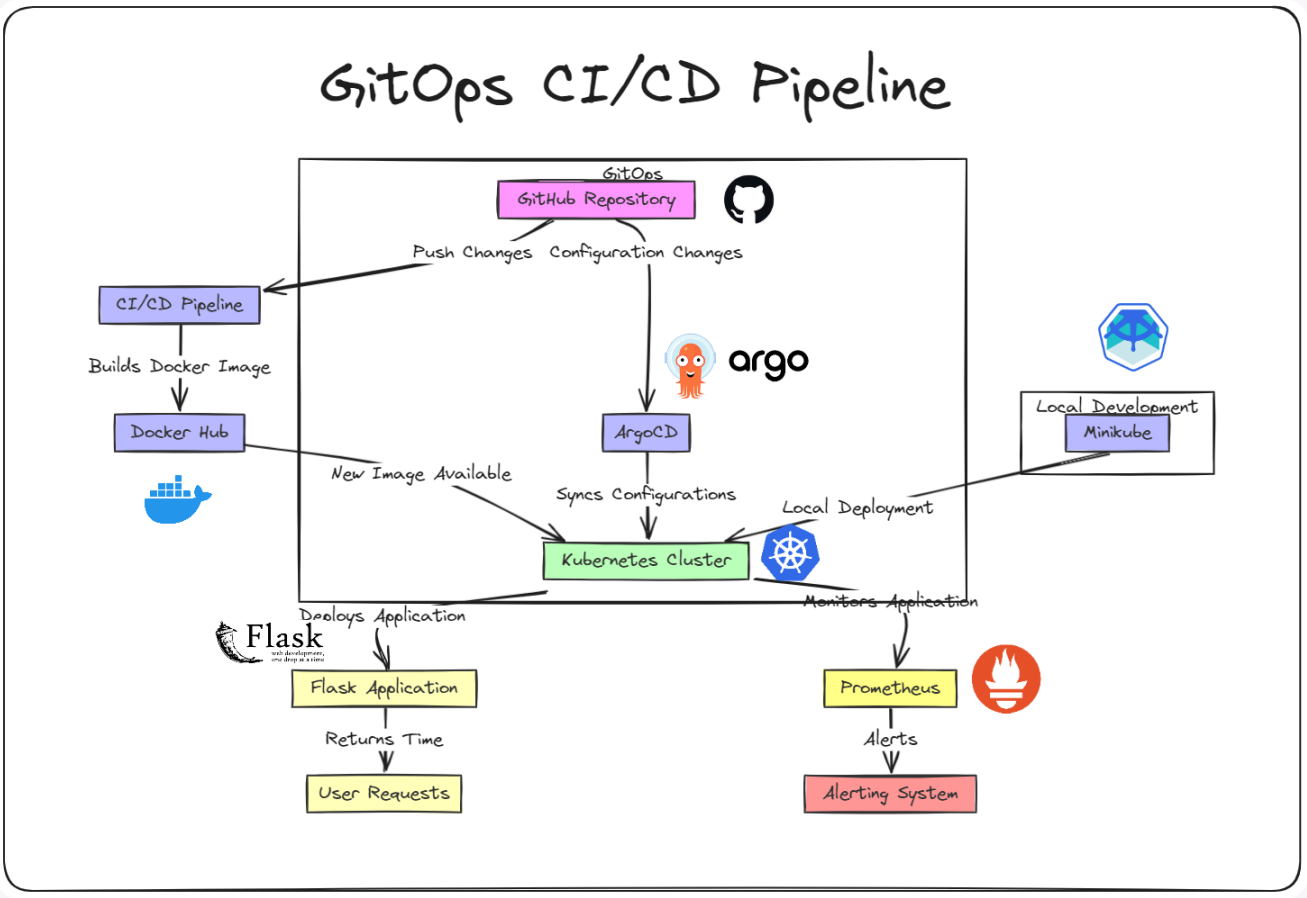
TP1 GitOps

> This lab was accomplished by the following team members:
>
> `Sandra Mourali` | `Mahmoud Hedi Nefzi` | `Omar Besbes`

The GitHub repository link: TP1AdvancedDevops

0. Technologies used :



1. Setting up the necessary configurations:

Our app is a simple Flask API that responds with a JSON message that includes a greeting and the current time.

