Final Project

Group Name: Desperate
Group members: 210103464 Kangilov Sulaimon
210103452 Mukhammat Andybayev

YouTube presentation link

Description:

This particular document describes detailed information about the architecture of Microservices, relations and connections by the service of every Pod and containerization via Docker Desktop.

This project is a mesh to a customer service web application, which is based on Docker(containers) and managed by Kubernetes(minicube, kubectl) with usage of external sources of databases like GCS and also local storage.

This project can be used as a base, to apply on further modifications and learning approaches for new developers. Therefore the project has no certain values and is not attached to particular problem solving mechanisms. It is a project created and designed to be easily modified and applied in any project for a customer service systems and WEB applications.

Introduction:

There will be provided code and description of every step, attached screenshots and sitings to information sources. Every step refers to a certain part of the Grading Policy given by the Final project Description.

- Container Orchestration with Kubernetes (30 points):

First of all we installed minicube and kubectl on our running system using code below:

Updating system packages and installing Minikube dependencies

```
$ sudo apt update & sudo apt upgrade
$ sudo apt install -y curl wget apt-transport-https
```

Installing Minikube

$ curl -LO
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

```
$ sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Installing kubectl utility

$ curl -LOhttps://storage.googleapis.com/kubernetes-release/release/"curl -s
https://storage.googleapis.com/kubernetes-release/release/stable.txt"/bin/linux/amd64/kubectl

$ chmod +x kubectl

$ sudo mv kubectl /usr/local/bin/

Minikube Start

$ minikube start — driver=docker

```
sula@DESKTOP-CAO6MC9:~/virt$ minikube version
minikube version: v1.33.0
commit: 86fc9d54fca63f295d8737c8eacdbb7987e89c67
```

```
sula@DESKTOP-CAO6MC9:~/virt$ minikube version
minikube version: v1.33.0
commit: 86fc9d54fca63f295d8737c8eacdbb7987e89c67
sula@DESKTOP-CAO6MC9:~/virt$ kubectl version
WARNING: This version information is deprecated and will be replaced with the output from kubectl version --short.  Use
--output=yaml|json to get the full version.
Client Version: version.Info{Major:"1", Minor:"24", GitVersion:"v1.24.7", GitCommit:"e6f35974b08862a23e7f4aad8e5d7f7f2de
26c15", GitTreeState:"clean", BuildDate:"2022-10-12T10:57:14Z", GoVersion:"go1.18.7", Compiler:"gc", Platform:"linux/amd
64"}
Kustomize Version: v4.5.4
Server Version: version.Info{Major:"1", Minor:"30", GitVersion:"v1.30.0", GitCommit:"7c48c2bd72b9bf5c44d21d7338cc7bea77d
0ad2a", GitTreeState:"clean", BuildDate:"2024-04-17T17:27:03Z", GoVersion:"go1.22.2", Compiler:"gc", Platform:"linux/amd
64"}
```

```
sula@DESKTOP-CAO6MC9:~/virt$ kubectl cluster-info
Kubernetes control plane is running at https://127.0.0.1:61533
CoreDNS is running at https://127.0.0.1:61533/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```

We created main deployment parts like:

- **Customer**
- **Customerdb**
- **Password**
- **Passworddb**
- **Product**
- **Producdb**

1. Customer pod should contain docker container which has Customers information analysis, caching, and request handlers

2. Customer db pod is an external GCS bucket which contains tables for storage of the information about customer

3. Password pod is used for security approach, to prevent any private information leaking, also for overall safety

4. Password db pod is a local storage which contains tables for storage of the information about customers password and private cookies

5. Product pod should contain docker container which has Product information analysis, caching, and request handlers

6. Product db pod is an external GCS bucket which contains tables for storage of the information about product

There you can see the coding part of the deployment creation:

```
sula@DESKTOP-CAO6MC9:~$ cd virt
sula@DESKTOP-CAO6MC9:~/virt$ kubectl create deployment customer db --image=customer
error: exactly one NAME is required, got 2
See 'kubectl create deployment -h' for help and examples
sula@DESKTOP-CAO6MC9:~/virt$ kubectl create deployment customerdb --image=customer
deployment.apps/customerdb created
sula@DESKTOP-CAO6MC9:~/virt$ kubectl create deployment passworddb --image=customer
deployment.apps/passworddb created
sula@DESKTOP-CAO6MC9:~/virt$ kubectl create deployment productdb --image=customer
deployment.apps/productdb created
sula@DESKTOP-CAO6MC9:~/virt$ kubectl create deployment customerdb --image=customer
error: failed to create deployment: deployments.apps "customerdb" already exists
sula@DESKTOP-CAO6MC9:~/virt$ kubectl create deployment customer --image=customer
deployment.apps/customer created
sula@DESKTOP-CAO6MC9:~/virt$ kubectl create deployment product --image=customer
deployment.apps/product created
sula@DESKTOP-CAO6MC9:~/virt$ kubectl create deployment password --image=customer
deployment.apps/password created
```

