

Fatih Yigit

DevOps Engineer

Istanbul, Turkey
+90 536 979 8563
yigitf15@itu.edu.tr

Study Steps

Main:

- Environment decision and designing the architecture
- Creating the Kubernetes cluster
- Dockerizing the Java application
- Creating Kubernetes manifests and deploying the app

Extra:

- Monitoring
- Testing the HPA and Troubleshooting

Environment Decision and Designing the Architecture

To begin with the environment, I have a desktop PC which has 16GB ram and a processor with 12CPUs. So, the local environment is suitable for running virtual machines instead of cloud environments. There are several options for a local kubernetes cluster, like: minikube, vagrant, virtualbox (which also used by vagrant) and kind. I already had Docker Desktop installed on my PC, so I decided to study with Kind instead of other tools.

Creating the Kubernetes Cluster

I worked with the exe file(kind.exe) to keep the dependencies minimum for the github repo I created. After a couple trials created a basic yaml configuration file and created the cluster with these configurations with two nodes and a master node.

```
PS C:\dreamgames3> .\kind.exe create cluster --config kind-config.yaml
Creating cluster "kind" ...
  • Ensuring node image (kindest/node:v1.27.1) █ ...
  ✓ Ensuring node image (kindest/node:v1.27.1) █
  • Preparing nodes 🟡 🟡 🟡 ...
  ✓ Preparing nodes 🟡 🟡 🟡
  • Writing configuration 📄 ...
  ✓ Writing configuration 📄
  • Starting control-plane 🚦 ...
  ✓ Starting control-plane 🚦
  • Installing CNI 🚦 ...
  ✓ Installing CNI 🚦
  • Installing StorageClass 📄 ...
  ✓ Installing StorageClass 📄
  • Joining worker nodes 🟢 ...
  ✓ Joining worker nodes 🟢
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind
```

```

! kind-config.yaml
1  kind: Cluster
2  apiVersion: kind.x-k8s.io/v1alpha4
3  nodes:
4  - role: control-plane
5    kubeadmConfigPatches:
6    - |
7      kind: InitConfiguration
8      nodeRegistration:
9      kubeletExtraArgs:
10     node-labels: "ingress-ready=true"
11    extraPortMappings:
12    - containerPort: 80
13      hostPort: 80
14      protocol: TCP
15  - role: worker
16  - role: worker

```

Now, the cluster needs an ingress for exposing applications to the outside of the cluster and load-balancing. I used a simple predesigned configuration for installing Ingress Nginx services.

```

PS C:\dreamgames3> kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/kind/deploy.yaml
namespace/ingress-nginx created
serviceaccount/ingress-nginx created
serviceaccount/ingress-nginx-admission created
role.rbac.authorization.k8s.io/ingress-nginx created
role.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrole.rbac.authorization.k8s.io/ingress-nginx created
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created
rolebinding.rbac.authorization.k8s.io/ingress-nginx created
rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
configmap/ingress-nginx-controller created
service/ingress-nginx-controller created
service/ingress-nginx-controller-admission created
deployment.apps/ingress-nginx-controller created
job.batch/ingress-nginx-admission-create created
job.batch/ingress-nginx-admission-patch created
ingressclass.networking.k8s.io/nginx created
validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-nginx-admission created

```

We do not need a request validation for local environment. So we can delete the relevant configuration with the command below. I tried to run the cluster without this exception and it failed to expose web ports, so it's a part of the project now.

```

kubectl delete -A ValidatingWebhookConfiguration ingress-nginx-admission

```

Dockerizing the Java Application

I used the sample java application which is provided in the task study. At the first downloaded Eclipse IDE and tried to run this application with IDE. Running the following commands provided the jar file.

Downloading the dependencies:

```
mvn dependency:go-offline
```

Packaging the application to a .jar file with skipping tests:

```
mvn package -DskipTests
```

Then running the jar file on terminal

```
Java -jar app-0.0.1-SNAPSHOT.jar
```

```

/\ / ---' - - - - ( ) - - - - \\\ \ \ \
( ( ) \ --- | ' - | ' - | | ' - \ \ \ \ \
\ \ / --- ) | | ) | | | | | | | ( | ) ) ) )
' | --- | - - | | | | | | | \ \ \ / / / /
===== | | ===== | --- / - / - / - /
:: Spring Boot ::                (v2.6.1)

2023-06-07 12:58:50.529 INFO 25596 --- [           main] com.samplejavaapp.app.DemoApplication : Startin
g DemoApplication v0.0.1-SNAPSHOT using Java 17.0.2 on DESKTOP-TMA3ESV with PID 25596 (C:\javaapp\docker\app
.jar started by yati in C:\javaapp\docker)
2023-06-07 12:58:50.534 INFO 25596 --- [           main] com.samplejavaapp.app.DemoApplication : No acti
ve profile set, falling back to default profiles: default
PS C:\javaapp\docker> 2023-06-07 12:58:51.382 INFO 25596 --- [           main] o.s.b.w.embedded.tomcat.Tomc
atWebServer : Tomcat initialized with port(s): 9002 (http)
2023-06-07 12:58:51.448 INFO 25596 --- [           main] w.s.c.ServletWebServerApplicationContext : Root We
bApplicationContext: initialization completed in 856 ms
2023-06-07 12:58:51.763 INFO 25596 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat
started on port(s): 9002 (http) with context path ''
2023-06-07 12:58:51.771 INFO 25596 --- [           main] com.samplejavaapp.app.DemoApplication : Started DemoApplication in 1.668 seconds (JVM running for 2.001
)

```

The project is up and running. Now it's time to create a docker image with Dockerfile with these steps. I used two stages to build and run the app.

