# Kubernetes Lab Report

## Autors: Jaquet Steven Marrino Jarrin Gabriel Ngueukam Djeuda Wilfried Karel

October 2025

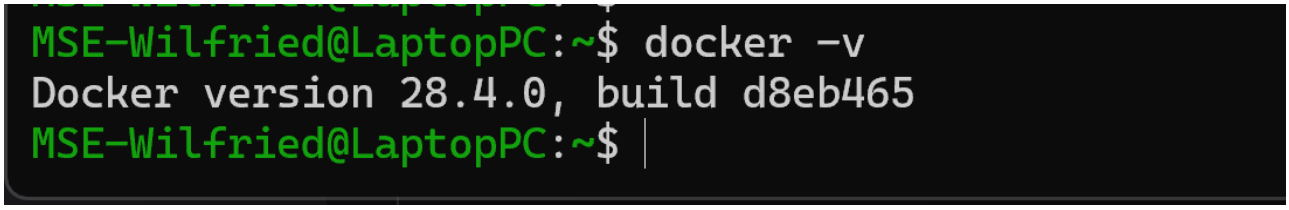
Course : TSM_CloudSys

Instructors:

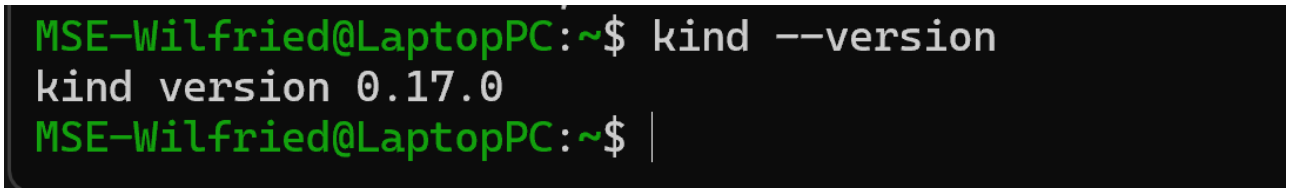
- White John (Teacher)
- Chebbi Abir (Assistant)

### Set up Environment

- Ensure docker is installed:




- Ensure Kind is installed




- Create the kind configuration file: We create a simple cluster configuration file containing as requested 1 control-plane and 5 workers. The contains of the file is given below:

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
- role: worker
- role: worker
- role: worker
- role: worker
- role: worker
```

- Creating cluster: We use the following command to create our cluster: `kind create cluster --config kind-config-cluster.yml`. As result, we are getting the following output:

```
MSE-Wilfried@LaptopPC:~/courses/CloudSys/labs/k8s-lab$ kind create cluster --config kind-config-cluster.yml
Creating cluster "kind" ...
 ✓ Ensuring node image (kindest/node:v1.25.3) 🖼
 ✓ Preparing nodes 📦 📦 📦 📦 📦 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
 ✓ Joining worker nodes 🚜
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Have a nice day! 👋
```

We can check our cluster is created using command `kind get clusters` and `docker ps` to list docker's containers created for our cluster.

```
MSE-Wilfried@LaptopPC:~/courses/CloudSys/labs/k8s-lab$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS         PORTS                       NAMES
d08dad3e1b3c   kindest/node:v1.25.3   "/usr/local/bin/entr…"   3 minutes ago   Up 3 minutes                               kind-worker5
933ab3714de5   kindest/node:v1.25.3   "/usr/local/bin/entr…"   3 minutes ago   Up 3 minutes                               kind-worker4
3b658ae2137f   kindest/node:v1.25.3   "/usr/local/bin/entr…"   3 minutes ago   Up 3 minutes                               kind-worker3
656a207c5371   kindest/node:v1.25.3   "/usr/local/bin/entr…"   3 minutes ago   Up 3 minutes                               kind-worker2
92c470870b88   kindest/node:v1.25.3   "/usr/local/bin/entr…"   3 minutes ago   Up 3 minutes                               kind-worker
416c5d90564b   kindest/node:v1.25.3   "/usr/local/bin/entr…"   3 minutes ago   Up 3 minutes   127.0.0.1:39409->6443/tcp   kind-control-plane
```

- Setting `kubectl` to use our cluster with `kubectl cluster-info --context kind-kind` since the default name for our cluster is ***kind***

```
MSE-Wilfried@LaptopPC:~/courses/CloudSys/labs/k8s-lab$ kubectl cluster-info --context kind-kind
Kubernetes control plane is running at https://127.0.0.1:39409
CoreDNS is running at https://127.0.0.1:39409/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