1. First we included all the requirements in `requirements.txt`
2. We created a Dockerfile for Containerization

```
FROM python:3.8-slim
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
EXPOSE 5000
CMD ["python", "app.py"]
```

unkown language

4. We built and Pushed the Docker Image using the following command :

```bash
docker build -t sandramourali/helloworld-flask .
```

Bash

```
[+] Building 3.9s (2/3)  docker:de[+] Building 4.0s (3/3)  docker:de[+] Buildi
ng 4.1s (4/4)  docker:de[+] Building 4.2s (5/9)  docker:de[+] Building 4.3s (4
/9)  docker:de[+] Building 4.4s (4/9)  docker:de[+] Building 4.5s (4/9)  docke
r:de[+] Building 4.6s (4/9)  docker:de[+] Building 4.7s (4/9)  docker:de[+] Bu
ilding 4.9s (4/9)  docker:de[+] Building 5.0s (4/9)  docker:de[+] Building 5.1
[+] Building 12.3s (5/9)                                        docker:default
 => [internal] load build definition from dockerfile                      0.2s
 => => transferring dockerfile: 162B                                      0.1s
 => [internal] load metadata for docker.io/library/python:3.8-slim        3.7s
 => [auth] library/python:pull token for registry-1.docker.io             0.0s
 => [internal] load .dockerignore                                         0.1s
 => => transferring context: 2B                                           0.0s
 => [1/4] FROM docker.io/library/python:3.8-slim@sha256:1d52838af602b4b   8.2s
 => => resolve docker.io/library/python:3.8-slim@sha256:1d52838af602b4b   0.1s
 => => sha256:a3f1dfe736c5f959143f23d75ab522a60be2da90 3.15MB / 14.53MB   8.0s
 => => sha256:1d52838af602b4b5a831beb13a0e4d073280665 10.41kB / 10.41kB   0.0s
 => => sha256:314bc2fb0714b7807bf5699c98f0c73817e579799 1.75kB / 1.75kB   0.0s
 => => sha256:b5f62925bd0f63f48cc8acd5e87d0c3a07e2f229c 5.25kB / 5.25kB   0.0s
 => => sha256:302e3ee498053a7b5332ac79e8efebec16e90028 2.10MB / 29.13MB   8.0s
 => => sha256:030d7bdc20a63e3d22192b292d006a69fa3333949 1.05MB / 3.51MB   8.0s
 => [internal] load build context                                         4.4s
 => => transferring context: 160.24MB                                     4.3s
```

```yaml
docker push sandramourali/helloworld-flask
```

YAML

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    COMMENTS

 => => transferring context: 160.24MB                                  4.3s
 => [2/4] WORKDIR /app                                                 0.4s
 => [3/4] COPY . /app                                                  0.8s
 => [4/4] RUN pip install -r requirements.txt                         10.7s
 => exporting to image                                                 0.9s
 => => exporting layers                                                0.8s
 => => writing image sha256:a1c148047cf8b44d66aecda8558bc0c49c1f3f3cba0  0.0s
 => => naming to docker.io/sandramourali/helloworld-flask              0.0s
root@SandraMourali:/mnt/c/Users/HP OMEN/Desktop/Nouveau dossier# docker login
Authenticating with existing credentials...
Login Succeeded
root@SandraMourali:/mnt/c/Users/HP OMEN/Desktop/Nouveau dossier# docker tag he
lloworld-flask sandramourali/helloworld-flask:latest
docker push sandramourali/helloworld-flask:latest
Error response from daemon: No such image: helloworld-flask:latest
The push refers to repository [docker.io/sandramourali/helloworld-flask]
711fe7f87058: Pushed
9ae1bb52608a: Pushing    127MB/160.2MB
7c956e4e8fb4: Pushed
d2a2207b52a4: Mounted from library/python
5d2d143f3d7f: Mounted from library/python
c3772b569c3a: Mounted from library/python
8d853c8add5d: Mounted from library/python