To build the java maven project it will be better to use an image which has both java and maven installed in it. Otherwise, we will have to install these dependencies manually. So, I preferred to use:

```
FROM maven:3.8.3-openjdk-11-slim AS build
```

Changing the working directory for app to /app and copy the pom.xml to get dependencies.

```
WORKDIR /app
```

```
COPY app/pom.xml .
```

The pom.xml is in the app folder because the dockerfile will be outside of the repo to ease the of my github repo.

Then get the dependencies and copy the source java files.

```
RUN mvn dependency:go-offline
```

```
COPY app/src ./src
```

Now we have everything to package our application.

```
RUN mvn package -DskipTests
```

Packaging was the last step of Build stage. Now, we have to run this package but there are so many unnecessary files and dependencies in this image. So, I will create another low-sized image with only java installed in it and run the .jar package I just created.

```
FROM openjdk:11-jre-slim
```

```
WORKDIR /app
```

While copying the jar file from build image, I wanted to change its name to app.jar for simplicity.

```
COPY --from=build /app/target/app-0.0.1-SNAPSHOT.jar ./app.jar
```

Exposing the default port for the application and running the application.

```
EXPOSE 9001
```

```
CMD ["java", "-jar", "app.jar"]
```

To create the image you have to clone the repo and run the dockerfile:

```
PS C:\dreamgames3> git clone https://github.com/SampleJavaApp/app.git
Cloning into 'app'...
remote: Enumerating objects: 32, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 32 (delta 0), reused 2 (delta 0), pack-reused 0Receiving objects: 34%
(11/32)
Receiving objects: 100% (32/32), 53.88 KiB | 1.04 MiB/s, done.
```

```
PS C:\dreamgames3> docker build -t fatih-app:0.0.1 .
[+] Building 0.9s (15/15) FINISHED
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 32B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/openjdk:11-jre-slim 0.7s
=> [internal] load metadata for docker.io/library/maven:3.8.3-openjdk-11-slim 0.7s
=> [build 1/6] FROM docker.io/library/maven:3.8.3-openjdk-11-slim@sha256:a4c3 0.0s
=> [internal] load build context 0.0s
=> => transferring context: 963B 0.0s
=> [stage-1 1/3] FROM docker.io/library/openjdk:11-jre-slim@sha256:93af7df230 0.0s
=> CACHED [stage-1 2/3] WORKDIR /app 0.0s
=> CACHED [build 2/6] WORKDIR /app 0.0s
=> CACHED [build 3/6] COPY app/pom.xml . 0.0s
=> CACHED [build 4/6] RUN mvn dependency:go-offline 0.0s
=> CACHED [build 5/6] COPY app/src ./src 0.0s
=> CACHED [build 6/6] RUN mvn package -DskipTests 0.0s
=> CACHED [stage-1 3/3] COPY --from=build /app/target/app-0.0.1-SNAPSHOT.jar 0.0s
=> exporting to image 0.0s
=> => exporting layers 0.0s
=> => writing image sha256:ab27776db5a54e2fe6dce167bfd0065b51b1c68cc1434f5a9f 0.0s
=> => naming to docker.io/library/fatih-app:0.0.1 0.0s
```

Now checking the docker image:

```
PS C:\dreamgames3> docker images
REPOSITORY          TAG          IMAGE ID
fatih-app            0.0.1       ab27776db5a5
```

Testing the application with running on docker.

```
PS C:\dreamgames3\K8S-with-Kind> docker run -p 9001:9001 fatih-app:0.0.1

  /\  /---' _ _ _ _ _ ( _ _ _ _ _  \ \ \ \
( ( ) \---| ' _ | ' _ | ' _ \ _ _ | \ \ \ \
\ \ _ _ _| |_) | | | | | | | | | | | | | | | |
' | _ _ _| . _ | | | | | | | | | | | | | | | |
=====|_|=====|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|
:: Spring Boot ::                (v2.6.1)

2023-06-07 11:23:03.523 INFO 1 --- [           main] com.samplejavaapp.app.De
cation : Starting DemoApplication v0.0.1-SNAPSHOT using Java 11.0.16 on 611
```

← → ↺ ⓘ localhost:9001/api/foos?val=TEST

QueryStringValue: TEST

Tests are successful. So, the image is working. Now it's time to push this image into the cluster.

```
PS C:\dreamgames3\K8S-with-Kind> .\kind.exe load docker-image fatih-app:0.0.1
Image: "fatih-app:0.0.1" with ID "sha256:ab27776db5a54e2fe6dce167bfd0065b51b1c68cc1434f5a9fb31d92d255fe8a" not yet present on node "kind-control-plane", loading...
Image: "fatih-app:0.0.1" with ID "sha256:ab27776db5a54e2fe6dce167bfd0065b51b1c68cc1434f5a9fb31d92d255fe8a" not yet present on node "kind-worker", loading...
Image: "fatih-app:0.0.1" with ID "sha256:ab27776db5a54e2fe6dce167bfd0065b51b1c68cc1434f5a9fb31d92d255fe8a" not yet present on node "kind-worker2", loading...
PS C:\dreamgames3\K8S-with-Kind> |
```

The image is imported to all three nodes, now I can deploy my application in the cluster.

