CLD Lab 05: KUBERNETES

Authors: Kevin Auberson and Léo Zmoos - L5GrR

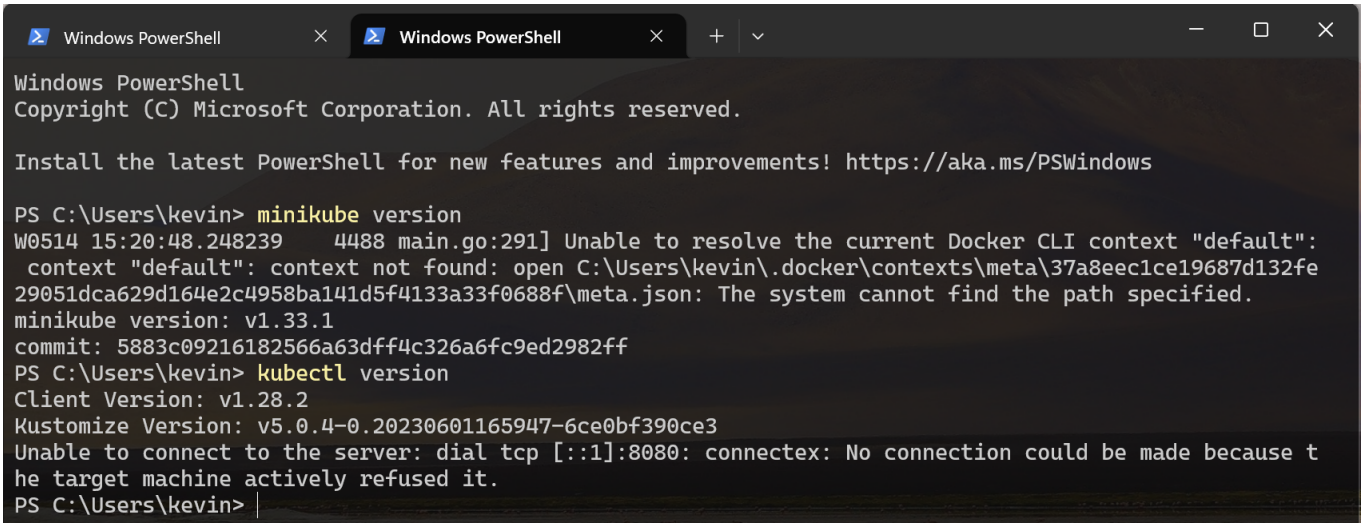
Date: May 7, 2024

TASK 1 - DEPLOY THE APPLICATION ON A LOCAL TEST CLUSTER

> Document any difficulties you faced and how you overcame them. Copy the object descriptions into the lab report.

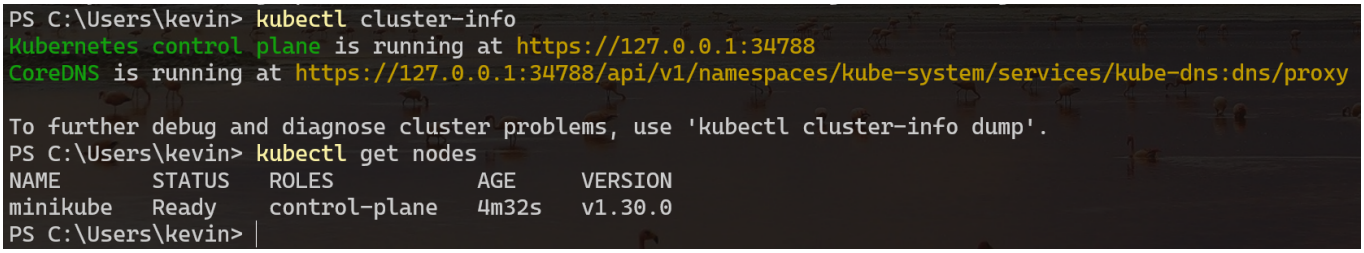
1.1 & 1.2- Installation of Minikube & Kubectl

We had no problem installing Minikube and kubectl. Here is the screenshot showing the installed version :



1.3 - Create a one-node cluster on your local machine

The cluster creation process was easy to follow, and we did not have any issue doing it. This screenshot shows the cluster information, once it has been created.



1.4 - Deploy the application

Once again, we didn't encounter any issue deploying the application.
This screenshot shows the Redis deployment with the `redis-svc` and `redis-pod` with the config files :

```
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl create -f redis-svc.yaml
service/redis-svc created
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl create -f redis-pod.yaml
pod/redis created
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl get all
NAME        READY   STATUS    RESTARTS   AGE
pod/redis   1/1     Running   0          25s

NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP    23h
service/redis-svc    ClusterIP   10.109.240.191   <none>        6379/TCP   43s
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis>
```

The following screenshots show the description of the redis service and pod :

```
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl describe pod/redis
Name:             redis
Namespace:        default
Priority:         0
Service Account:  default
Node:             minikube/192.168.49.2
Start Time:       Wed, 15 May 2024 15:21:48 +0200
Labels:           app=todo
                  component=redis
Annotations:      <none>
Status:           Running
IP:               10.244.0.4
IPs:
  IP:  10.244.0.4
Containers:
  redis:
    Container ID:  docker://1f84e32df1315210c0c5e23f8341e1ad506ec6bdccbbc8392b313651bad908db
    Image:         redis
    Image ID:      docker-pullable://redis@sha256:bf2eef6365155332a8a9f86255818c8cef43f1ebb70
ed0335712d596662c1510
    Port:          6379/TCP
    Host Port:     0/TCP
    Args:
      redis-server
      --requirepass ccp2
      --appendonly yes
    State:          Running
      Started:      Wed, 15 May 2024 15:21:58 +0200
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-ww6vv (ro)
```

```
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  kube-api-access-ww6vv:
    Type:                     Projected (a volume that contains injected data from multiple so
urces)
    TokenExpirationSeconds:   3607
    ConfigMapName:            kube-root-ca.crt
    ConfigMapOptional:        <nil>
    DownwardAPI:              true
QoS Class:                    BestEffort
Node-Selectors:               <none>
Tolerations:                  node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                              node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  76s   default-scheduler  Successfully assigned default/redis to minikube
  Normal  Pulling    76s   kubelet            Pulling image "redis"
  Normal  Pulled     67s   kubelet            Successfully pulled image "redis" in 8.376s (8.
376s including waiting). Image size: 116496163 bytes.
  Normal  Created    67s   kubelet            Created container redis
  Normal  Started    67s   kubelet            Started container redis
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis>
```

```
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl describe svc/redis-svc
Name:              redis-svc
Namespace:         default
Labels:            component=redis
Annotations:       <none>
Selector:          app=todo,component=redis
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.109.240.191
IPs:               10.109.240.191
Port:              redis  6379/TCP
TargetPort:        6379/TCP
Endpoints:         10.244.0.4:6379
Session Affinity:  None
Events:            <none>
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis>
```