⊗ 0 ⚠ 0   0        Ln 2, Col 1   Spaces: 4   UTF-8   CRLF   pip requirements
```

5. We set up Kubernetes with Minikube

```bash
minikube start
```

6. We created Kubernetes YAML Files

`deployment.yaml`

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-deployment
spec:
  replicas: 2
  selector:
```

```yaml
    matchLabels:
      app: flask-app
  template:
    metadata:
      labels:
        app: flask-app
    spec:
      containers:
      - name: flask-container
        image: sandramourali/helloworld-flask:latest
        ports:
        - containerPort: 5000
```

`service.yaml`

```yaml
apiVersion: v1
kind: Service
metadata:
  name: flask-service
spec:
  selector:
    app: flask-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
  type: LoadBalancer
```

2.Setting up Minikube and Kubernetes

We applied the YAML files we created to the Kubernetes Cluster :

```bash
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml
```

To test whether our configurations were correctly applied we used:

```shell
kubectl get deployments
kubectl get pods
kubectl get services
```

```
root@SandraMourali:/mnt/c/Users/HP OMEN/Desktop/Nouveau dossier# kubectl get deployments
kubectl get pods
kubectl get services
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
flask-deployment     0/2     2            0           58m
NAME                                  READY   STATUS             RESTARTS         AGE
flask-deployment-7fcd78dd8c-dq7h2     0/1     CrashLoopBackOff   14 (3m25s ago)   58m
flask-deployment-7fcd78dd8c-ffxjg     0/1     CrashLoopBackOff   14 (3m23s ago)   58m
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
flask-service   LoadBalancer   10.108.64.196   localhost     80:30231/TCP   58m
kubernetes      ClusterIP      10.96.0.1       <none>        443/TCP        53d
root@SandraMourali:/mnt/c/Users/HP OMEN/Desktop/Nouveau dossier#
```

3.GitOps with ArgoCD

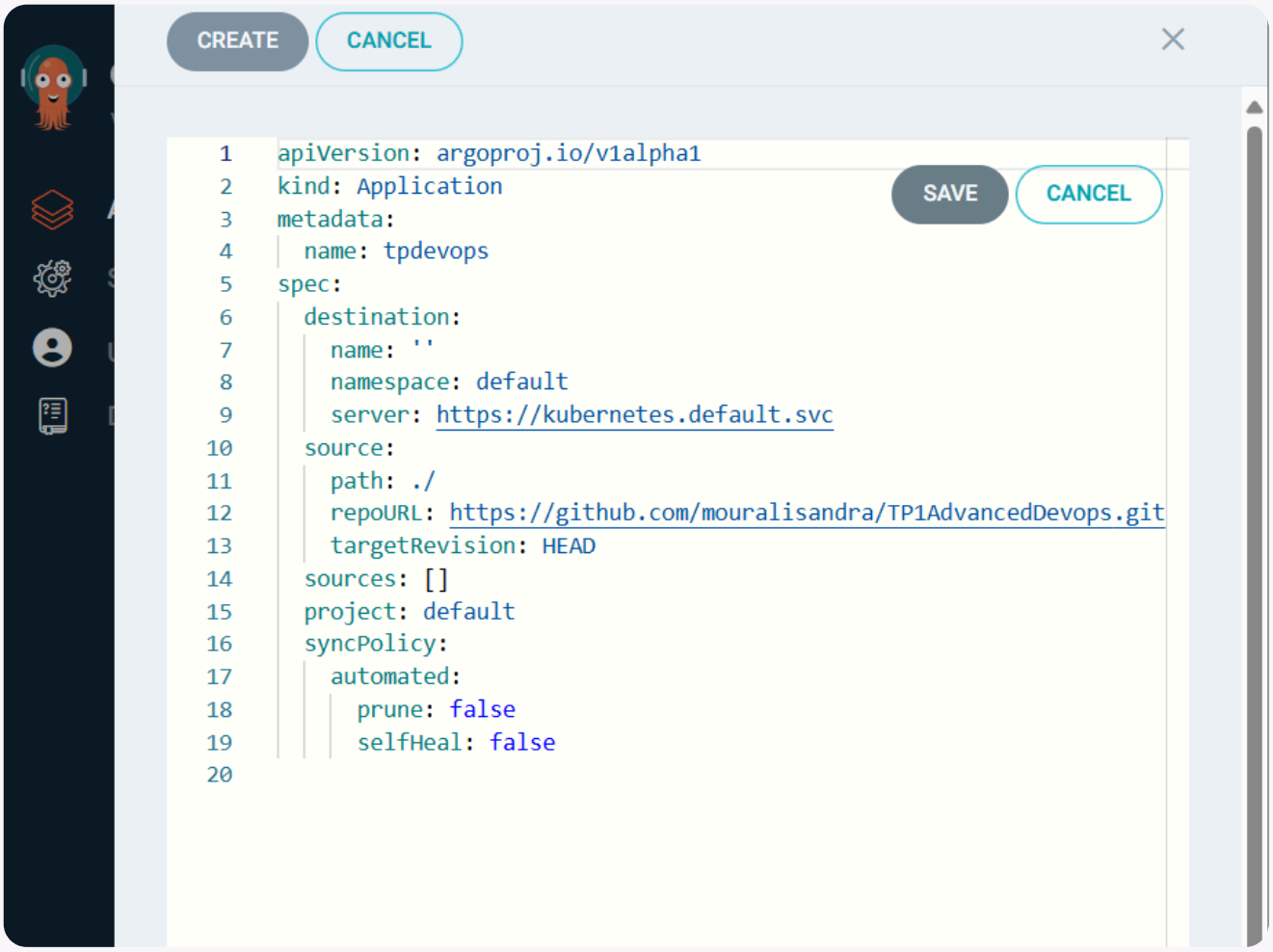
ArgoCD is one of the best tools to automate deployments as everytime we push changes to the repository, ArgoCD will automatically deploy the new version of our Flask app.