Creating K8S Manifests & Deploying the Application

Namespace

Before creating the manifests, I create a namespace for the java application. All the other manifests about the application will be in the same namespace.

```
apiVersion: v1
kind: Namespace
metadata:
  name: uygulama
```

Deployment

After creating the namespace, I created a configuration file for deployment manifest. I set the initial replica count as 3, cpu and memory values are enough to run the applications and not so high to challenge my PC.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: fatih-app
  namespace: uygulama
spec:
  replicas: 3
  selector:
    matchLabels:
      app: fatih-app
  template:
    metadata:
      labels:
        app: fatih-app
    spec:
      containers:
        - name: fatih-app
          image: fatih-app:0.0.1
          ports:
            - containerPort: 9001
```

```

env:
  - name: SPRING_PROFILES_ACTIVE
    value: production
resources:
  limits:
    cpu: "1"
    memory: "1Gi"
  requests:
    cpu: "500m"
    memory: "512Mi"

```

```

PS C:\dreamgames3\K8S-with-Kind> kubectl apply -f application/namespace.yaml
namespace/uygulama created
PS C:\dreamgames3\K8S-with-Kind> kubectl apply -f application/deployment.yaml
deployment.apps/fatih-app created

```

Now I runned the deployment, let's check the deployment and pods.

```

PS C:\dreamgames3\K8S-with-Kind> kubectl get deployment -n uygulama
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
fatih-app     3/3     3             3           9m44s

```

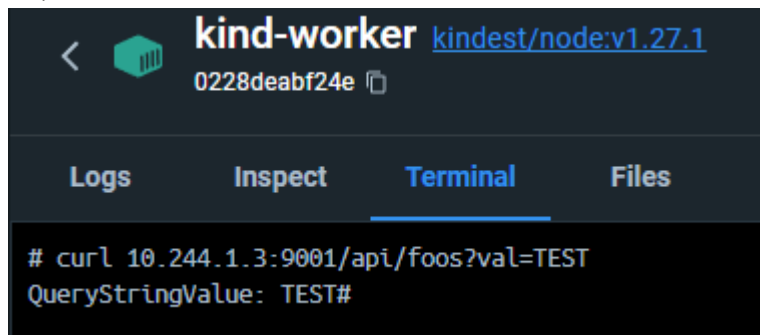
```

PS C:\dreamgames3\K8S-with-Kind> kubectl get pods -n uygulama -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP            NODE
GATES
fatih-app-676d7c66c5-4frqt  1/1     Running   0           6m18s  10.244.2.4    kind-worker
fatih-app-676d7c66c5-6hccw  1/1     Running   0           6m18s  10.244.2.3    kind-worker
fatih-app-676d7c66c5-8hzxd  1/1     Running   0           6m18s  10.244.1.3    kind-worker2

```

Three pods, as configured. I want to check if the application is running without a problem. I do not have a service and ingress installed; so I can check one of the pods on its own machine.

I opened the terminal of "kind-worker" and tried the curl.



The screenshot shows a terminal window titled "kind-worker" with the URL "kindest/node:v1.27.1" and ID "0228deabf24e". Below the title bar are tabs for "Logs", "Inspect", "Terminal", and "Files". The "Terminal" tab is active, displaying the command "# curl 10.244.1.3:9001/api/foos?val=TEST" and its output "QueryStringValue: TEST#".

Service

The test is successful. But we need to reach the application outside of the node and balance the pods. So, first creating a service configuration.

```

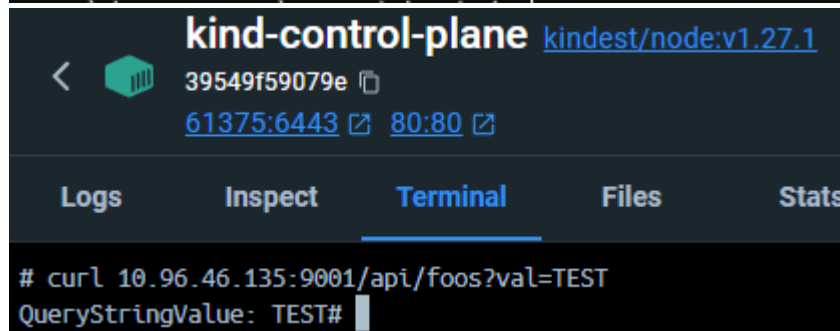
apiVersion: v1
kind: Service
metadata:
  name: fatih-app-service
  namespace: uygulama
spec:
  selector:
    app: fatih-app
  ports:
    - port: 9001

```

```
targetPort: 9001
type: ClusterIP
```

Running the service and checking if it's created. Then I will try to reach this service from the service IP and port from the master node.

```
PS C:\dreamgames3\K8S-with-Kind> kubectl apply -f application/service.yaml
service/fatih-app-service created
PS C:\dreamgames3\K8S-with-Kind> kubectl get service -n uygulama
NAME                TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
fatih-app-service   ClusterIP     10.96.46.135  <none>         9001/TCP   7s
```



The service manifest test is also seems fine, and we can say that the service is running.

Ingress

Now we can reach the application with directly to the pod from its own node, and the service from master node. But still we can not reach this application from the outside of the cluster.

We need to expose the service with and Ingress, so we can reach the application from outside. I created a configuration file for Ingress below.

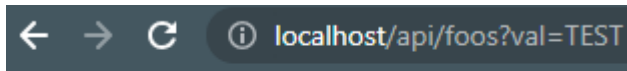
```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: fatih-app-ingress
  namespace: uygulama
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
    - http:
        paths:
          - path: "/"
            pathType: Prefix
            backend:
              service:
                name: fatih-app-service
                port:
                  number: 9001
```

In this configuration it seems that the Ingress forwarding the "fatih-app-service" service from the "port:9001" on its own root path. Let's apply this yaml file.


```
PS C:\dreamgames3\K8S-with-Kind> kubectl apply -f application/ingress.yaml
ingress.networking.k8s.io/fatih-app-ingress created
```

```
PS C:\dreamgames3\K8S-with-Kind> kubectl get ingress -n uygulama
NAME          CLASS  HOSTS    ADDRESS    PORTS    AGE
fatih-app-ingress  nginx  *        localhost  80       43s
```

Finally, we can reach the application from the browser which is actually outside of the cluster.



QueryStringValue: TEST

Horizontal Pod Autoscaler

We can also add the Horizontal Pod Autoscaler (HPA) manifest to our cluster to auto scale our replicas. I want to set minimum replicas to "4" which is higher than our initial replica count "3". So, I can see if this manifest works.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: fatih-app-hpa
  namespace: uygulama
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: fatih-app
  minReplicas: 4
  maxReplicas: 5
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
```

```
PS C:\dreamgames3\K8S-with-Kind> kubectl apply -f application/hpa.yaml
horizontalpodautoscaler.autoscaling/fatih-app-hpa created
PS C:\dreamgames3\K8S-with-Kind> kubectl get pods -n uygulama
NAME          READY   STATUS    RESTARTS   AGE
fatih-app-676d7c66c5-4frqt  1/1     Running   0           47m
fatih-app-676d7c66c5-6hccw  1/1     Running   0           47m
fatih-app-676d7c66c5-8hzxd  1/1     Running   0           47m
fatih-app-676d7c66c5-lcxr5  0/1     ContainerCreating  0           0s
PS C:\dreamgames3\K8S-with-Kind> kubectl get pods -n uygulama
NAME          READY   STATUS    RESTARTS   AGE
fatih-app-676d7c66c5-4frqt  1/1     Running   0           47m
fatih-app-676d7c66c5-6hccw  1/1     Running   0           47m
fatih-app-676d7c66c5-8hzxd  1/1     Running   0           47m
fatih-app-676d7c66c5-lcxr5  1/1     Running   0           7s
PS C:\dreamgames3\K8S-with-Kind> |
```

We can see that the new pod is created after applying HPA. Because the minreplicas configuration for HPA is higher than our initial replica count.

Monitoring (Extra)

Kubernetes Dashboard

The Kubernetes Cluster for a sample java application is actually completed. But, things did not go that straight forward like I documented above. There were challenging steps that I tried to solve the problem for hours. For example; importing the image into cluster and installation ingress or configuring it. I had to troubleshoot my cluster somehow. I used Kubernetes Dashboard for troubleshooting my errors.

I installed Kubernetes Dashboard with the preconfigured yaml file below.

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml
```

I created a service account and authorized it to be the dashboard administrator.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
```

```
kubectl apply -f monitoring/admin-user.yaml
kubectl proxy
```

Running the commands above exposed the dashboard by the url below.

```
http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/
```

Creating the admin token with the user I just created.

```
kubectl -n kubernetes-dashboard create token admin-user
```

Loki & Prometheus and Grafana

To monitor the logs that our sample java application created I wanted to implement Loki to our Kubernetes environment. I am also interested in the server metrics, so I will also implement a sample Prometheus stack for Kubernetes Cluster.