Deploy the ToDo-API Service and Pod

We created the api-svc config:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: api-svc
  labels:
    component: api
```

```yaml
spec:
  ports:
  - port: 8081
    targetPort: 8081
    name: api
  selector:
    app: todo
    component: api
  type: ClusterIP
```

The following screenshot shows the deployment of the `api-svc` with the config file :

```
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl create -f .\api-svc.yml
service/api-svc created
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl create -f .\api-pod.yaml
pod/api created
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl get all
NAME           READY   STATUS    RESTARTS   AGE
pod/api        1/1     Running   0          5m53s
pod/frontend   1/1     Running   0          21m
pod/redis      1/1     Running   0          112m

NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/api-svc      ClusterIP   10.103.38.133    <none>        8081/TCP   81m
service/kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP    25h
service/redis-svc    ClusterIP   10.109.240.191   <none>        6379/TCP   112m
```

We can see that the service exposes the port 8081. There is already the frontend/pod because when I did this operation I forgot to deploy the `api-pod.yaml`.

The following screenshot shows the description of the api service and pod:

```
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl describe svc/api-svc
Name:              api-svc
Namespace:         default
Labels:            component=api
Annotations:       <none>
Selector:          app=todo,component=api
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.103.38.133
IPs:               10.103.38.133
Port:              api  8081/TCP
TargetPort:        8081/TCP
Endpoints:         10.244.0.6:8081
Session Affinity:  None
Events:            <none>
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl describe pod/api
Name:             api
Namespace:        default
Priority:         0
Service Account:  default
Node:             minikube/192.168.49.2
Start Time:       Wed, 15 May 2024 17:07:59 +0200
Labels:           app=todo
                  component=api
Annotations:      <none>
Status:           Running
IP:               10.244.0.6
IPs:
  IP:  10.244.0.6
Containers:
  api:
```

```
    Container ID:   docker://ab7d6a378a24efb1df1def192da194db61ff57f11c917053fed3b5af2eb7cb49
    Image:          icclabcna/ccp2-k8s-todo-api
    Image ID:       docker-pullable://icclabcna/ccp2-k8s-todo-api@sha256:13cb50bc9e93fdf10b46
08f04f2966e274470f00c0c9f60815ec8fc987cd6e03
    Port:           8081/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Wed, 15 May 2024 17:08:06 +0200
    Ready:          True
    Restart Count:  0
    Environment:
      REDIS_ENDPOINT:  redis-svc
      REDIS_PWD:       ccp2
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-6p2c7 (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  kube-api-access-6p2c7:
    Type:                    Projected (a volume that contains injected data from multiple so
urces)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  13m   default-scheduler  Successfully assigned default/api to minikube
  Normal  Pulling    13m   kubelet            Pulling image "icclabcna/ccp2-k8s-todo-api"
  Normal  Pulled     13m   kubelet            Successfully pulled image "icclabcna/ccp2-k8s-t
odo-api" in 6.221s (6.221s including waiting). Image size: 683793243 bytes.
  Normal  Created    13m   kubelet            Created container api
  Normal  Started    13m   kubelet            Started container api
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis>
```

Deploy the Frontend Pod

Here is our frontend-api configuration file. The `API_ENDPOINT_URL` environment variable should be set to the address of our API service within the Kubernetes cluster, so the URL would be `http://api-svc:8081`.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
  labels:
    component: frontend
    app: todo
```

```yaml
spec:
  containers:
  - name: frontend
    image: icclabcna/ccp2-k8s-todo-frontend
    ports:
    - containerPort: 8080
    env:
    - name: API_ENDPOINT_URL
      value: http://api-svc:8081
```

Now we just have to deploy the frontend pod :

