Kubernetes

1. Installation and Configuration of Kubernetes

- Install and configure a local or online Kubernetes cluster.
- Verify that the cluster is operational using the kubectl command.

```
PS C:\Users\salma> kubectl config current-context
docker-desktop
```

```
kubectl config view
```

```
└─$ kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://kubernetes.docker.internal:6443
  name: docker-desktop
contexts:
- context:
    cluster: docker-desktop
    user: docker-desktop
  name: docker-desktop
current-context: docker-desktop
kind: Config
preferences: {}
users:
- name: docker-desktop
  user:
    client-certificate-data: DATA+OMITTED
    client-key-data: DATA+OMITTED
```

2. Application Deployment

- Create a Kubernetes deployment for the web application using a YAML configuration file.

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: first-deployment

spec:

  replicas: 1

  selector:

    matchLabels:

      app: spring-app

  template:

    metadata:

      labels:

        app: spring-app
```

```
    spec:
      containers:
        - name: spring-app
          image: salmaghabri/spring-app
          ports:
            - containerPort: 8080
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

PS C:\Users\salma\OneDrive\Bureau\9raya\DevOupsie> kubectl apply -f first_deployment.yaml
deployment.apps/first-deployment created
PS C:\Users\salma\OneDrive\Bureau\9raya\DevOupsie>
```

- Verify that the deployment is done correctly and that the pods are running.

```
PS C:\Users\salma\OneDrive\Bureau\9raya\DevOupsie> kubectl get pods --show-labels
NAME                                READY   STATUS    RESTARTS   AGE     LABELS
first-deployment-c6f7889f9-x85rx    1/1     Running   0          6m11s   app=spring-app,pod-template-hash=c6f7889f9
```

```
PS C:\Users\salma\OneDrive\Bureau\9raya\DevOupsie> kubectl get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
```

3. Replica Management

- Scale the application by modifying the number of replicas of the deployment.

```
kubectl scale deployment first-deployment --replicas=3
```

```
first-deployment   1/1     1            1           18m
PS C:\Users\salma\OneDrive\Bureau\9raya\DevOupsie> kubectl scale deployment first-deployment --replicas=3
deployment.apps/first-deployment scaled
```

- Verify that the new replicas are created and deployed correctly.

```
PS C:\Users\salma\OneDrive\Bureau\9raya\DevOupsie> kubectl scale deployment first-deployment --replicas=3
deployment.apps/first-deployment scaled
PS C:\Users\salma\OneDrive\Bureau\9raya\DevOupsie> kubectl get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
first-deployment   3/3     3            3           21m
```

4. Service Definition

- Create a Kubernetes service to expose the application internally.

```
apiVersion: v1
kind: Service
metadata:
  name: spring-app-service
spec:
  selector:
    app: spring-app
  ports:
```

```
    - protocol: TCP
      port: 80
      targetPort: 8080
```

- `selector` specifies the pods targeted by this service. It matches the pods labeled with `app: spring-app`, which is the same label used in your deployment.
- `ports` specifies the ports exposed by the service. In this example, the service exposes port 80 on the cluster, which is forwarded to port 8080 of the pods.

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

```
PS C:\Users\salma\OneDrive\Bureau\9raya\DevOupsie> kubectl apply -f first_service.yaml
service/spring-app-service created
PS C:\Users\salma\OneDrive\Bureau\9raya\DevOupsie> kubectl get services
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
kubernetes          ClusterIP   10.96.0.1        <none>        443/TCP   5h17m
spring-app-service  ClusterIP   10.109.155.198   <none>        80/TCP    54s
```

- Verify that the service is accessible from the Kubernetes cluster.