Also there is a pods existing in this project:

```
sula@DESKTOP-CAO6MC9:~/virt$ kubectl get pods
NAME                           READY   STATUS             RESTARTS   AGE
customer-74979dd688-g6scs      0/1     ImagePullBackOff   0          98s
customerdb-687d45dd78-q7br6    0/1     ImagePullBackOff   0          2m31s
nginx-depl-85c9d7c5f4-t9s5l    1/1     Running            0          10s
password-859b966bfc-jscjr      0/1     ImagePullBackOff   0          51s
passworddb-5cd9795759-kjq98    0/1     ImagePullBackOff   0          2m11s
product-6c6d788895-pfs7j       0/1     ImagePullBackOff   0          87s
productdb-7875bc94b9-t7krd     0/1     ImagePullBackOff   0          2m
```

Here you can find every pod and deployment in projects cluster

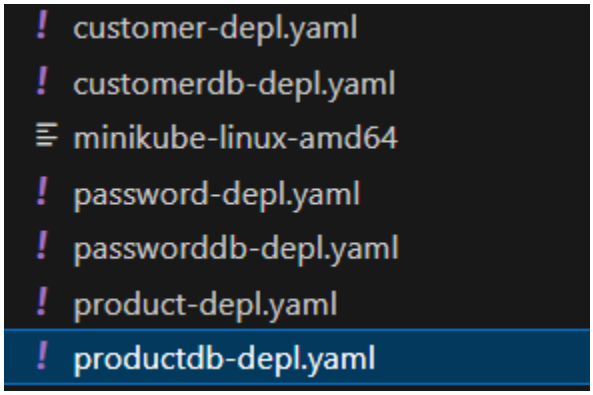
```
sula@DESKTOP-CAO6MC9:~/virt$ kubectl get all
NAME                               READY   STATUS             RESTARTS   AGE
pod/customer-74979dd688-g6scs      0/1     ImagePullBackOff   0          80m
pod/customerdb-687d45dd78-q7br6    0/1     ImagePullBackOff   0          81m
pod/nginx-depl-85c9d7c5f4-t9s5l    1/1     Running            0          78m
pod/password-859b966bfc-jscjr      0/1     ImagePullBackOff   0          79m
pod/passworddb-5cd9795759-kjq98    0/1     ImagePullBackOff   0          80m
pod/product-6c6d788895-pfs7j       0/1     ImagePullBackOff   0          79m
pod/productdb-7875bc94b9-t7krd     0/1     ImagePullBackOff   0          80m

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   42h

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/customer       0/1     1            0           80m
deployment.apps/customerdb     0/1     1            0           81m
deployment.apps/nginx-depl     1/1     1            1           42h
deployment.apps/password       0/1     1            0           79m
deployment.apps/passworddb     0/1     1            0           80m
deployment.apps/product        0/1     1            0           79m
deployment.apps/productdb      0/1     1            0           80m

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/customer-74979dd688       1         1         0       80m
replicaset.apps/customerdb-687d45dd78     1         1         0       81m
replicaset.apps/nginx-depl-85c9d7c5f4     1         1         1       42h
replicaset.apps/password-859b966bfc       1         1         0       79m
replicaset.apps/passworddb-5cd9795759     1         1         0       80m
replicaset.apps/product-6c6d788895        1         1         0       79m
replicaset.apps/productdb-7875bc94b9      1         1         0       80m
```