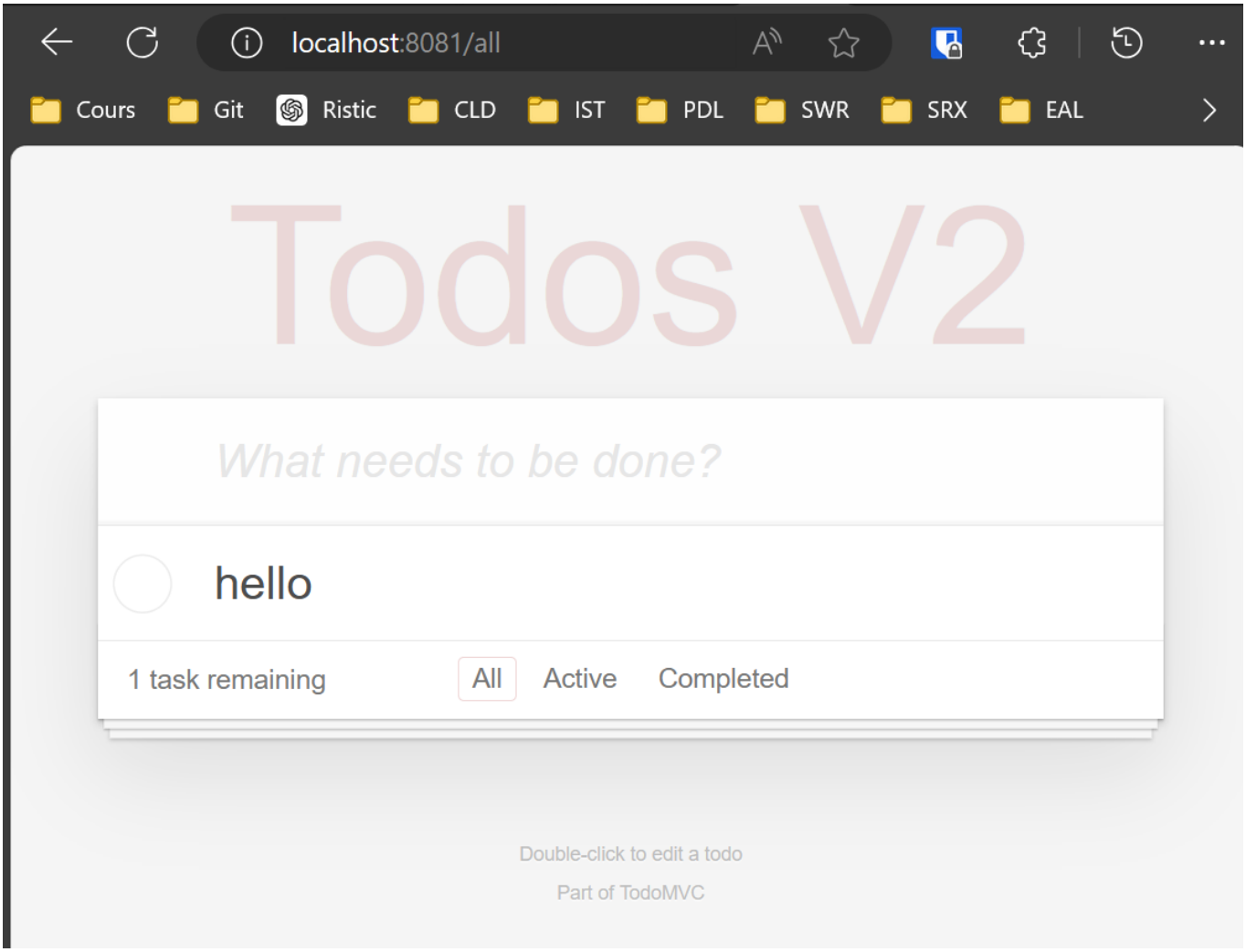
```
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl create -f .\frontend-pod.yml
pod/frontend created
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl get all
NAME           READY   STATUS              RESTARTS   AGE
pod/frontend   0/1     ContainerCreating   0          12s
pod/redis      1/1     Running             0          91m

NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/api-svc      ClusterIP   10.103.38.133    <none>        8081/TCP   60m
service/kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP    25h
service/redis-svc    ClusterIP   10.109.240.191   <none>        6379/TCP   91m
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis>
```

The following screenshot shows the description of the frontend pod :

```
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl describe pod/frontend
Name:             frontend
Namespace:        default
Priority:         0
Service Account:  default
Node:             minikube/192.168.49.2
Start Time:       Wed, 15 May 2024 16:52:51 +0200
Labels:           app=todo
                  component=frontend
Annotations:      <none>
Status:           Running
IP:               10.244.0.5
IPs:
  IP:  10.244.0.5
Containers:
  frontend:
    Container ID:   docker://28c8ad6a727cd4d4906d4b0119c7f38ce58f56e3a7d52e96d0b4e7ddfc63bf55
    Image:          icclabcna/ccp2-k8s-todo-frontend
    Image ID:       docker-pullable://icclabcna/ccp2-k8s-todo-frontend@sha256:5892b8f75a4dd3a
a9d9cf527f8796a7638dba574ea8e6beef49360a3c67bbb44
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Wed, 15 May 2024 16:53:31 +0200
    Ready:          True
    Restart Count:  0
    Environment:
      API_ENDPOINT_URL:  http://api-svc:8081
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-xs4lr (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  kube-api-access-xs4lr:
    Type:                    Projected (a volume that contains injected data from multiple so
urces)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  3m19s  default-scheduler  Successfully assigned default/frontend to mini
kube
  Normal  Pulling    3m18s  kubelet            Pulling image "icclabcna/ccp2-k8s-todo-fronten
d"
  Normal  Pulled     2m39s  kubelet            Successfully pulled image "icclabcna/ccp2-k8s-
todo-frontend" in 39.432s (39.433s including waiting). Image size: 746900794 bytes.
  Normal  Created    2m39s  kubelet            Created container frontend
  Normal  Started    2m39s  kubelet            Started container frontend
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis>
```

Verify the ToDo application

Then, using the kubectl port forwarding `kubectl port-forward frontend 8081:8080`, we can access the web app and see that it is served properly :



TASK 2 - DEPLOY THE APPLICATION IN KUBERNETES ENGINE

> Document any difficulties you faced and how you overcame them. Copy the object descriptions into the lab report (if they are unchanged from the previous task just say so).

2.1 - Create Project & 2.2 Create a cluster

> Take a screenshot of the cluster details from the GKE console.

We didn't have any issue creating the cluster in GKE.

Cluster basics

Name	gke-cluster-1	
Location type	Zonal	
Control plane zone	europe-west1-b	
Default node zones	europe-west1-b	
Release channel	Regular channel	UPGRADE AVAILABLE
Version	1.28.7-gke.1026000	
Total size	2	
External endpoint	34.77.235.255 Show cluster certificate	
Internal endpoint	10.132.0.9 Show cluster certificate	

Automation

Maintenance window	At any time	
Maintenance exclusions	None	
Notifications	Disabled	
Vertical Pod Auto-scaling	Disabled	
Node auto-provisioning	Disabled	
Auto-provisioning network tags		
Auto-scaling profile	Balanced	

Networking

Private cluster	Disabled	🔒
Default SNAT	Enabled	✎
Control plane global access	Disabled	✎
Network	default	🔒
Subnet	default	🔒
Stack type	IPv4	✎
Private control plane's endpoint subnet	default	🔒
VPC-native traffic routing	Enabled	🔒
Cluster pod IPv4 range (default)	10.12.0.0/14	🔒
Cluster pod IPv4 ranges (additional) ?	None	✎
Maximum pods per node	110	🔒
IPv4 service range	10.63.192.0/20	🔒
Intranode visibility	Disabled	✎
HTTP load balancing	Enabled	✎
Subsetting for L4 internal load balancers	Disabled	✎
Control plane authorised networks	Disabled	✎
Calico Kubernetes network policy	Disabled	✎
Dataplane V2	Disabled	🔒

Dataplane V2 metrics	Disabled	
Dataplane V2 observability	Disabled	
DNS provider	Kube-dns	
NodeLocal DNSCache	Disabled	
Gateway API	Disabled	
Multi-networking	Disabled	

Security

Binary authorisation	Disabled	
Shielded GKE nodes	Enabled	
Confidential GKE Nodes	Disabled	
Application-layer secrets encryption	Disabled	
Workload Identity	Disabled	
Google Groups for RBAC	Disabled	
Legacy authorisation	Disabled	
Basic authentication	Disabled	
Client certificate	Disabled	
Configuration auditing	Enabled	
Workload vulnerability scanning	Disabled	

Metadata

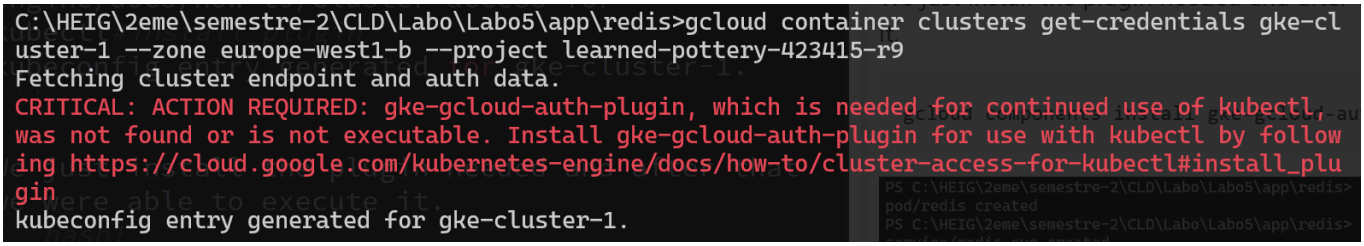
Description	None	
Labels	None	
Tags	None	

Features

Logging	System, Workloads View Logs	
Cloud Monitoring	System View GKE dashboard	
Managed service for Prometheus	Enabled	
Cloud TPU	Disabled	
Kubernetes alpha features	Disabled	
Cost allocation	Disabled	
GKE usage metering	Disabled	
Backup for GKE	Disabled	
Config Connector	Disabled	
Compute Engine persistent disk CSI Driver	Enabled	
Image streaming	Disabled	
Filestore CSI driver	Disabled	
Cloud Storage FUSE CSI driver	Disabled	
Service Mesh	Disabled	

2.3 - Deploy the application on the cluster

We had this error when we wanted to run the give command from the Kubernetes cluster dialog box.



So, we just install the plugin needed and after that we were able to execute the command.

```
gcloud components install gke-gcloud-auth-plugin
```

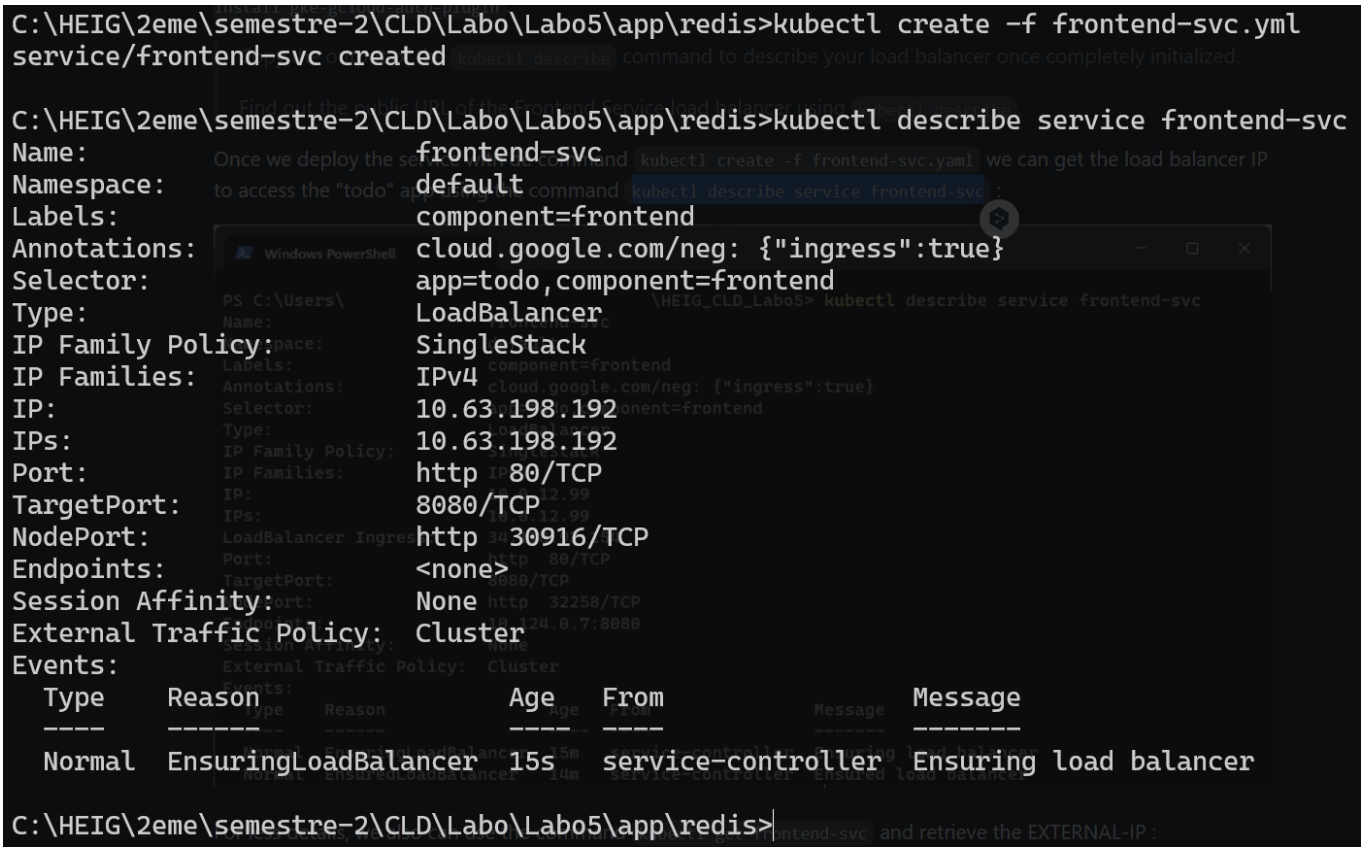
Then we did not encounter any problems deploying the pods and services on the GKE cluster. The command `kubectl get all` give the same output like the task 1 on subtask deploy frontend pod.

2.4 - Deploy the ToDo-Frontend Service

First, we created the frontend-svc.yaml configuration file :

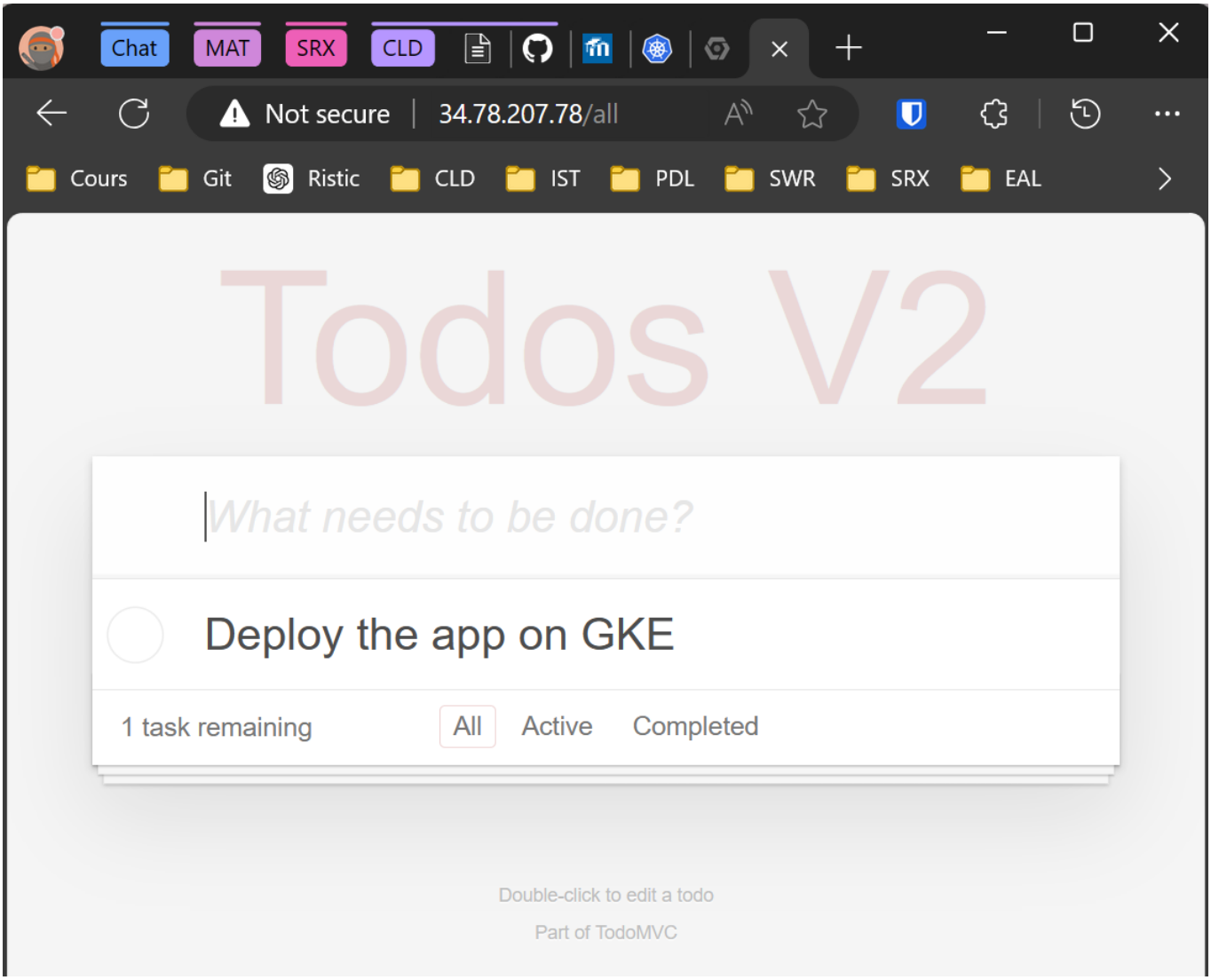
```
apiVersion: v1
kind: Service
metadata:
  labels:
    component: frontend
  name: frontend-svc
spec:
  ports:
  - port: 80
    targetPort: 8080
    protocol: TCP
        name: http
  selector:
    app: todo
    component: frontend
  type: LoadBalancer
```

Then, we deploy the service with de command kubectl create -f frontend-svc.yaml we can get the load balancer IP to access the "todo" app using the command kubectl describe service frontend-svc



Verify the ToDo application

Finally we can access the web app and see that it is served properly :



TASK 3 - ADD AND EXERCISE RESILIENCE

3.1 Add deployments

Firstly, we remove the existing pods with the commands:

```
kubectl delete pod api
kubectl delete pod redis
kubectl delete pod frontend
kubectl get pods # this is to verify
```

Then, we had to create the 3 deployment configurations as follow.

The `redis-deploy.yml` file config:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-deploy
  labels:
    component: redis
    app: todo
spec:
  replicas: 1
  selector:
    matchLabels:
      component: redis
      app: todo
  template:
    metadata:
      labels:
        component: redis
        app: todo
    spec:
      containers:
      - name: redis
        image: redis
        ports:
        - containerPort: 6379
        args:
        - redis-server
        - --requirepass ccp2
        - --appendonly yes
```

The `api-deploy.yml` file config:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-deploy
  labels:
    component: api
    app: todo
spec:
  replicas: 2
  selector:
    matchLabels:
      component: api
      app: todo
  template:
```

```
    metadata:
      labels:
        component: api
        app: todo
    spec:
      containers:
      - name: api
        image: icclabcna/ccp2-k8s-todo-api
        ports:
        - containerPort: 8081
        env:
        - name: REDIS_ENDPOINT
          value: redis-svc
        - name: REDIS_PWD
          value: ccp2
```

The `frontend-deploy.yml` file config:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deploy
  labels:
    component: frontend
    app: todo
spec:
  replicas: 2
  selector:
    matchLabels:
      component: frontend
      app: todo
  template:
    metadata:
      labels:
        component: frontend
        app: todo
    spec:
      containers:
      - name: frontend
        image: icclabcna/ccp2-k8s-todo-frontend
        ports:
        - containerPort: 8080
        env:
        - name: API_ENDPOINT_URL
          value: http://api-svc:8081
```

> Use only 1 instance for the Redis-Server. Why?

Redis is a database and running multiple instances without proper clustering can lead to data inconsistency and potential conflicts, as each instance would not share the same state. For simplicity and to avoid the complexity of setting up a Redis cluster, a single instance is sufficient.

The next step is to deploy the deployment using the `kubectl apply` command and then verify their availability :

```
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl apply -f .\frontend-deploy.yml
deployment.apps/frontend-deploy created
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl apply -f .\api-deploy.yml
deployment.apps/api-deploy created
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl apply -f .\redis-deploy.yml
deployment.apps/redis-deploy created
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl get deployments
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
api-deploy        0/2     2            0           17s
frontend-deploy   2/2     2            2           27s
redis-deploy      1/1     1            1           11s
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis>
```

All ressources:

```
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl get all
NAME                                  READY   STATUS              RESTARTS   AGE
pod/api-deploy-664fbdf7d9-9fndv       1/1     Running             0          16s
pod/api-deploy-664fbdf7d9-msbxc       0/1     ContainerCreating   0          16s
pod/frontend-deploy-dc6bd967f-n4pzv   0/1     ContainerCreating   0          5s
pod/frontend-deploy-dc6bd967f-rgzpn   1/1     Running             0          5s
pod/redis-deploy-56fb88dd96-gcbbq     1/1     Running             0          24s

NAME                   TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)        AGE
service/api-svc        ClusterIP      10.63.198.61    <none>         8081/TCP       2d11h
service/frontend-svc   LoadBalancer   10.63.198.192   34.78.207.78   80:30916/TCP   2d11h
service/kubernetes     ClusterIP      10.63.192.1     <none>         443/TCP        2d13h
service/redis-svc      ClusterIP      10.63.204.157   <none>         6379/TCP       2d12h

NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/api-deploy           1/2     2            1           17s
deployment.apps/frontend-deploy      1/2     2            1           5s
deployment.apps/redis-deploy         1/1     1            1           25s

NAME                                            DESIRED   CURRENT   READY   AGE
replicaset.apps/api-deploy-664fbdf7d9           2         2         1       16s
replicaset.apps/frontend-deploy-dc6bd967f       2         2         1       5s
replicaset.apps/redis-deploy-56fb88dd96         1         1         1       25s
```

"frontend-deploy" description:

```
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl describe deployment frontend
Name:                   frontend-deploy
Namespace:              default
CreationTimestamp:      Thu, 16 May 2024 14:39:57 +0200
Labels:                 app=todo
                        component=frontend
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=todo,component=frontend
Replicas:               2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=todo
           component=frontend
  Containers:
   frontend:
    Image:      icclabcna/ccp2-k8s-todo-frontend
    Port:       8080/TCP
    Host Port:  0/TCP
    Environment:
      API_ENDPOINT_URL:  http://api-svc:8081
    Mounts:              <none>
  Volumes:               <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   frontend-deploy-87855cddb (2/2 replicas created)
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  2m14s  deployment-controller  Scaled up replica set frontend
-deploy-87855cddb to 2
```

"api-deploy" description:

```
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl describe deployment api
Name:                   api-deploy
Namespace:              default
CreationTimestamp:      Thu, 16 May 2024 14:40:07 +0200
Labels:                 app=todo
                        component=api
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=todo,component=api
Replicas:               2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=todo
           component=api
  Containers:
   api:
    Image:      icclabcna/ccp2-k8s-todo-api
    Port:       8081/TCP
    Host Port:  0/TCP
    Environment:
      REDIS_ENDPOINT:  redis-svc
      REDIS_PWD:       ccp2
    Mounts:            <none>
  Volumes:             <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   api-deploy-67fdf6545d (2/2 replicas created)
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  2m10s  deployment-controller  Scaled up replica set api-depl
oy-67fdf6545d to 2
```

"redis-deploy" description:

```
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl describe deployment redis
Name:                   redis-deploy
Namespace:              default
CreationTimestamp:      Thu, 16 May 2024 14:40:13 +0200
Labels:                 app=todo
                        component=redis
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=todo,component=redis
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=todo
           component=redis
  Containers:
   redis:
    Image:      redis
    Port:       6379/TCP
    Host Port:  0/TCP
    Args:
      redis-server
      --requirepass ccp2
      --appendonly yes
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   redis-deploy-6fc58795fc (1/1 replicas created)
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  2m10s  deployment-controller  Scaled up replica set redis-de
ploy-6fc58795fc to 1
```

3.2 Verify the functionality of replica set

> What happens if you delete a Frontend or API Pod?

1. We execute the command `kubectl delete pod <pod-name>`.
2. The API Server receives and processes the deletion request, updating the Pod's status to "Terminating".
3. The Controller Manager (ReplicaSet) detects the desired number of replicas is not met and initiates the creation of a new Pod.
4. The Kubernetes scheduler assigns the new Pod to a node.
5. The kubelet on that node starts the Pod and after check up the app is available again.

> How long does it take for the system to react?

We can observe in the screenshot below that when a pod is deleted, it is automatically recreated after deletion. The "api" pod was recreated within 6 seconds and the "frontend" pod within 3 seconds.