```
PS C:\Users\salma\OneDrive\Bureau\9raya\DevOupsie> kubectl get services
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
kubernetes          ClusterIP   10.96.0.1        <none>        443/TCP   5h26m
spring-app-service  ClusterIP   10.109.155.198   <none>        80/TCP    10m
PS C:\Users\salma\OneDrive\Bureau\9raya\DevOupsie> kubectl get endpoints spring-app-service
NAME                ENDPOINTS                                         AGE
spring-app-service  10.1.0.11:8080,10.1.0.12:8080,10.1.0.13:8080   13m
```

the output shows that the `spring-app-service` service is currently routing traffic to three different pods (`10.1.0.11:8080`, `10.1.0.12:8080`, and `10.1.0.13:8080`), all of which are listening on port `8080`. This suggests that our Spring application is being served by multiple pods and is ready to receive traffic.

5. External Application Exposure:

- Define a LoadBalancer type service to expose the application externally.

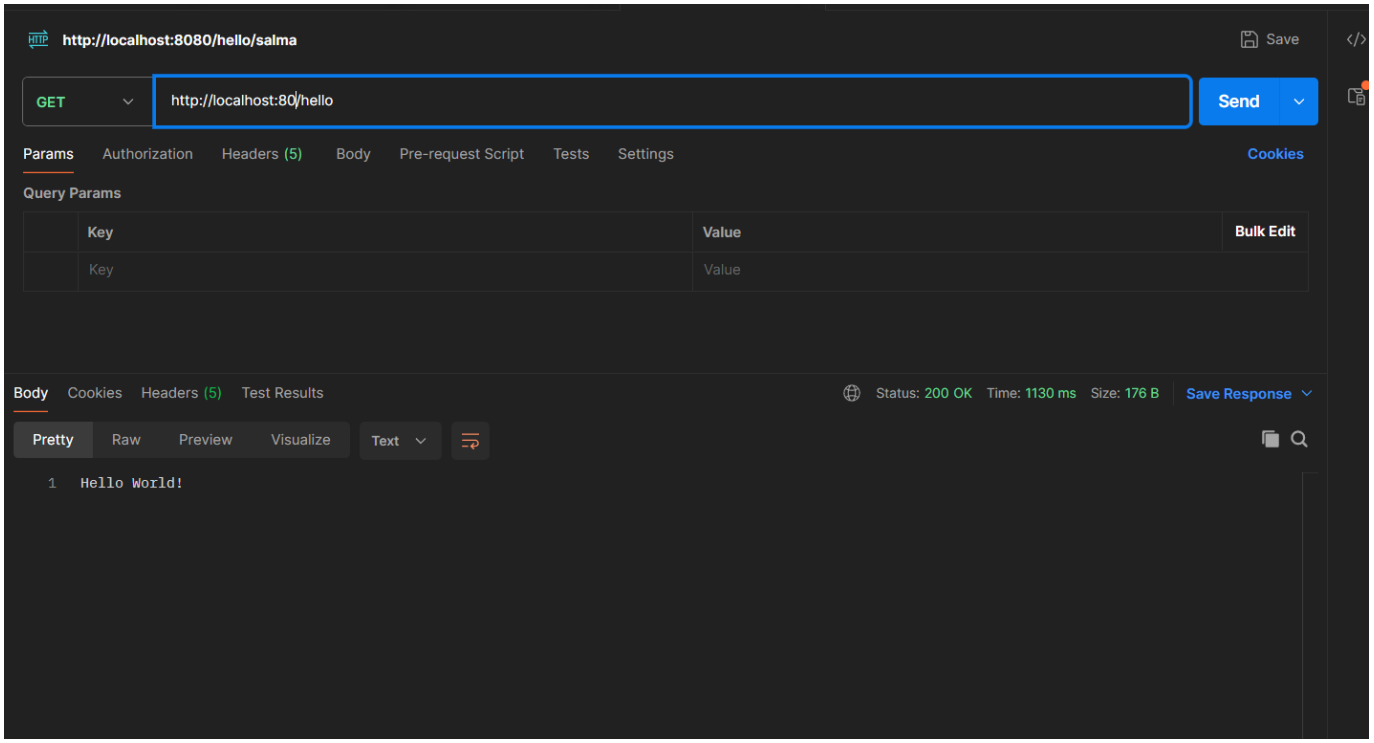
```
apiVersion: v1
kind: Service
metadata:
  name: spring-app-loadbalancer
spec:
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  selector:
    app: spring-app
```

- Verify that the application can be accessed from outside the cluster using the assigned IP address. Once the LoadBalancer type service is created, we can check its status and get the assigned IP address. Service information, including its type, exposed ports, and the IP address assigned by the LoadBalancer.