I found some preconfigured straight forward installations on the internet. Helm is required to install with these instructions below.

```
# helm repo add grafana https://grafana.github.io/helm-charts
# helm repo add Prometheus-community https://prometheus-community.github.io/helm-charts
# helm repo update

# helm install loki grafana/loki-stack --namespace log-monitoring --create-namespace --
set grafana.enabled=true

# helm install prometheus Prometheus-community/kube-prometheus-stack --namespace
metrics-monitoring --create-namespace

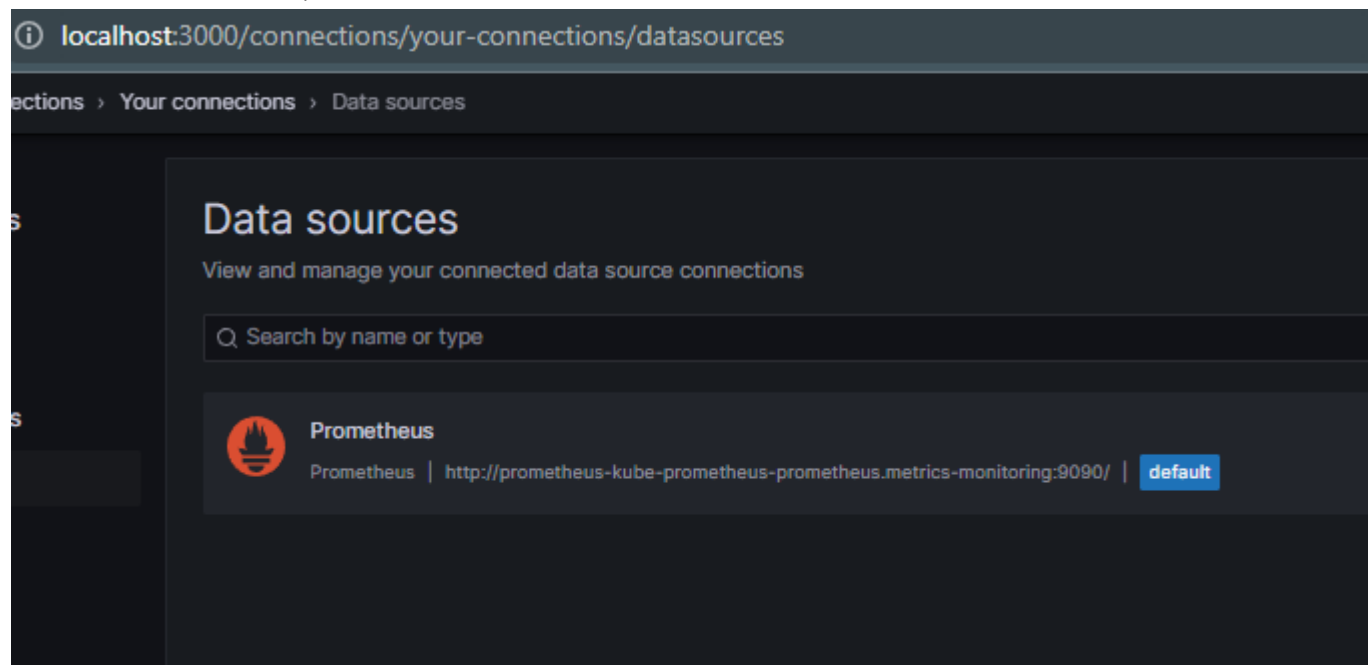
# kubectl port-forward --namespace metrics-monitoring service/prometheus-grafana 3000:80
```

Now, the grafana will be active on <http://localhost:3000>

But still we do not have the credentials. We can get the secret manifest as yaml configuration of the grafana with the command below and decode it with base64 decoding.

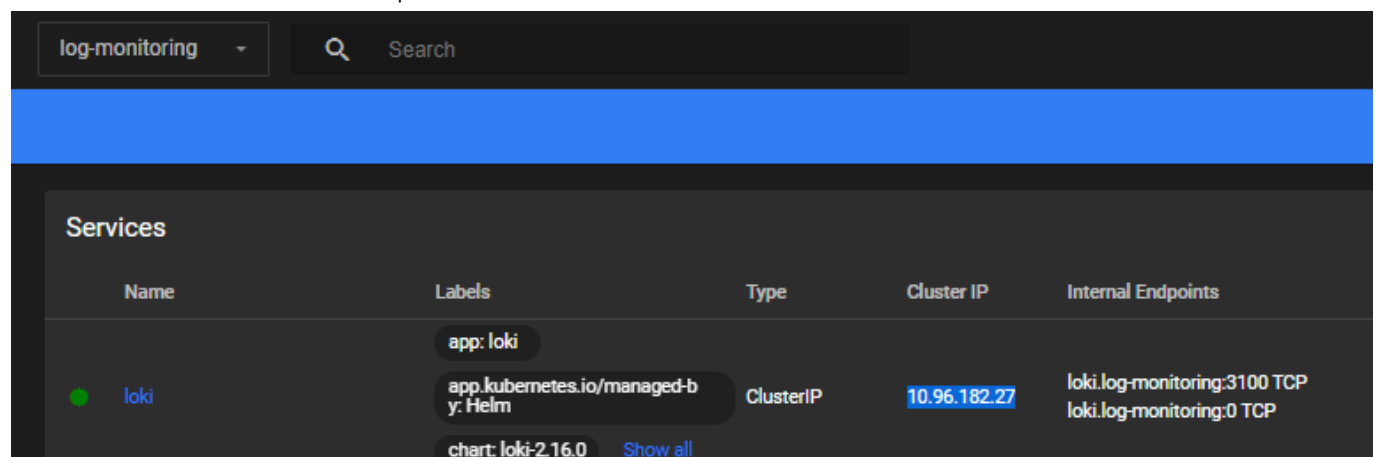
```
kubectl get secret -n metrics-monitoring prometheus-grafana -o yaml
```