There you can see the list of deployment .yaml



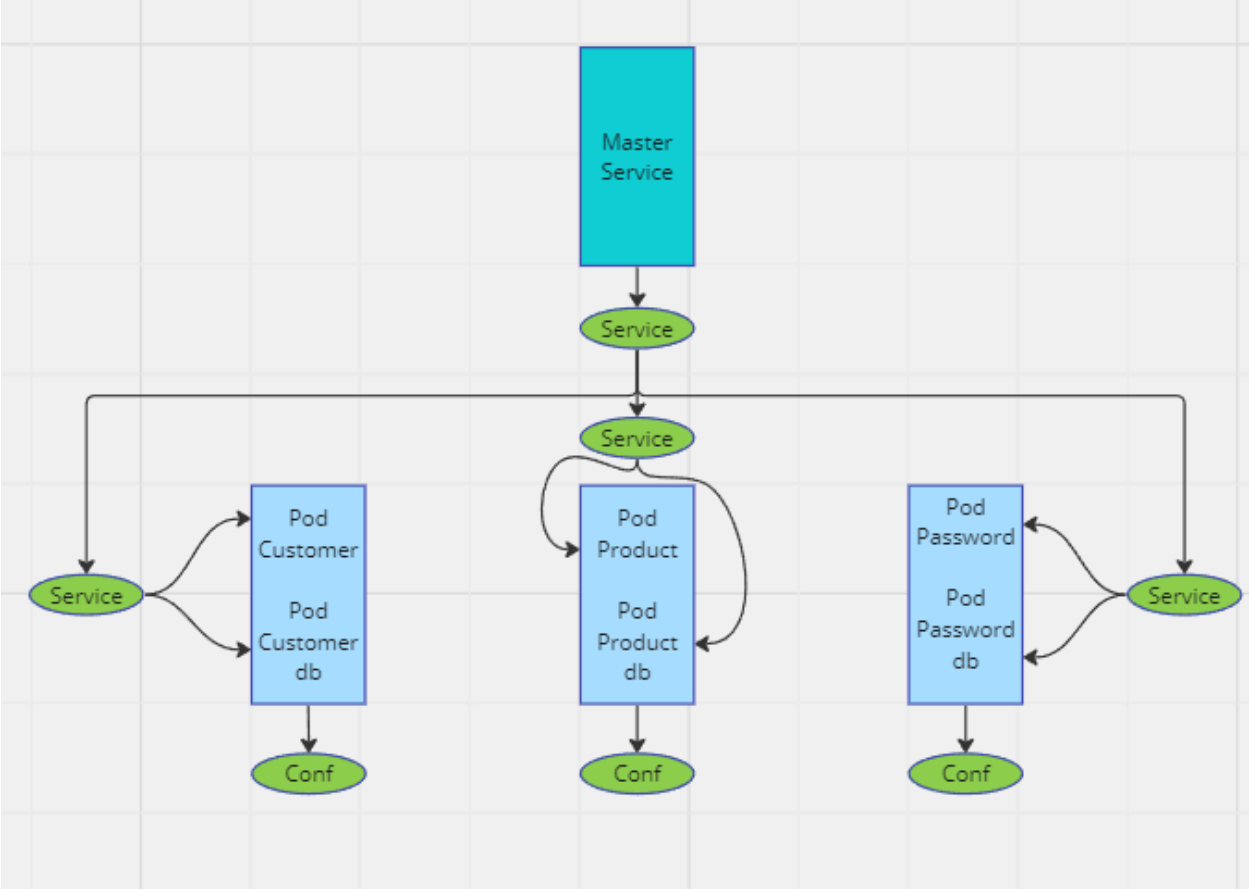
Example: Customer-depl.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: customer-depl
  labels:
    app: customer
spec:
  replicas: 2
  selector:
    matchLabels:
      app: customer
  template:
    metadata:
      labels:
        app: customer
    spec:
      containers:
      - name: customer
        image: cutomer-serve
        ports:
        - containerPort:
        ---
apiVersion: v1
kind: Service
metadata:
  name: customer-service
  labels:
    app: customer
spec:
  selector:
    app: customer
```