```
kubectl get svc spring-app-loadbalancer
```

```
PS C:\Users\salma\OneDrive\Bureau\9raya\DevOupsie> kubectl apply -f load_balancer_service.yaml
service/spring-app-loadbalancer created
PS C:\Users\salma\OneDrive\Bureau\9raya\DevOupsie> kubectl get svc spring-app-loadbalancer
NAME                      TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
spring-app-loadbalancer   LoadBalancer   10.111.25.46   localhost     80:32040/TCP   3m56s
```

On our machine we can access the app via `http://localhost:80` if we send a get request to `/hello`



6. Update Management

Update the container image used by the application.

We change the image to nginx for example:

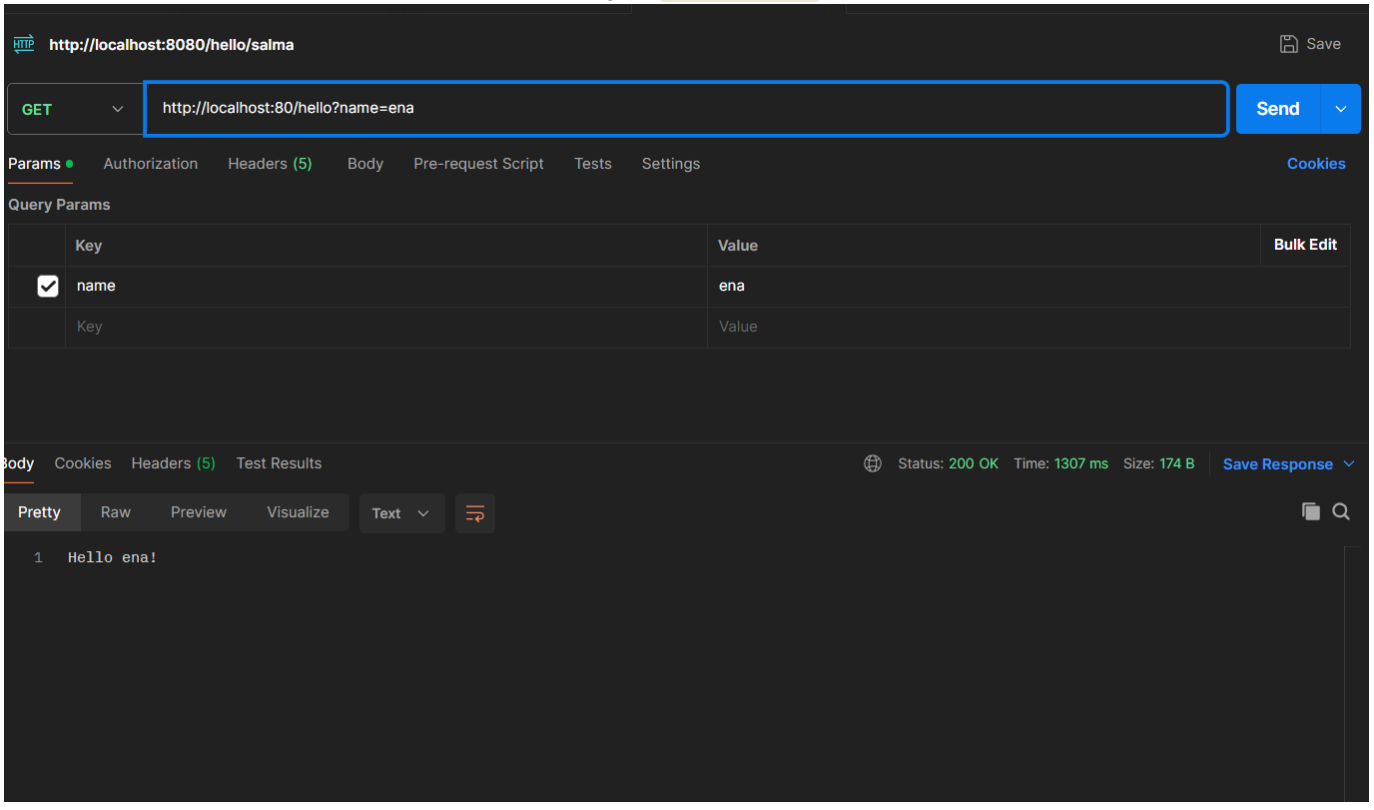
```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: first-deployment

spec:

  replicas: 1

  selector:

    matchLabels:

      app: spring-app
```

```
template:
  metadata:
    labels:
      app: spring-app
  spec:
    containers:
      - name: spring-app
        image: nginx // image changed
        ports:
          - containerPort: 80
```

Verify that the new version of the application is deployed correctly without service interruption.

```
PS C:\Users\salma\OneDrive\Bureau\9raya\DevOupsie> kubectl get pods
NAME                                READY   STATUS              RESTARTS   AGE
debug                               1/1     Running             0          90m
first-deployment-59ff986c5d-njtb5   0/1     ContainerCreating   0          28s
```

while the container is created, the old one is still listening on `localhost:80`



7. Monitoring and Logging: Prometheus

source

kube-state-metrics (KSM) is a simple service that listens to the Kubernetes API server and generates metrics about the state of the objects deployed in the cluster.
Kube State Metrics is commonly used in conjunction with Prometheus, a popular open-source monitoring and alerting system.

Install `kube-state-metrics`

```
git clone https://github.com/kubernetes/kube-state-metrics.git
```

```
kubectl apply -f kubernetes
```

```
┌──(salma㉿LAPTOP-C38TPMN9)-[/mnt/c/Users/salma/OneDrive/Bureau/9raya/DevOupsie/kube-state-metrics]
└─$ kubectl apply -f kubernetes
clusterrolebinding.rbac.authorization.k8s.io/kube-state-metrics created
clusterrole.rbac.authorization.k8s.io/kube-state-metrics created
deployment.apps/kube-state-metrics created
serviceaccount/kube-state-metrics created
service/kube-state-metrics created
```

Set Up `kube-state-metrics` to Expose Metrics for Your Kubernetes Cluster

We create the `kube-state-metrics` service