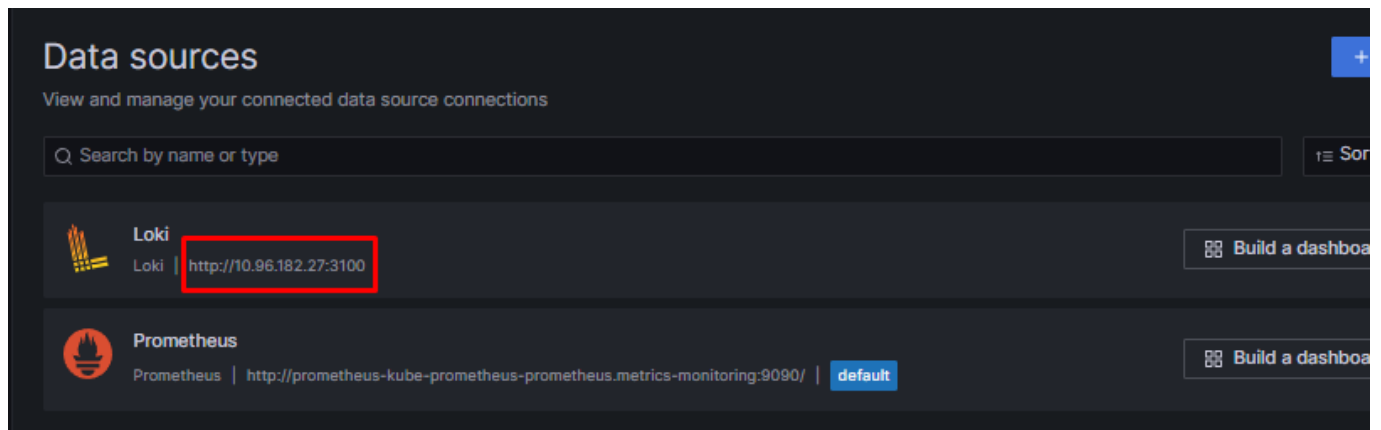
We just exposed the Grafana which is actually in the Prometheus-stack. So, Prometheus database is saved in the Grafana by default.



We have another Grafana with the loki-stack but I prefer to use two datasources in one Grafana server; this lets us to create dashboards with both Loki and Prometheus sources.

I have to add Loki as a data source here, first I need the loki endpoint. We can reach the data source via loki service IP and the default port of the Loki database "3100". I will use the Kubernetes Dashboard.





Now we have two datasources in one Grafana client. We can create Dashboards with both logs and metrics now.

Testing the HPA and Troubleshooting (Extra)

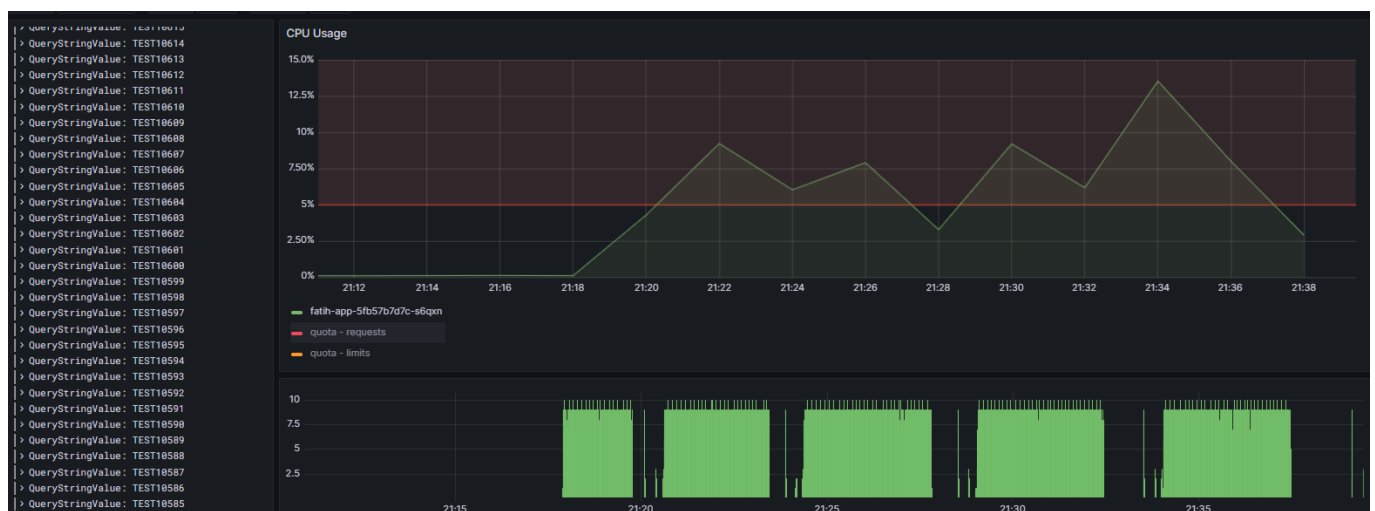
Now, I want to test the Horizontal Pod Autoscaler.

First I want to rearrange my deployment and HPA configuration to observe the scaling easily. Decreased the initial and minimum replica count to "1" increased the maxReplicas to "7". Decreased CPU and Memory limits to ¼ of the previous configuration. And decreased the CPU utilization threshold to 5%. I created a python script for load testing on my Kubernetes cluster.

```
import requests
import time
i=1
interval= 1/int(input("requests per second: "))
while True:
    resp= requests.get(url=f"http://localhost/api/foos?val=TEST{i}")
    i=i+1
    print(resp.text)
    time.sleep(interval)
```

To start the load test, enter the following commands. (Python must be installed)

```
pip install requests
python bot/test.py
```



```
PS C:\Users\yati> kubectl get pods -n uygulama
NAME                                READY   STATUS    RESTARTS   AGE
fatih-app-5fb57b7d7c-s6qxn         1/1     Running   6 (93s ago)  58m
```

It doesn't seem to be working properly. But it passed our first test when the minimum replica count was higher than initial replica count. It autoscaled to its minimum replica count, but now it does not scale the pods according to utilization.

So I checked the Kubernetes Dashboard for troubleshooting and saw the error below.

●	fatih-app-hpa.17667560f6b9f539	FailedGetResourceMetric	failed to get cpu utilization: unable to get metrics for resource cpu: unable to fetch metrics from resource metrics API: the server is currently unable to handle the request (get pods.metrics.k8s.io)	horizontal-pod-autoscaler
●	fatih-app-hpa.176662dc0395956f	FailedGetResourceMetric	failed to get cpu utilization: unable to get metrics for resource cpu: unable to fetch metrics from resource metrics API: the server could not find the requested resource (get pods.metrics.k8s.io)	horizontal-pod-autoscaler

I searched for this error on the internet and I figured that the metrics-server which actually should scrape the metrics from the pods could be absent; and checked.

```
PS C:\Users\yati> kubectl get pods -n kube-system
NAME                                READY   STATUS    RESTARTS   AGE
coredns-5d78c9869d-5bksg           1/1     Running   0           7h17m
coredns-5d78c9869d-zfc8k           1/1     Running   0           7h17m
etcd-kind-control-plane             1/1     Running   0           7h17m
kindnet-8mr7x                       1/1     Running   0           7h17m
kindnet-vr2zp                       1/1     Running   0           7h17m
kindnet-xwqrt                       1/1     Running   0           7h17m
kube-apiserver-kind-control-plane    1/1     Running   0           7h17m
kube-controller-manager-kind-control-plane 1/1     Running   0           7h17m
kube-proxy-57brz                    1/1     Running   0           7h17m
kube-proxy-thl4w                     1/1     Running   0           7h17m
kube-proxy-zbh8z                     1/1     Running   0           7h17m
kube-scheduler-kind-control-plane     1/1     Running   0           7h17m
```

Yes, I think that is the problem and the following command will install the metrics-server with a preconfigured yaml file.

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

```
PS C:\dreamgames3\K8S-with-Kind> kubectl get pods -n kube-system
NAME                                READY   STATUS    RESTARTS   AGE
coredns-5d78c9869d-5bksg           1/1     Running   0           7h43m
coredns-5d78c9869d-zfc8k           1/1     Running   0           7h43m
etcd-kind-control-plane             1/1     Running   0           7h43m
kindnet-8mr7x                       1/1     Running   0           7h43m
kindnet-vr2zp                       1/1     Running   0           7h43m
kindnet-xwqrt                       1/1     Running   0           7h43m
kube-apiserver-kind-control-plane    1/1     Running   0           7h43m
kube-controller-manager-kind-control-plane 1/1     Running   0           7h43m
kube-proxy-57brz                    1/1     Running   0           7h43m
kube-proxy-thl4w                     1/1     Running   0           7h43m
kube-proxy-zbh8z                     1/1     Running   0           7h43m
kube-scheduler-kind-control-plane     1/1     Running   0           7h43m
metrics-server-7b4c4d4bfd-6v749    0/1     Running   0           17m
```

It seems like I installed the metrics-server, but I actually waited a little too much the server to be ready. There is a problem, and I wanted to check it from the logs of the pod with the following command.

```
kubectl logs -n kube-system metrics-server-7b4c4d4bfd-6v749
```

Then I found this Failure in the container logs:

```
cannot validate certificate for 172.18.0.4 because it doesn't contain any IP SANs"
node="kind-control-plane"
```

I searched the problem on the internet. And solved with the following approach:

```
kubectl get deployment metrics-server -n kube-system -o yaml > monitoring/metrics-server.yaml
```

added those two lines in "args" part of the configuration:

```
args:
  - --kubelet-insecure-tls
  - --kubelet-preferred-address-types=InternalIP,Hostname,InternalDNS,ExternalDNS,ExternalIP
```

And applied with this configuration:

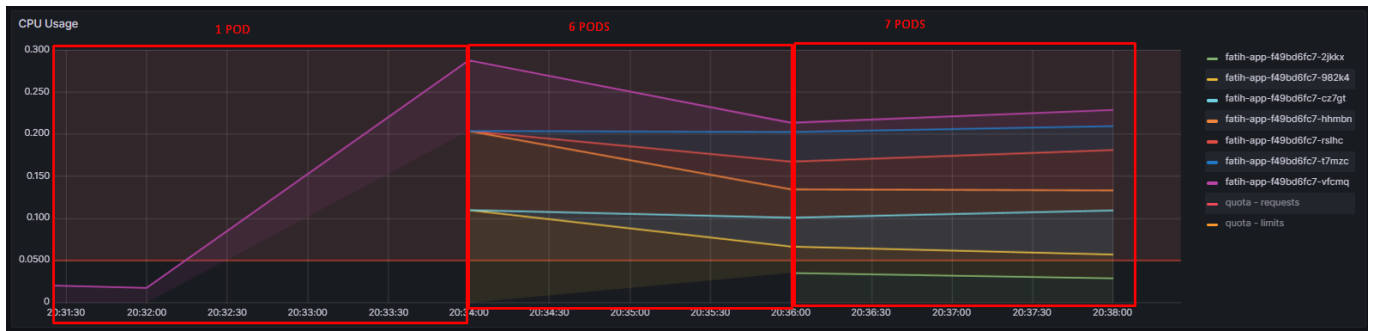
```
kubectl apply -f monitoring/metrics-server.yaml -n kube-system
```

kube-proxy-thl4w	1/1	Running	0	7h54m
kube-proxy-zbh8z	1/1	Running	0	7h54m
kube-scheduler-kind-control-plane	1/1	Running	0	7h55m
metrics-server-8655c9684c-pt7vn	1/1	Running	0	57s

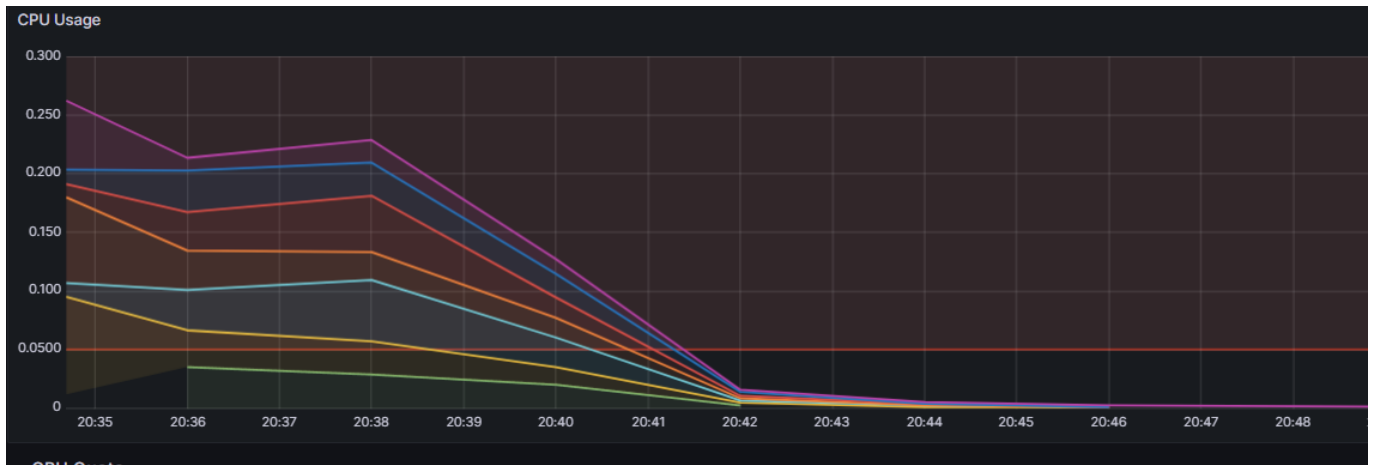
It worked, then I started the load test again:

```
python bot/test.py
```

```
PS C:\dreamgames3\K8S-with-Kind> kubectl get pods -n uygulama
NAME                                READY    STATUS    RESTARTS   AGE
fatih-app-f49bd6fc7-vfcmq          1/1     Running   0           45m
PS C:\dreamgames3\K8S-with-Kind> kubectl get pods -n uygulama
NAME                                READY    STATUS    RESTARTS   AGE
fatih-app-f49bd6fc7-2jkkx          1/1     Running   0           11s
fatih-app-f49bd6fc7-982k4          1/1     Running   0           41s
fatih-app-f49bd6fc7-cz7gt          1/1     Running   0           11s
fatih-app-f49bd6fc7-hhmbn          1/1     Running   0           41s
fatih-app-f49bd6fc7-rslhc          1/1     Running   0           11s
fatih-app-f49bd6fc7-vfcmq          1/1     Running   0           46m
PS C:\dreamgames3\K8S-with-Kind> kubectl get pods -n uygulama
NAME                                READY    STATUS    RESTARTS   AGE
fatih-app-f49bd6fc7-2jkkx          1/1     Running   0           32s
fatih-app-f49bd6fc7-982k4          1/1     Running   0           62s
fatih-app-f49bd6fc7-cz7gt          1/1     Running   0           32s
fatih-app-f49bd6fc7-hhmbn          1/1     Running   0           62s
fatih-app-f49bd6fc7-rslhc          1/1     Running   0           32s
fatih-app-f49bd6fc7-t7mzc          1/1     Running   0           17s
fatih-app-f49bd6fc7-vfcmq          1/1     Running   0           46m
PS C:\dreamgames3\K8S-with-Kind>
```



The HPA is automatically upscaling the replicas when cpu utilization gets higher than the threshold. Now I stopped my request bot:



It downscaled the replicas to the minimum value "1".

```
PS C:\dreamgames3\K8S-with-Kind> kubectl get pods -n uygulama
NAME                                READY   STATUS    RESTARTS   AGE
fatih-app-f49bd6fc7-vfcmq          1/1     Running   0           97m
PS C:\dreamgames3\K8S-with-Kind>
```

Github Link:

<https://github.com/yigitf/K8S-with-Kind>