## Task 1 : Redis deployment

### 1. complete redis-deployment.yaml file

To deploy our redis pod, we complete the *redis-deployement.yaml* file with necessary informations. The first one is the `spec.selector.matchLabels` property. We add an `app: redis-pod` property to indicate at this deployment to manage pods with label `app: redis-pod`. We use the same logic in service configuration and add `app:redis-pod` to `spec.selector` property. This is how our file now looks like:

```
...
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis-pod
...
spec:
  selector:
    app: redis-pod
  ports:
...
```

### 2. Deployment of redis pod

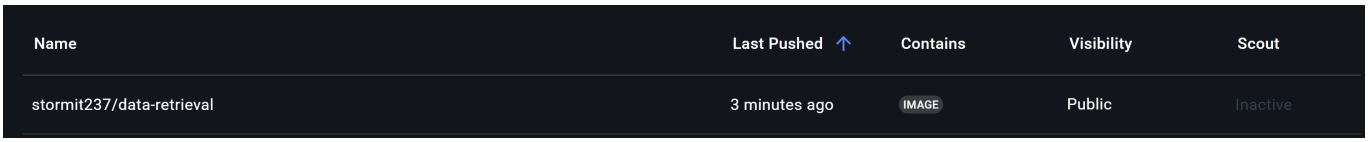
We now applying our deployment with command `kubectl apply -f deployment\redis-deployment.yaml`and getting the following output:

```
MSE-Wilfried@LaptopPC:~/courses/CloudSys/labs/k8s-lab$ kubectl apply -f deployment/redis-deployment.yaml
deployment.apps/redis-deployment created
service/redis created
```

## Task 2 : Data-retrieval deployment

### 1. Building image and push it to our dockerhub account

For this part, we browse to `deployment\data-retrieval\Dockerfile` and we executed the command `docker build -t stormit237/data-retrieval .`. After a few seconds we have executed `docker images` to check our images is created and then we make a `docker push stormit237/data-retrieval` to push our image to dockerhub. As result, on our dockerhub account, we could see the image below :




### 2. Complete data-retrieval-deployment.yaml file

In this part, we complete missing information to deploy data-retrieval job. We had our docker image from our dockerhub account, the host of the redis services `redis.default.svc.cluster.local` our AWS_ACCESS_KEY_ID and our AWS_SECRET_ACCESS_KEY. The final file is similar to :

```
spec:
      containers:
      - name: data-retrieval
        image: stormit237/data-retrieval  # Replace with your Docker image name
        imagePullPolicy: Always        # Force the pull of the container image - used for testing purposes
        env:
        - name: REDIS_HOST
          value: redis.default.svc.cluster.local  # The host of the Redis service
        - name: AWS_ACCESS_KEY_ID
          value: "...EKYIBTQ....U6H"  # Fill in with the Secret key value
        - name: AWS_SECRET_ACCESS_KEY
          value: "...+ZOWL1xnbFVs5VQDcEX"  # Fill in with the Secret key value
      restartPolicy: Never
```

### 3. Deploy the data-retrieval module

To deploy the data-retrieval module, we use the command `kubectl apply -f deployment\data-retrieval.deployment.yaml`. Then with the command `kubectl logs job/data-retrieval-job` we can

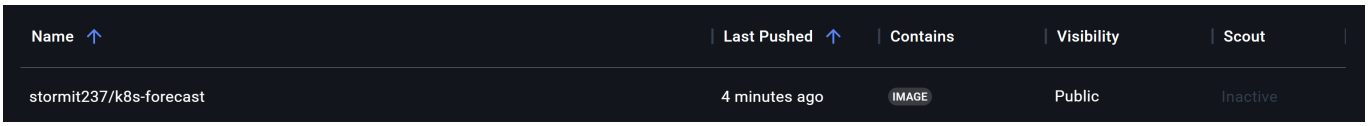
see the process executed successfully.

```
MSE-Wilfried@LaptopPC:~/courses/CloudSys/labs/k8s-lab$ kubectl logs job/data-retrieval-job
2025-11-02 23:06:05,144 INFO Connected to Redis: redis.default.svc.cluster.local
2025-11-02 23:06:05,156 INFO Successfully connected to Redis and performed a test operation.
2025-11-02 23:06:05,412 INFO Reading CSV from S3 bucket: k8s-class-2024
2025-11-02 23:06:06,278 INFO Data read from CSV successfully.
2025-11-02 23:06:06,278 INFO Number of columns = 11
2025-11-02 23:06:06,278 INFO Writing dataset to RedisTimeSeries
2025-11-02 23:06:06,361 INFO Writing data for device: ts:424661
2025-11-02 23:06:07,654 INFO Writing data for device: ts:424665
2025-11-02 23:06:08,942 INFO Writing data for device: ts:424668
2025-11-02 23:06:10,259 INFO Writing data for device: ts:424670
2025-11-02 23:06:12,123 INFO Writing data for device: ts:424671
2025-11-02 23:06:13,436 INFO Writing data for device: ts:424674
2025-11-02 23:06:14,731 INFO Writing data for device: ts:424675
2025-11-02 23:06:16,143 INFO Writing data for device: ts:424676
2025-11-02 23:06:17,421 INFO Writing data for device: ts:424677
2025-11-02 23:06:18,702 INFO Writing data for device: ts:424678
2025-11-02 23:06:19,995 INFO Finished writing dataset to RedisTimeSeries
```