```
type: NodePort
ports:
- port: 80
  nodePort: 31364
  targetPort: 8006
  protocol: TCP
  name: http
```

- Microservices Architecture Design (5 points):

We already created our Pods, each is placed in deployment and has deployment.yaml, so called deployment configurations. Each pod connected to each other through Service. So here we are done with the creating and architecture of the cluster.



Master Controller is responsible for correct work of each deployment and pod, for API requests, defines which API request should be handled by a certain pod or deployment. Does health check for every part of the cluster and detects microservices fall, preventing it by replacing defective pod by its replica

Service is a Port handler and communicator between parts of the cluster like Pods, Master, Deployments. Service manages what will be sent to a certain pod and transports the answers.

Configuration is used to deploy and change everything the pod consists of. For example: its image, app, type, port, and etc.

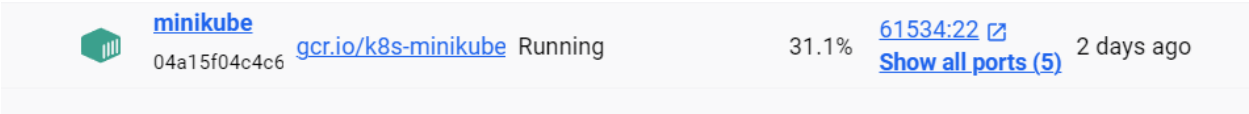
Eventually the whole cluster is stable and works correctly and can work without microservices fall, Master controls and does self-healing work and simultaneously checks the health of each pod all running time, which allows cluster non-stop workflow.

- Containerization with Docker (30 points):

Every pod in the cluster has its own image created in docker, also it can be updated by docker while a replica of the pod will hold the process smooth.

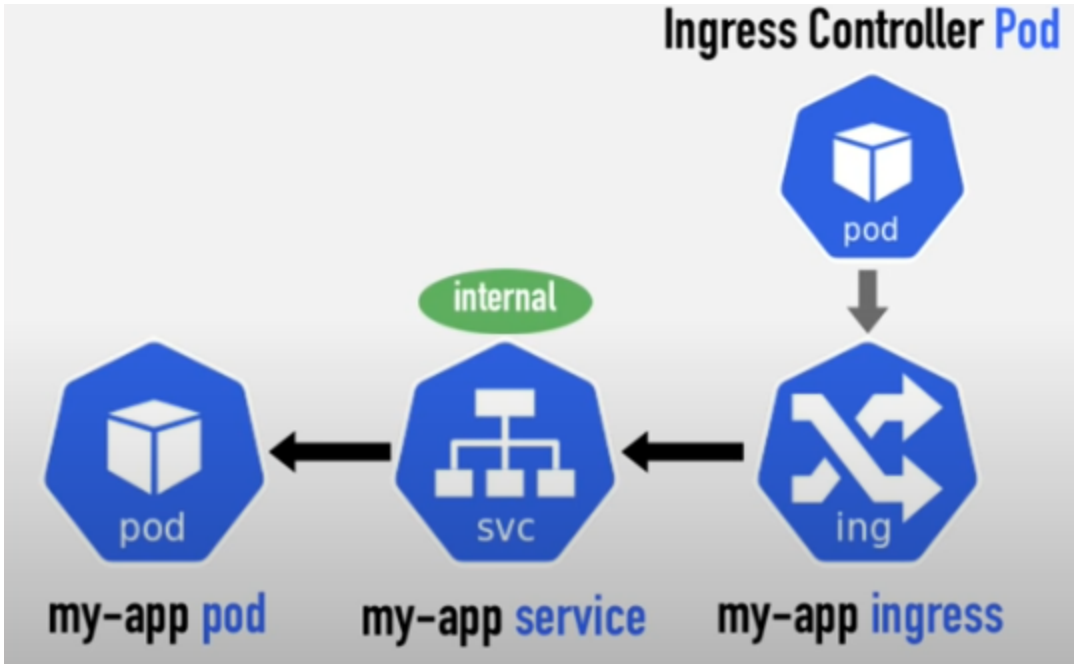
We can see our image in the deployment configurations
Example:

```
spec:
  containers:
  - name: customer
    image: cutomer-serve
    ports:
    - containerPort:
```



- Scaling and Load Balancing (10 points):

Here you can see the Ingress work system:



So we enabled Ingress on our cluster successfully:

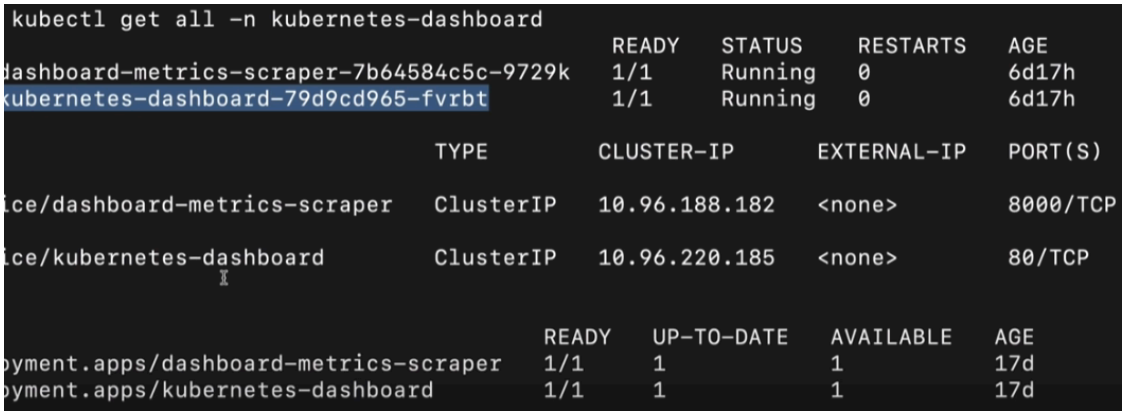
```
minikube addons enable ingress
ingress was successfully enabled
```

Here you can see the code for Ingress Dashboard

dashboard-ingress.yaml

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: dashboard-ingress
  namespace: kubernetes-dashboard
spec:
  rules:
  - host: dashboard.com
    http:
      paths:
      - backend:
          serviceName: kubernetes-dashboard
          servicePort: 80
```

Here you can see the dashboard with all dependencies:



Here you can see some basic Ingress functions: