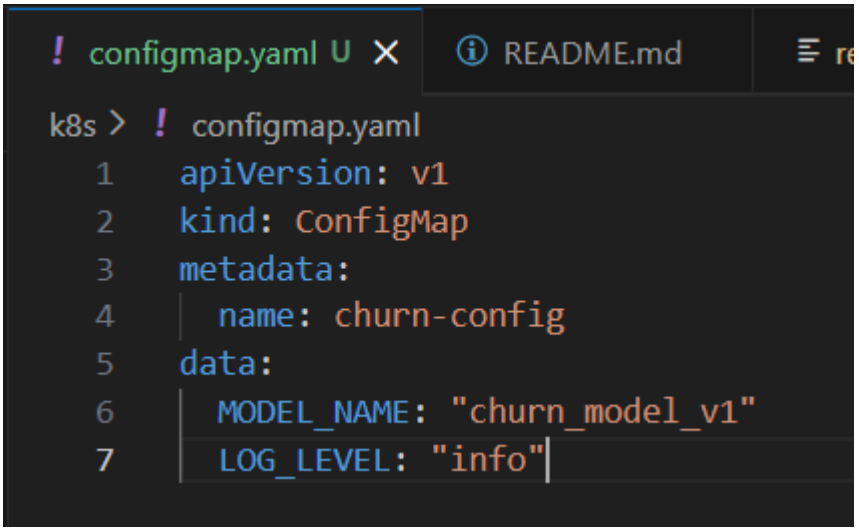
Étape 8 : Injecter la configuration MLOps via ConfigMap

Instructions :Créer :

New-Item k8s/configmap.yaml



Coller :



Appliquer :

`kubectl apply -f k8s/configmap.yaml`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl apply -f k8s/configmap.yaml
configmap/churn-config created
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

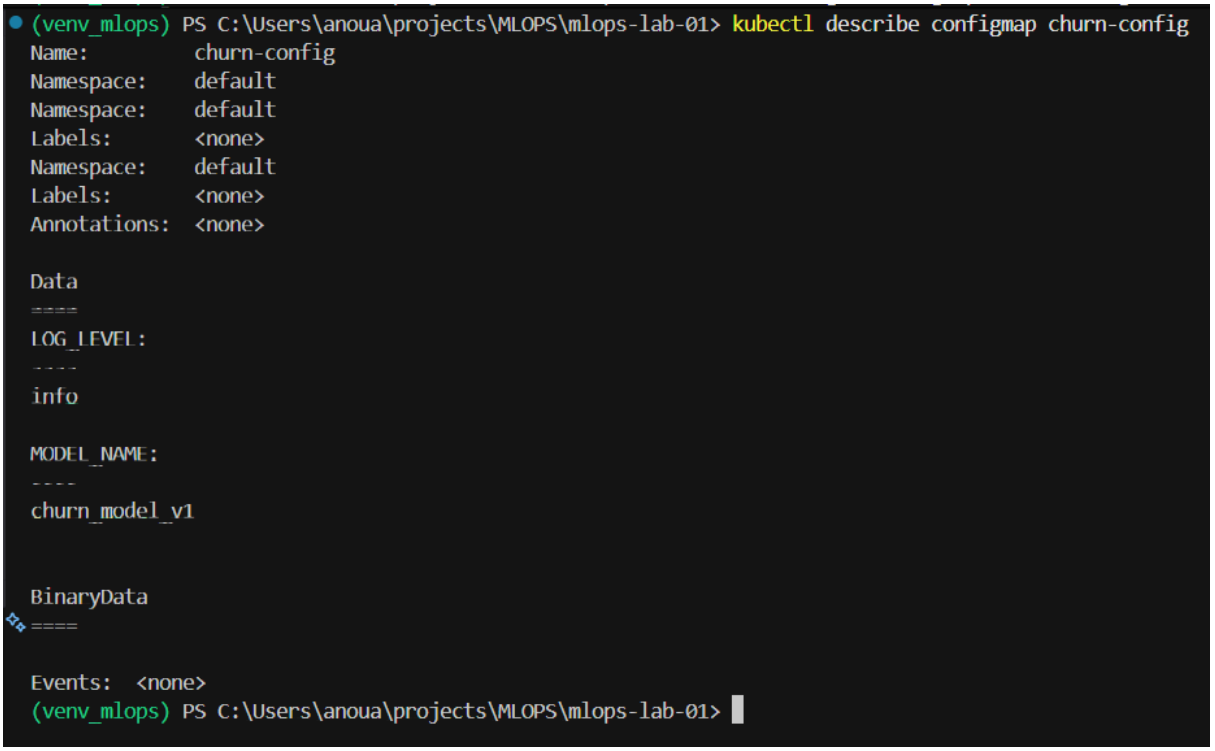
Visualiser la création du ConfigMap :

`kubectl get configmap churn-config`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl get configmap churn-config
NAME           DATA   AGE
churn-config   2      21s
```

Afficher le contenu du ConfigMap :

`kubectl describe configmap churn-config`



Modifier `k8s/deployment.yaml`

pour injecter ces variables dans le conteneur :Dans la section `containers: - name: api` **, ajouter :**

configmap.yaml U | README.md | requirements.txt M | deployment.yaml U

k8s > deployment.yaml

```yaml
  spec:
    selector:
  template:
    metadata:
      labels:
        app: churn-api
    spec:
      containers:
        - name: api
          image: churn-api:v1 # v1 presente le tag de l'image cible
          ports:
            - containerPort: 8000
          env:
            - name: MODEL_NAME
              valueFrom:
                configMapKeyRef:
                  name: churn-config
                  key: MODEL_NAME
            - name: LOG_LEVEL
              valueFrom:
                configMapKeyRef:
                  name: churn-config
                  key: LOG_LEVEL
```

Réappliquer le Deployment :

`kubectl apply -f k8s/deployment.yaml`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl apply -f k8s/deployment.yaml
deployment.apps/churn-api configured
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Suivre le rollout :

`kubectl rollout restart deployment churn-api`
`kubectl rollout status deployment churn-api`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl rollout restart deployment churn-api
deployment.apps/churn-api restarted
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl rollout status deployment churn-api
deployment "churn-api" successfully rolled out
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> S
```

Vérifier que les variables sont bien injectées dans un Pod :

`kubectl exec -it deploy/churn-api -- printenv MODEL_NAME`
`kubectl exec -it deploy/churn-api -- printenv LOG_LEVEL`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl exec -it deploy/churn-api -- printenv M
ODEL_NAME
churn_model_v1
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl exec -it deploy/churn-api -- printenv L
OG_LEVEL
info
```

Étape 9 : Gérer les secrets (MONITORING_TOKEN)

Instructions :Encoder une valeur simple en base64 (exemple `"abc123"`) : Depuis PowerShell :

`[Convert]::ToBase64String([Text.Encoding]::UTF8.GetBytes("abc123"))`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> [Convert]::ToBase64String([Text.Encoding]::UTF8
.GetBytes("abc123"))
YWJjMTIz
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Garder la valeur obtenue, par ex : `YWJjMTIz` Créer le fichier :

`New-Item k8s/secret.yaml`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> New-Item k8s/secret.yaml


    Directory: C:\Users\anoua\projects\MLOPS\mlops-lab-01\k8s


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         1/15/2026   5:04 PM              0 secret.yaml
```

Coller :

```yaml
k8s > ! secret.yaml
1  apiVersion: v1
2  kind: Secret
3  metadata:
4    name: churn-secret
5  type: Opaque
6  data:
7    MONITORING_TOKEN: "YWJjMTIz"
```

Appliquer et Visualiser la création du Secret :

`kubectl apply -f k8s/secret.yaml`
`kubectl get secret churn-secret`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl apply -f k8s/secret.yaml
secret/churn-secret created
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl get secret churn-secret
NAME           TYPE     DATA   AGE
churn-secret   Opaque   1      27s
```

Afficher les détails (sans afficher les valeurs en clair) :
`kubectl describe secret churn-secret`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl describe secret churn-secret
Name:         churn-secret
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
MONITORING_TOKEN:  6 bytes
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Ajouter la variable d'environnement dans `k8s/deployment.yaml` (dans `env:` déjà présent) :

k8s > ! deployment.yaml

```yaml
            - name: LOG_LEVEL
              valueFrom:
                configMapKeyRef:
                  name: churn-config
                  key: LOG_LEVEL
            - name: MONITORING_TOKEN
              valueFrom:
                secretKeyRef:
                  name: churn-secret
                  key: MONITORING_TOKEN
```

Réappliquer ET Vérifier le redémarrage et l'état des Pods :

`kubectl apply -f k8s/deployment.yaml`

`kubectl get pods`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl apply -f k8s/deployment.yaml
deployment.apps/churn-api configured
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl get pods
NAME                         READY   STATUS        RESTARTS   AGE
churn-api-54b6c6d49b-d6scb   0/1     Terminating   0          40m
churn-api-54b6c6d49b-jk76f   1/1     Terminating   0          40m
churn-api-5cc7f85545-gw2rd   1/1     Running       0          4s
churn-api-5cc7f85545-qlbpg   1/1     Running       0          6s
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Vérifier que la variable d'environnement est bien injectée dans un Pod :

`kubectl exec -it deploy/churn-api -- printenv MONITORING_TOKEN`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl exec -it deploy/churn-api -- printenv M
ONITORING_TOKEN
abc123
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Étape 10 : Mise en place des endpoints de santé et des probes Kubernetes pour l'API Churn

Instructions :Ajouter les endpoints nécessaires aux probes Kubernetes dans l'API.

Ouvrir le fichier contenant la définition de l'API FastAPI (par exemple

`src/api.py`).

Ajouter les endpoints suivants :

```
@app.get("/health")
def health() -> dict[str, Any]:
    """
    Endpoint de santé de l'API.


    Vérifie simplement qu'un modèle courant est bien configuré.


    Retour
    ------
    dict
        - status : "ok" ou "error"
        - current_model : nom du modèle courant (si OK)
        - detail : message d'erreur (si error)
    """
    try:
        model_name = get_current_model_name()
        return {"status": "ok", "current_model": model_name}
    except Exception as exc:  # pragma: no cover - simple endpoint de debu
g
        return {"status": "error", "detail": str(exc)}

@app.get("/startup")
def startup() -> dict[str, Any]:
    """
    Endpoint utilisé par Kubernetes startupProbe.

    L'application est considérée comme démarrée UNIQUEMENT si :
    - le registry existe,
    - le fichier current_model.txt existe,
    - le fichier n'est pas vide.
    """
    if not REGISTRY_DIR.exists():
        raise HTTPException(
```

```
            status_code=503,
            detail="Registry non monté (PVC absent ou incorrect).",
        )

    if not CURRENT_MODEL_PATH.exists():
        raise HTTPException(
            status_code=503,
            detail="Aucun modèle courant. Lancer train.py (avec gate) d'abor
d.",
        )

    name = CURRENT_MODEL_PATH.read_text(encoding="utf-8").strip()
    if not name:
        raise HTTPException(
            status_code=503,
            detail="current_model.txt vide.",
        )

    return {
        "status": "ok",
        "current_model": name,
    }


@app.get("/ready")
def ready() → dict[str, Any]:
    try:
        model_name = get_current_model_name()
        return {"status": "ready", "current_model": model_name}
    except Exception as exc:
        raise HTTPException(status_code=503, detail=str(exc))
```

```python
src > api.py > health

@app.get("/health")
def health() -> dict[str, Any]:
    """
    Endpoint de santé de l'API.


    Vérifie simplement qu'un modèle courant est bien configuré.


    Retour
    ------
    dict
        - status : "ok" ou "error"
        - current_model : nom du modèle courant (si OK)
        - detail : message d'erreur (si error)
    """
    try:
        model_name = get_current_model_name()
        return {"status": "ok", "current_model": model_name}
    except Exception as exc:  # pragma: no cover - simple endpoint de debug
        return {"status": "error", "detail": str(exc)}



@app.get("/startup")
def startup() -> dict[str, Any]:
    """
    Endpoint utilisé par Kubernetes startupProbe.

    L'application est considérée comme démarrée UNIQUEMENT si :
    - le registry existe,
    - le fichier current_model.txt existe
```

Sauvegarder le fichier.

Reconstruire l'image Docker de l'API :

```
docker build-t churn-api:v1 .
```

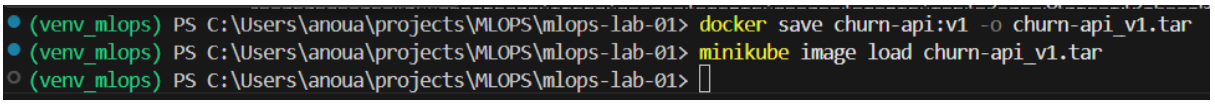
```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> docker build -t churn-api:v1 .
[+] Building 46.1s (8/9)                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                        0.1s
 => => transferring dockerfile: 501B                                                        0.0s
 => [internal] load metadata for docker.io/library/python:3.12-slim                         1.5s
 => [internal] load .dockerignore                                                           0.0s
 => => transferring context: 2B                                                             0.0s
 => [1/5] FROM docker.io/library/python:3.12-slim@sha256:d75c4b6cdd039ae966a34cd3ccab9e0e5f7299280ad  0.0s
 => [internal] load build context                                                          41.2s
```

Exporter l'image Docker :

```
docker save churn-api:v1 -o churn-api_v1.tar
```

Charger l'image dans Minikube :

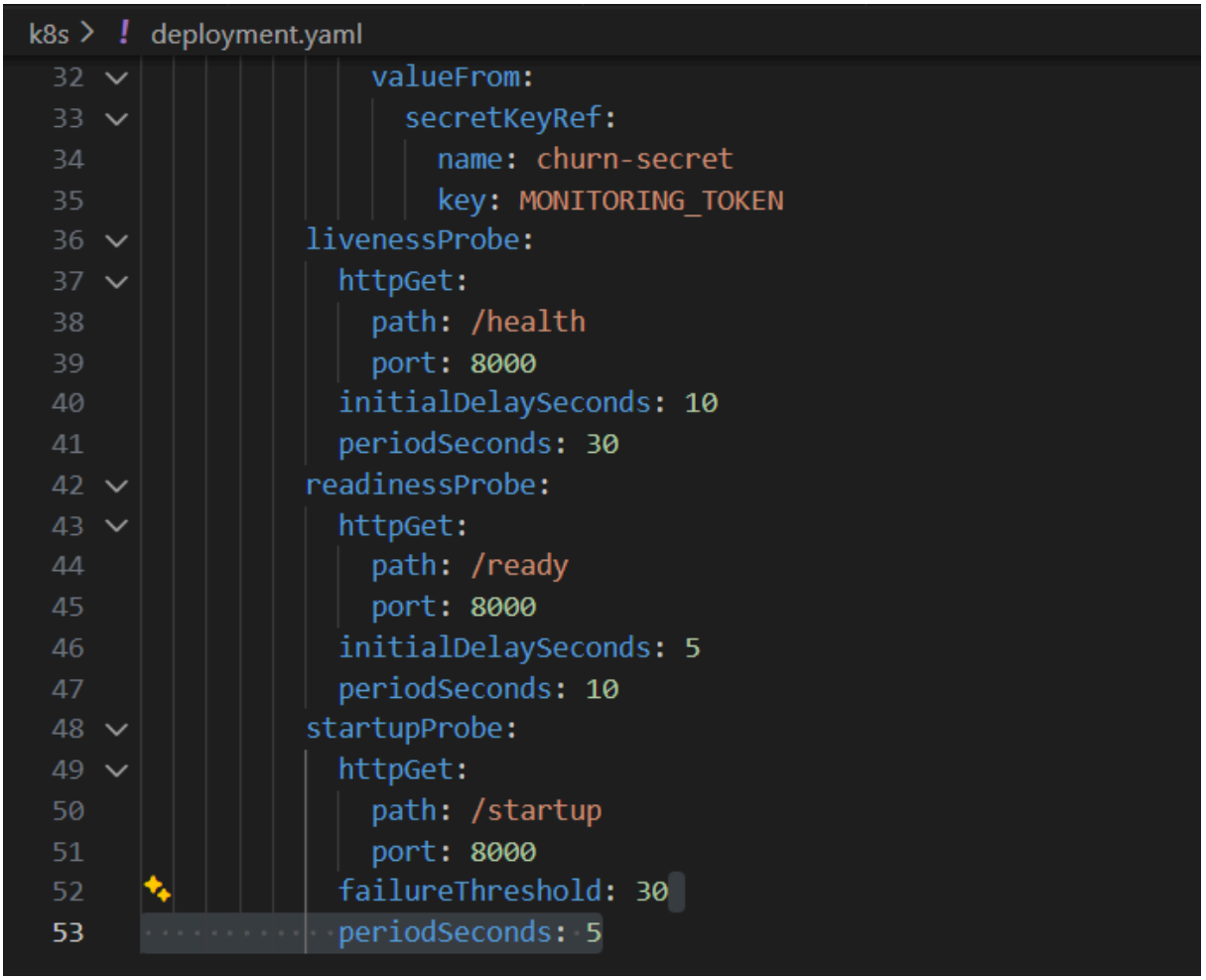
```
minikube image load churn-api_v1.tar
```



Étape 11 : Ajouter les probes (liveness / readiness / startup)

Instructions :

1. Dans `k8s/deployment.yaml`, compléter le conteneur :



2. Réappliquer :

```
kubectl apply -f k8s/deployment.yaml
kubectl describe pod -l app=churn-api
```

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl apply -f k8s/deployment.yaml
deployment.apps/churn-api configured
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl describe pod -l app=churn-api
Name:             churn-api-5cc7f85545-gw2rd
Namespace:        default
Priority:         0
Service Account:  default
Node:             minikube/192.168.49.2
Start Time:       Thu, 15 Jan 2026 17:35:14 +0100
Labels:           app=churn-api
                  pod-template-hash=5cc7f85545
Annotations:      kubectl.kubernetes.io/restartedAt: 2026-01-15T16:55:58+01:00
Status:           Running
IP:               10.244.0.13
IPs:
  IP:           10.244.0.13
Controlled By:  ReplicaSet/churn-api-5cc7f85545
Containers:
  api:
    Container ID:   docker://643e981d5ba14eca016a319b3f831c609cfe54ddcf9ddd18924e2309b2a157e4
    Image:          churn-api:v1
```

Redéployer l'application dans le cluster Kubernetes :

```
kubectl rollout restart deployment churn-api
kubectl rollout status deployment churn-api
```

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl rollout restart deployment churn-api
deployment.apps/churn-api restarted
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl rollout status deployment churn-api
Waiting for deployment "churn-api" rollout to finish: 1 out of 2 new replicas have been updated...
Waiting for deployment "churn-api" rollout to finish: 1 out of 2 new replicas have been updated...
Waiting for deployment "churn-api" rollout to finish: 1 out of 2 new replicas have been updated...
Waiting for deployment "churn-api" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "churn-api" rollout to finish: 1 old replicas are pending termination...
deployment "churn-api" successfully rolled out
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Vérifier l'état des Pods :

```
kubectl get pods
```

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl get pods
NAME                         READY   STATUS    RESTARTS   AGE
churn-api-f6fd88cc4-7ll5j    1/1     Running   0          3m23s
churn-api-f6fd88cc4-tjqdf    1/1     Running   0          2m19s
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Objectif :

Kubernetes redémarre un Pod si `/health` ne répond plus,et n'envoie du trafic qu'aux Pods ready.

Étape 12 : Volume persistant pour registry + logs

Instructions :Créer le PVC (PersistentVolumeClaim) : ce volume persistant permet de conserver les fichiers du modèle et des logs même si les Pods sont recréés.
Créer le fichier :

`New-Item k8s/pvc.yaml`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> New-Item k8s/pvc.yaml


    Directory: C:\Users\anoua\projects\MLOPS\mlops-lab-01\k8s


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         1/15/2026   7:37 PM              0 pvc.yaml
```

Coller :

k8s > ! pvc.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: churn-storage
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

Appliquer :

`kubectl apply -f k8s/pvc.yaml`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl apply -f k8s/pvc.yaml
persistentvolumeclaim/churn-storage created
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Vérifier que le PVC est bien créé et associé à un volume :

`kubectl get pvc`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl get pvc
NAME            STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS
   AGE
churn-storage   Bound    pvc-bdf3ed8d-b164-4ee7-b2cd-4725bccefd7c   5Gi        RWO            standard
   76s
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Créer et exécuter le Job d'entraînement : ce Job initialise le contenu du PVC en entraînant un premier modèle et en écrivant les artefacts dans `/app/registry`. Créer le fichier :

`New-Item k8s/job-train.yaml`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> New-Item k8s/job-train.yaml


    Directory: C:\Users\anoua\projects\MLOPS\mlops-lab-01\k8s


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         1/15/2026   7:39 PM              0 job-train.yaml
```

Coller :

k8s > ! job-train.yaml

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: churn-train
spec:
  backoffLimit: 1
  template:
    spec:
      restartPolicy: Never
      volumes:
        - name: churn-volume
          persistentVolumeClaim:
            claimName: churn-storage
      containers:
        - name: train
          image: churn-api:v1
          command: ["python", "src/train.py"]
          volumeMounts:
            - name: churn-volume
              mountPath: /app/models
              subPath: models
            - name: churn-volume
              mountPath: /app/registry
              subPath: registry
```

Appliquer le Job :

`kubectl apply -f k8s/job-train.yaml`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl apply -f k8s/job-train.yaml
job.batch/churn-train created
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Attendre la fin du Job :

`kubectl wait --for=condition=complete job/churn-train`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl wait --for=condition=complete job/churn
-train
job.batch/churn-train condition met
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Monter le PVC dans le Deployment : l'API doit lire le modèle courant et écrire ses logs dans le même stockage persistant.
Modifier `k8s/deployment.yaml` pour ajouter le volume dans `spec.template.spec`

Dans la section `containers: - name: api`, monter ce volume à deux emplacements distincts :

- **`/app/registry` pour stocker et lire les modèles,**
- **`/app/logs` pour écrire les logs applicatifs**

```
k8s > ! deployment.yaml
45                port: 8000
46              initialDelaySeconds: 5
47              periodSeconds: 10
48            startupProbe:
49              httpGet:
50                path: /startup
51                port: 8000
52              failureThreshold: 30
53              periodSeconds: 5
54            volumeMounts:
55              - name: churn-volume
56                mountPath: /app/registry
57                subPath: registry
58              - name: churn-volume
59                mountPath: /app/models
60                subPath: models
61              - name: churn-volume
62                mountPath: /app/logs
63                subPath: logs
64          volumes:
65            - name: churn-volume
66              persistentVolumeClaim:
67                claimName: churn-storage
```

Réappliquer :

`kubectl apply -f k8s/deployment.yaml`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl apply -f k8s/deployment.yaml
deployment.apps/churn-api configured
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Vérifier que les Pods redémarrent correctement :

`kubectl get pods`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl get pods
NAME                         READY   STATUS      RESTARTS   AGE
churn-api-7655fd649b-tg6fz   1/1     Running     0          5m50s
churn-api-7655fd649b-vzd4l   1/1     Running     0          6m50s
churn-train-6kzf6            0/1     Completed   0          9m30s
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Vérifier que les dossiers sont bien accessibles depuis un Pod :

```
kubectl exec -it deploy/churn-api -- ls /app/registry
kubectl exec -it deploy/churn-api -- ls /app/logs
```

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl exec -it deploy/churn-api -- ls /app/re
gistry
current_model.txt  metadata.json
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl exec -it deploy/churn-api -- ls /app/lo
gs
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Étape 13 : NetworkPolicy

Instructions :

1. Créer le fichier :

```
New-Item k8s/networkpolicy.yaml
```

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> New-Item k8s/networkpolicy.yaml


    Directory: C:\Users\anoua\projects\MLOPS\mlops-lab-01\k8s


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         1/15/2026   7:54 PM              0 networkpolicy.yaml
```