## Task 3. Forecast deployment

### 1. Build and push container

To build container, we executed the command : `docker build -t stormit237/k8s-forecast .`. Once the command end with success, we push to our dockerhub repository using the command : `docker push stormit237/k8s-forecast`. As result we are getting the repository save to our dockerhub account :




### 2. Fill the forecast-deployment.yaml file

We complete this by providing `app: forecast-pod` as label in `template.metadata.labels`. and providing informations for the redis host `value: redis.default.svc.cluster.local`. The final file for this step looks like

```
...
 template:
    metadata:
      labels:
        app: forecast-pod  # Fill with the correct label
    spec:
      containers:
      - name: forecast
        image: stormit237/k8s-forecast:latest  # Replace with your Docker image
name
        imagePullPolicy: Always        # Force the pull of the container image -
used for testing purposes
        env:
        - name: REDIS_HOST
          value: redis.default.svc.cluster.local  # The host of the Redis service
...
```

### 3. Deploy the forecast module

To deploy this module, we use command `kubectl apply -f deployment\forecast-deployment.yaml`. We then make a `kubectl get pods` to check if our command executed successfully. After a few minutes we get this output:

```
MSE-Wilfried@LaptopPC:~/courses/CloudSys/labs/k8s-lab$ kubectl get pods
NAME                                  READY   STATUS      RESTARTS   AGE
data-retrieval-job-24rp9              0/1     Completed   0          79m
dnsutils                              1/1     Running     0          5h10m
forecast-deployment-5c57bdbb8d-6cm5l  1/1     Running     0          18m
forecast-deployment-5c57bdbb8d-87q2q  1/1     Running     0          18m
forecast-deployment-5c57bdbb8d-fdnfb  1/1     Running     0          18m
forecast-deployment-5c57bdbb8d-hpjsk  1/1     Running     0          18m
forecast-deployment-5c57bdbb8d-k88d5  1/1     Running     0          18m
forecast-deployment-5c57bdbb8d-kkd8r  1/1     Running     0          18m
forecast-deployment-5c57bdbb8d-lr8hq  1/1     Running     0          18m
forecast-deployment-5c57bdbb8d-rbf4g  1/1     Running     0          18m
forecast-deployment-5c57bdbb8d-sd479  1/1     Running     0          18m
forecast-deployment-5c57bdbb8d-wxrs4  1/1     Running     0          18m
redis-deployment-844c57b765-ksdpx     1/1     Running     0          11h
```

We can see we have 10 pods for the 10 replicas configured.

## Task 4. Deploy Grafana

We complete this task by providing a property `app: grafana-pod` at `spec.selector` and running command `kubectl apply -f deployment\grafana-deployment.yaml`. After a few minutes, we run command `kubectl get pods` and we get this result :