1. First we installed and set up ArgoCd :

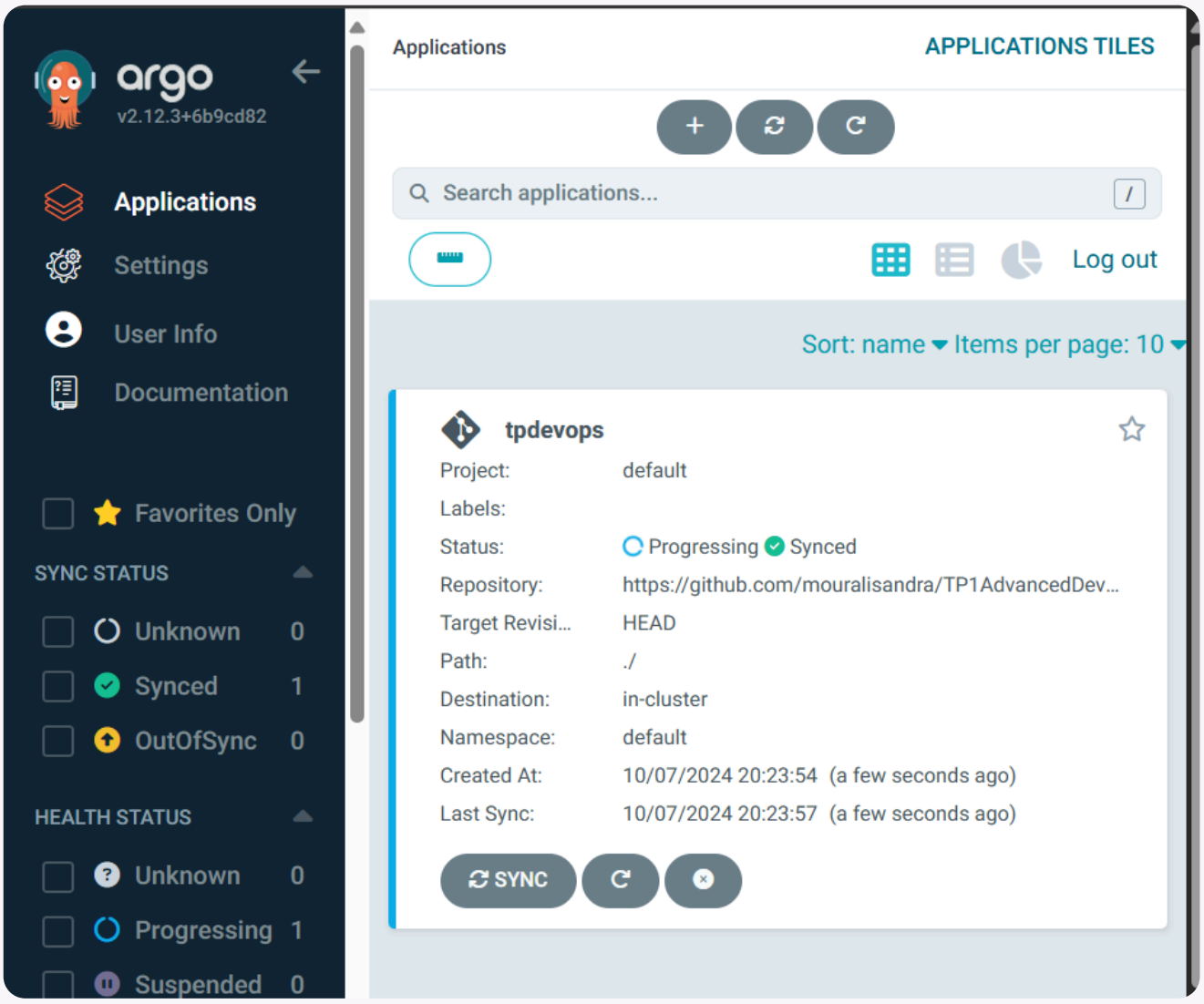
```shell
kubectl port-forward svc/argocd-server -n argocd 8080:443
```

```
root@SandraMourali:/mnt/c/Users/HP OMEN/Desktop/Nouveau dossier# kubectl port-forward svc/argocd-server -n argocd 8080:443
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
```

3. Then we created and configured our project via ArgoCD's UI :

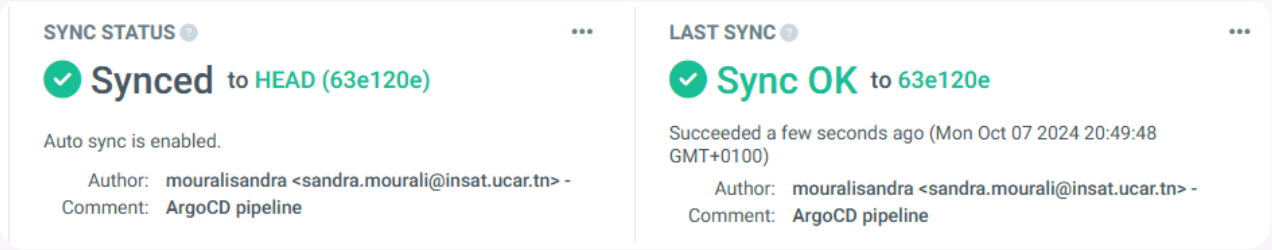


2. Once the configurations done, we synced up with our Github Repository :



4. Monitoring and alerts:

1. For the monitoring part, ArgoCD already does the monitoring as it provides basic health status indicators for applications. It shows whether an application is Healthy, Progressing, Degraded, or Suspended. This information can be viewed in the Argo CD UI.



2. For the alerts part, we integrated Prometheus with ArgoCd:

   We created an alerts pipeline with Prometheus by setting up the following `alert.yaml` config:

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: example-alert-rules
  namespace: monitoring
spec:
  groups:
    - name: example
      rules:
        - alert: HighCPUUsage
          expr:
avg(rate(container_cpu_usage_seconds_total{job="kubelet"}[5m])) by
(instance) > 0.5
          for: 5m
          labels:
            severity: warning
          annotations:
            summary: "High CPU Usage on {{ $labels.instance }}"
            description: "CPU usage is above 50% for more than 5
minutes."
```

We applied the alert using the following command :

```shell
kubectl apply -f alert.yaml
```

Then we restarted the deployment to apply changes made :

```shell
kubectl rollout restart deployment prometheus-kube-prometheus-
alertmanager -n monitoring
```

Afterwards we synced with ArgoCD

```shell
argocd app sync prometheus-app
```