2. Coller le contenu suivant :

k8s > ! networkpolicy.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-internal-services
spec:
  podSelector:
    matchLabels:
      app: churn-api
  policyTypes:
    - Ingress
  ingress:
    - from:
        - podSelector: {}
      ports:
        - port: 8000
          protocol: TCP
```

3. Appliquer la configuration :

`kubectl apply -f k8s/networkpolicy.yaml`
`kubectl get networkpolicy`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl apply -f k8s/networkpolicy.yaml
networkpolicy.networking.k8s.io/allow-internal-services created
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl get networkpolicy
NAME                       POD-SELECTOR    AGE
allow-internal-services    app=churn-api   4s
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

Étape 14 : Vérifications finales

Instructions :

1. Vérifier les Pods et les Services :

`kubectl get pods -l app=churn-api`
`kubectl get svc`

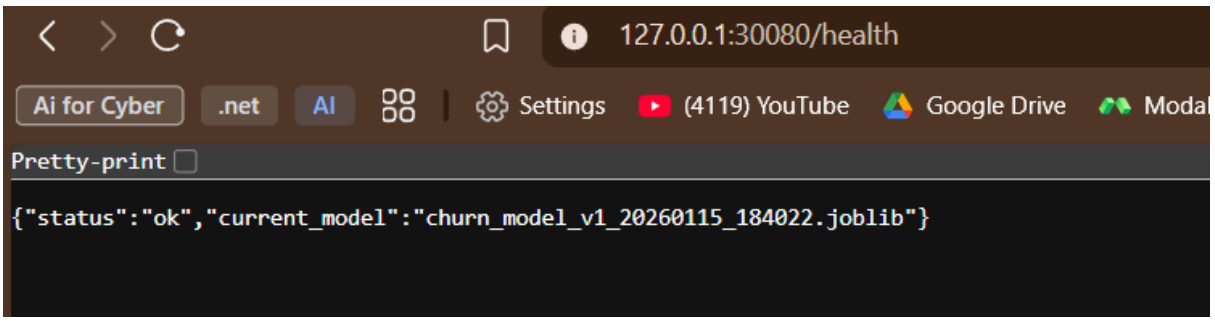
```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl get pods -l app=churn-api
NAME                         READY   STATUS    RESTARTS   AGE
churn-api-7655fd649b-tg6fz   1/1     Running   0          12m
churn-api-7655fd649b-vzd4l   1/1     Running   0          13m
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl get svc
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
churn-api-service   NodePort    10.100.129.224   <none>        80:30080/TCP   3h22m
kubernetes          ClusterIP   10.96.0.1        <none>        443/TCP        29h
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

2. Tester l'endpoint /health :

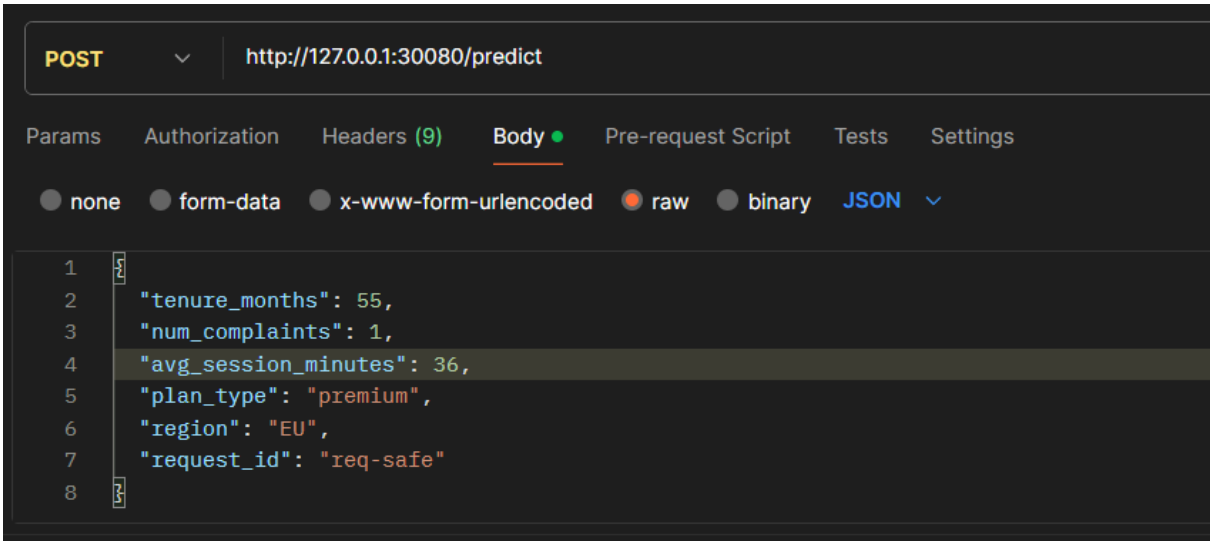
`kubectl port-forward svc/churn-api-service 30080:80`

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl port-forward svc/churn-api-service 3008
0:80
Forwarding from 127.0.0.1:30080 -> 8000
Forwarding from [::1]:30080 -> 8000
```

`GET http://127.0.0.1:30080/health`

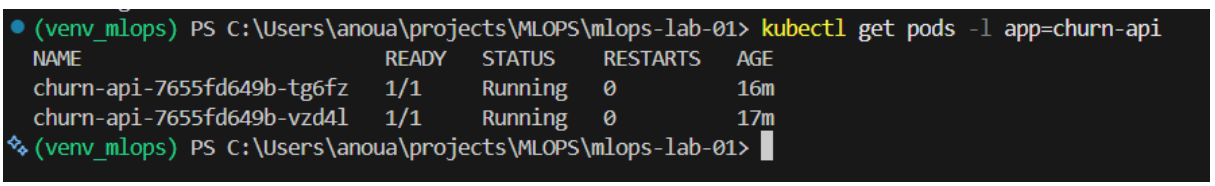


3. Envoyer quelques requêtes /predict.



4. Lister les Pods pour choisir un Pod churn :

`kubectl get pods -l app=churn-api`



5. Exécuter la détection de drift dans le Pod :

`kubectl exec -it <nom_du_pod> -- python src/monitor_drift.py`

poblem :

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl exec -it churn-api-7655fd649b-tg6fz -c
api -- python src/monitor_drift.py
Traceback (most recent call last):
  File "/app/src/monitor_drift.py", line 193, in <module>
    main()
  File "/app/src/monitor_drift.py", line 81, in main
    raise FileNotFoundError(
FileNotFoundError: train_stats.json introuvable. Lancer d'abord prepare_data.py pour générer les statistiqu
es.
command terminated with exit code 1
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```

- Result: Drift check completed with 1 recent request; no drift detected.
- Files present: train_stats.json, processed.csv, logs used from predictions.log.
- PVC mounts: Verified in the pod under /app/registry.

```
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> $pod = "churn-api-7655fd649b-tg6fz"
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl exec -it $pod -c api -- python src/prep
are_data.py
[OK] Fichier prétraité généré : /app/data/processed.csv
[OK] Statistiques d'entraînement générées : /app/registry/train_stats.json
HElloo
cheeck
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl exec -it $pod -c api -- ls -la /app/reg
istry
total 24
drwxrwxrwx 2 root root 4096 Jan 15 19:07 .
drwxr-xr-x 1 root root 4096 Jan 15 16:39 ..
-rw-r--r-- 1 root root   37 Jan 15 18:40 current_model.txt
-rw-r--r-- 1 root root  409 Jan 15 18:40 metadata.json
-rw-r--r-- 1 root root  271 Jan 15 19:07 train_stats.json
>>          PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01> kubectl exec -it $pod -c api -- python src/moni
tor_drift.py
=== Drift check sur 1 requêtes récentes ===
- tenure_months: mean_prod=55.000 | mean_train=30.246 | z=1.458
- num_complaints: mean_prod=1.000 | mean_train=1.174 | z=0.157
- avg_session_minutes: mean_prod=36.000 | mean_train=35.124 | z=0.074
Résultat : aucun drift détecté.
(venv_mlops) PS C:\Users\anoua\projects\MLOPS\mlops-lab-01>
```