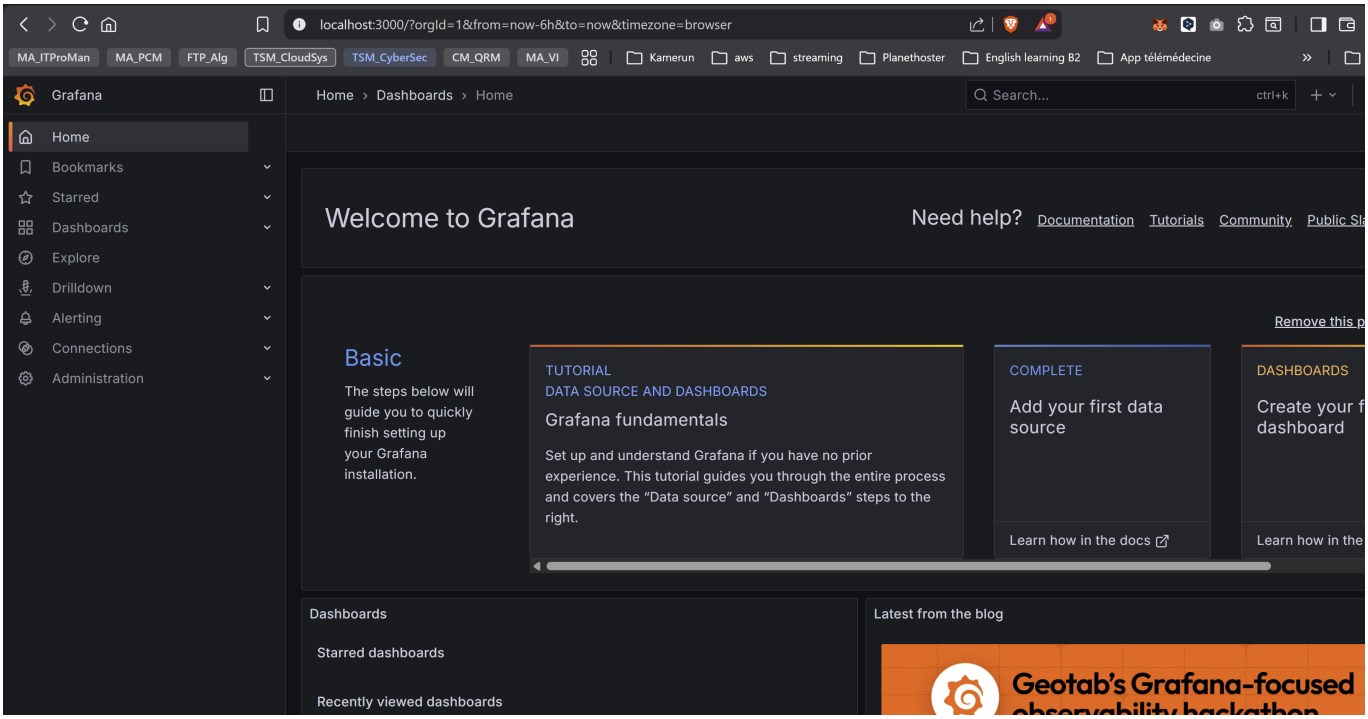
```
MSE-Wilfried@LaptopPC:~/courses/CloudSys/labs/k8s-lab$ kubectl get pods
NAME                                  READY   STATUS      RESTARTS   AGE
data-retrieval-job-24rp9              0/1     Completed   0          92m
dnsutils                              1/1     Running     0          5h23m
forecast-deployment-5c57bdbb8d-6cm5l  1/1     Running     0          31m
forecast-deployment-5c57bdbb8d-87q2q  1/1     Running     0          31m
forecast-deployment-5c57bdbb8d-fdnfb  1/1     Running     0          31m
forecast-deployment-5c57bdbb8d-hpjsk  1/1     Running     0          31m
forecast-deployment-5c57bdbb8d-k88d5  1/1     Running     0          31m
forecast-deployment-5c57bdbb8d-kkd8r  1/1     Running     0          31m
forecast-deployment-5c57bdbb8d-lr8hq  1/1     Running     0          31m
forecast-deployment-5c57bdbb8d-rbf4g  1/1     Running     0          31m
forecast-deployment-5c57bdbb8d-sd479  1/1     Running     0          31m
forecast-deployment-5c57bdbb8d-wxrs4  1/1     Running     0          31m
grafana-deployment-64d8fcbf9c-7b8m8   1/1     Running     0          3m41s
grafana-deployment-64d8fcbf9c-gzspq   1/1     Running     0          3m41s
grafana-deployment-64d8fcbf9c-t8kjl   1/1     Running     0          3m41s
redis-deployment-844c57b765-ksdpx     1/1     Running     0          11h
```

We can see on this picture 3 pods are created for grafana.

## Task 5. Load balancer

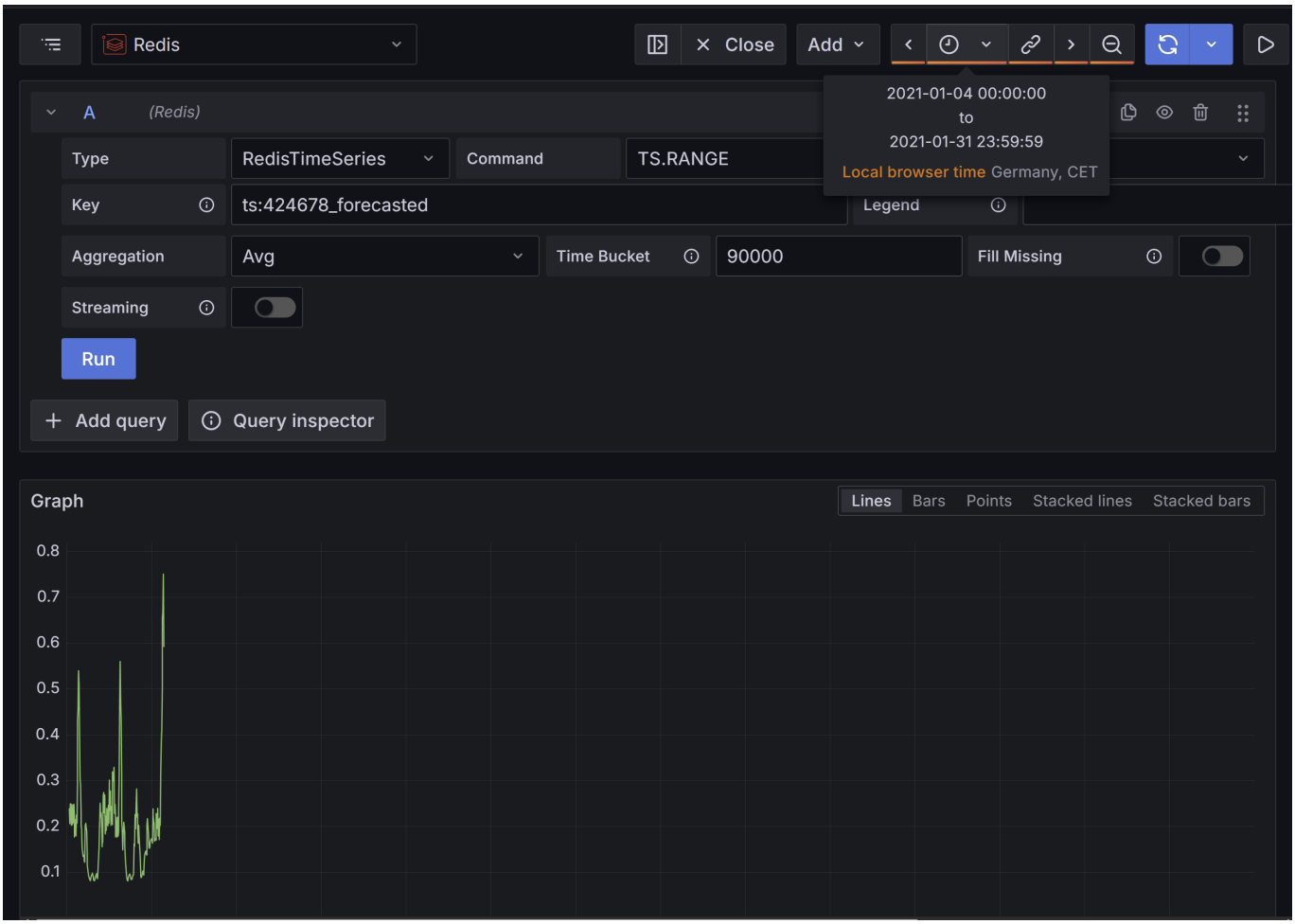
We have created a port forwarding to get access to grafana with our browser. The command we use was : `kubectl port-forward service/grafana-service 3000:3000`. As result we are getting the following

result when navigate to `localhost:3000`in our browser and login as admin :




## Task 6. Capture

From our grafana dashboard, we fill the for as represent on the two pictures and for each, we are getting following results:




Redis
Close
Add to dashboard
Type
RedisTimeSeries
Command
TS.RANGE
Key
ts:424678_real
Legend
Aggregation
Avg
Time Bucket
90000
Fill Missing
Streaming
Run
Add query
Query inspector
2021-01-04 00:00:00
to
2021-01-31 23:59:59
Local browser time Germany, CET
Graph
Lines
Bars
Points
Stacked lines
3
2.5
2
1.5
1
0.5
0
01/04
01/06
01/08
01/10
01/12
01/14
01/16
01/18
01/20
01/22
01/24
01/26
01/28
01/30