```
kind: Service
apiVersion: v1
metadata:
  namespace: kube-system
  name: kube-state-nodeport
spec:
  selector:
    k8s-app: kube-state-metrics
  ports:
  - protocol: TCP
    port: 8080
    nodePort: 30000
  type: NodePort
```

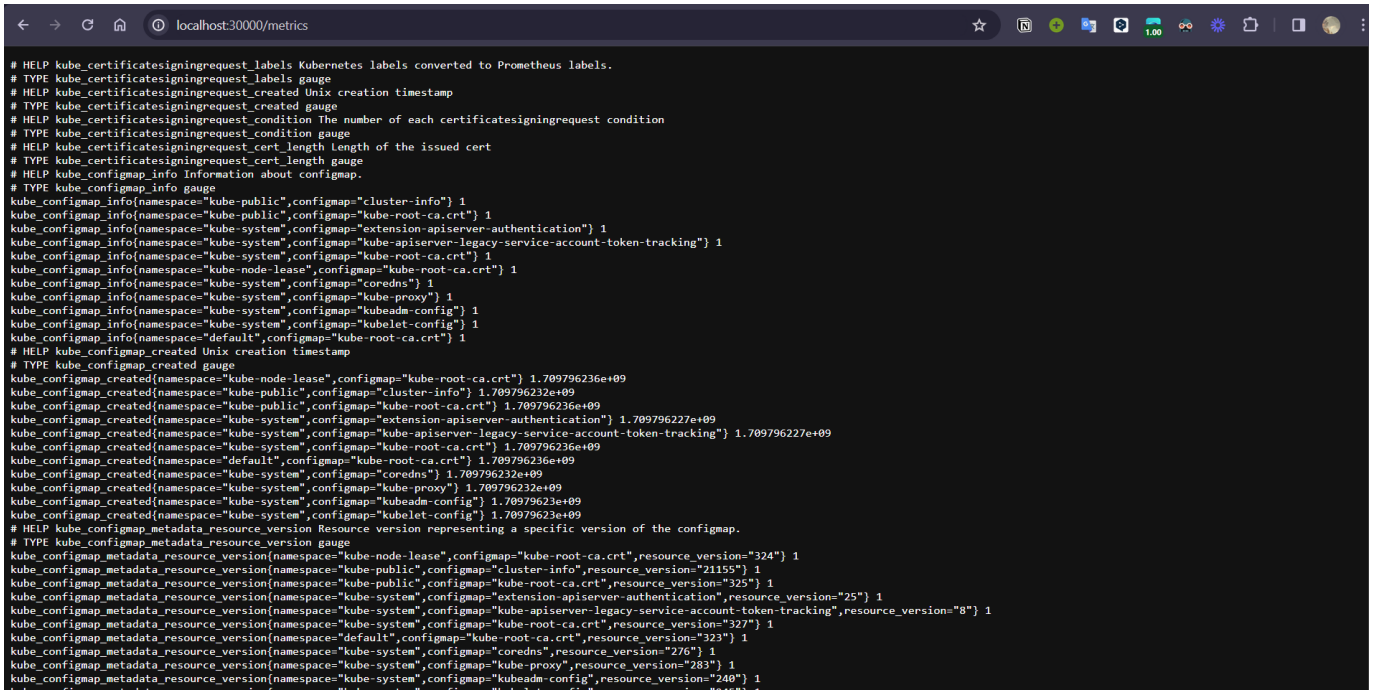
```
kubectl apply -f kube-state-metrics-nodeport-svc.yml
```

```
┌──(salma㉿LAPTOP-C38TPMN9)-[/mnt/c/Users/salma/OneDrive/Bureau/9raya/DevOupsie]
└─$ kubectl apply -f kube-state-metrics-nodeport-svc.yml
service/kube-state-nodeport created
```

If we open the browser and go to `localhost:30000` we can see the data of our cluster

Kube Metrics

- metrics
- healthz



Configure Prometheus to Scrape Metrics from `kube-state-metrics`

- create a promotheus.yaml

```
# my global config

global:

  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.

  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.

  # scrape_timeout is set to the global default (10s).



# Alertmanager configuration
```

```
alerting:
  alertmanagers:
    - static_configs:
        - targets:
          # - alertmanager:9093


# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"
# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: "prometheus"
    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.
    static_configs:
      - targets: ["localhost:9090"]
    - job_name: kubernetes
    static_configs:
      - targets: ["localhost:30000"]
```

Example of a query

```
kube_pod_status_ready{namespace="default",condition="true"}
```