```
PS C:\Users\kevin> kubectl get pods --watch
NAME                              READY   STATUS    RESTARTS      AGE
api-deploy-67fdf6545d-mjhzp       1/1     Running   2 (75m ago)   75m
api-deploy-67fdf6545d-s4xcz       1/1     Running   2 (75m ago)   75m
frontend-deploy-87855cddb-5kmv2   1/1     Running   0             75m
frontend-deploy-87855cddb-6zz7g   1/1     Running   0             75m
redis-deploy-6fc58795fc-wwwfv     1/1     Running   0             75m
api-deploy-67fdf6545d-mjhzp       1/1     Terminating   2 (77m ago)   77m
api-deploy-67fdf6545d-ts2dv       0/1     Pending       0             0s
api-deploy-67fdf6545d-ts2dv       0/1     Pending       0             0s
api-deploy-67fdf6545d-ts2dv       0/1     ContainerCreating   0             0s
api-deploy-67fdf6545d-ts2dv       1/1     Running             0             6s
api-deploy-67fdf6545d-mjhzp       0/1     Terminating         2 (78m ago)   78m
api-deploy-67fdf6545d-mjhzp       0/1     Terminating         2 (78m ago)   78m
api-deploy-67fdf6545d-mjhzp       0/1     Terminating         2 (78m ago)   78m
api-deploy-67fdf6545d-mjhzp       0/1     Terminating         2 (78m ago)   78m
frontend-deploy-87855cddb-5kmv2   1/1     Terminating         0             78m
frontend-deploy-87855cddb-dvc87   0/1     Pending             0             0s
frontend-deploy-87855cddb-dvc87   0/1     Pending             0             0s
frontend-deploy-87855cddb-dvc87   0/1     ContainerCreating   0             0s
frontend-deploy-87855cddb-5kmv2   0/1     Terminating         0             78m
frontend-deploy-87855cddb-5kmv2   0/1     Terminating         0             78m
frontend-deploy-87855cddb-5kmv2   0/1     Terminating         0             78m
frontend-deploy-87855cddb-5kmv2   0/1     Terminating         0             78m
frontend-deploy-87855cddb-dvc87   1/1     Running             0             3s
```

> What happens when you delete the Redis Pod?

Exactly the same as any other pod.

```
redis-deploy-6fc58795fc-wwwfv     1/1     Terminating         0             102m
redis-deploy-6fc58795fc-st559     0/1     Pending             0             0s
redis-deploy-6fc58795fc-st559     0/1     Pending             0             0s
redis-deploy-6fc58795fc-st559     0/1     ContainerCreating   0             0s
redis-deploy-6fc58795fc-wwwfv     0/1     Terminating         0             102m
redis-deploy-6fc58795fc-wwwfv     0/1     Terminating         0             102m
redis-deploy-6fc58795fc-wwwfv     0/1     Terminating         0             102m
redis-deploy-6fc58795fc-wwwfv     0/1     Terminating         0             102m
redis-deploy-6fc58795fc-st559     1/1     Running             0             3s
```

> How can you change the number of instances temporarily to 3? Hint: look for scaling in the deployment documentation

We can change the number of instances temporarily with the command `kubectl scale deployment <deployment-name> --replicas=3`, for example:

```
PS C:\Users\kevin> kubectl scale deployment api-deploy --replicas=3
deployment.apps/api-deploy scaled
PS C:\Users\kevin> kubectl get deployment
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
api-deploy        3/3     3            3           109m
frontend-deploy   2/2     2            2           110m
redis-deploy      1/1     1            1           109m
```

> What autoscaling features are available? Which metrics are used?

Kubernetes provides several powerful autoscaling features that help manage workloads efficiently on various metrics using the Horizontal Pod Autoscaler (HPA), Vertical Pod Autoscaler (VPA) or Cluster Autoscaler.

The Horizontal Pod Autoscaler (HPA) is a powerful feature in Kubernetes that automatically adjusts the number of Pods in a Deployment, ReplicaSet, or StatefulSet based on observed metrics such as CPU utilization or other custom metrics. This dynamic scaling ensures that applications can handle varying loads efficiently by increasing the number of Pods when demand is high and decreasing them when demand is low. By continuously monitoring the specified metrics, the HPA helps maintain optimal performance and resource utilization, providing a resilient and cost-effective way to manage workloads in a Kubernetes cluster.

The Vertical Pod Autoscaler (VPA) is a feature in Kubernetes that automatically adjusts the resource limits and requests for containers within a Pod. By analyzing historical and real-time data, the VPA ensures that each Pod has the appropriate amount of CPU and memory resources to handle its workload efficiently. This dynamic adjustment helps prevent resource bottlenecks and over-provisioning, optimizing the performance and cost-effectiveness of applications running in the Kubernetes cluster. By continuously tuning resource allocations, the VPA maintains the balance between application performance and resource usage, ensuring a more resilient and efficient deployment environment.

The Cluster Autoscaler is a feature in Kubernetes that automatically adjusts the size of the cluster by adding or removing nodes based on the scheduling needs of Pods. When there are pending Pods that cannot be scheduled due to insufficient resources, the Cluster Autoscaler increases the number of nodes to accommodate the workload. Conversely, if nodes are underutilized, the autoscaler can remove them to optimize costs. This dynamic adjustment ensures that the cluster always has the right number of nodes to handle current workloads efficiently, providing a balance between performance and cost-effectiveness.

> How can you update a component? (see “Updating a Deployment” in the deployment documentation)

Updating a deployment can be done by modifing the deployment's configuration file or applying these changes using `kubectl apply` command.

Using the `kubectl apply`, Kubernetes will automatically perform rolling updates to achieve the desired state by gradually replacing old Pods with new ones. This process ensures that the application remains available throughout the update.

If we are updating the image to deploy a new version of our application, we can use the kubectl set image command to update the image directly without editing the configuration file.

For example:

```
kubectl set image deployment/api-deploy api=icclabcna/ccp2-k8s-todo-api:v2
```

This command updates the api container in the `api-deploy` deployment to use the `icclabcna/ccp2-k8s-todo-api:v2` image.

3.3 Put autoscaling in place and load-test it

> Document your observations in the lab report. Document any difficulties you faced and how you overcame them. Copy the object descriptions into the lab report.

Firstly, we ensure that the Metrics Server is Running. So, we execute the given command :

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

Secondly, we need to update the `frontend-deploy` file and specifically focused on setting up the appropriate resource requests and limits. These settings are crucial for the Horizontal Pod Autoscaler (HPA) to function correctly because the HPA uses these metrics to determine when to scale the Pods.

This is the frontend-deploy config file with the update :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deploy
  labels:
    component: frontend
    app: todo
spec:
  replicas: 2
  selector:
    matchLabels:
      component: frontend
```

```
      app: todo
  template:
    metadata:
      labels:
        component: frontend
        app: todo
    spec:
      containers:
      - name: frontend
        image: icclabcna/ccp2-k8s-todo-frontend
        ports:
        - containerPort: 8080
        env:
        - name: API_ENDPOINT_URL
          value: http://api-svc:8081
        resources:
          requests:
            cpu: 100m
          limits:
            cpu: 200m
```

- Resource Requests: This specifies the minimum amount of CPU resources the container needs. Kubernetes uses this value for scheduling purposes, ensuring that the node where the Pod is scheduled has at least this much CPU available.
- Resource Limits: This specifies the maximum amount of CPU resources the container can use. This limit ensures that the container does not consume more than the specified amount of CPU, preventing it from using excessive resources that could impact other containers on the same node.

Updating this config on Kubernetes Engine, we use the following command :

```
kubectl apply -f frontend-deploy.yml
```

Thirdly, we create the Horizontal Pod Autoscaler (HPA). Define the HPA configuration to set up autoscaling with a target CPU utilization of 30% and a range of 1 to 4 replicas.

We create the following config name `frontend-hpa.yml` :

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-hpa
spec:
  scaleTargetRef:
```

```
    apiVersion: apps/v1
    kind: Deployment
    name: frontend-deploy
  minReplicas: 1
  maxReplicas: 4
  targetCPUUtilizationPercentage: 30
```

We use the following command to deploy the auto-scaling :

```
kubectl apply -f frontend-hpa.yml
```

```
PS C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5\app\redis> kubectl apply -f frontend-hap.yml
horizontalpodautoscaler.autoscaling/frontend-hpa created
```

We can verify that the HPA is created and monitoring the CPU usage.

```
C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5>kubectl get hpa
NAME           REFERENCE                     TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
frontend-hpa   Deployment/frontend-deploy   0%/30%    1         4         1          4d11h
```

And also get detailed information about the HPA to ensure it is configured correctly.

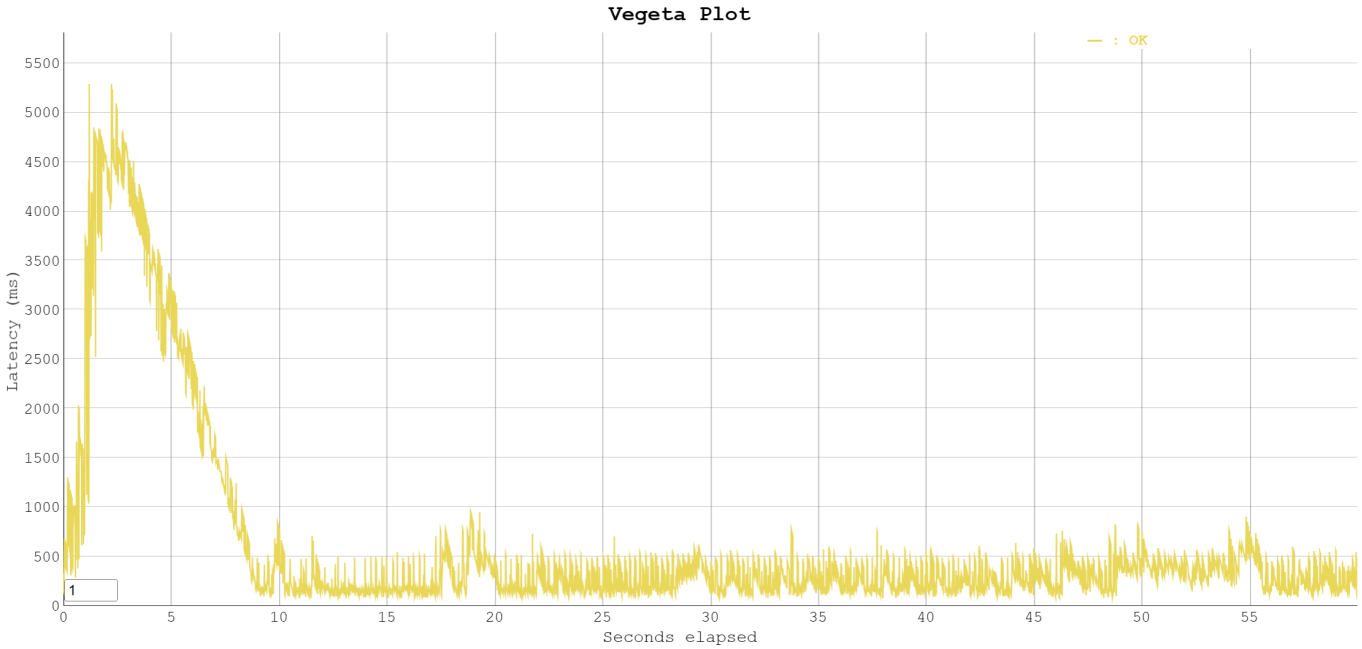
```
C:\HEIG\2eme\semestre-2\CLD\Labo\Labo5>kubectl describe hpa frontend-hpa
Name:                                                  frontend-hpa
Namespace:                                             default
Labels:                                                <none>
Annotations:                                           <none>
CreationTimestamp:                                     Sat, 18 May 2024 23:04:00 +0200
Reference:                                             Deployment/frontend-deploy
Metrics:                                               ( current / target )
  resource cpu on pods  (as a percentage of request):  10% (1m) / 30%
Min replicas:                                          1
Max replicas:                                          4
Deployment pods:                                       1 current / 1 desired
Conditions:
  Type            Status  Reason              Message
  ----            ------  ------              -------
  AbleToScale     True    ReadyForNewScale    recommended size matches current size
  ScalingActive   True    ValidMetricFound    the HPA was able to successfully calculate a replica count from cpu resour
ce utilization (percentage of request)
  ScalingLimited  False   DesiredWithinRange  the desired count is within the acceptable range
Events:           <none>
```

At least, we need to test this configuration to ensure that is running good. So, we do this with the vegeta :

rate 100

```
 echo "GET http://34.78.207.78/all" |
vegeta attack -duration=60s -rate=100 | tee results.bin | vegeta report
Requests      [total, rate, throughput]         6000, 100.02, 99.19
Duration      [total, attack, wait]             1m0s, 59.99s, 496.907ms
Latencies     [min, mean, 50, 90, 95, 99, max]  72.966ms, 628.108ms,
290.207ms, 1.576s, 3.463s, 4.631s, 5.33s
Bytes In      [total, mean]                     3786000, 631.00
Bytes Out     [total, mean]                     0, 0.00
Success       [ratio]                           100.00%
```

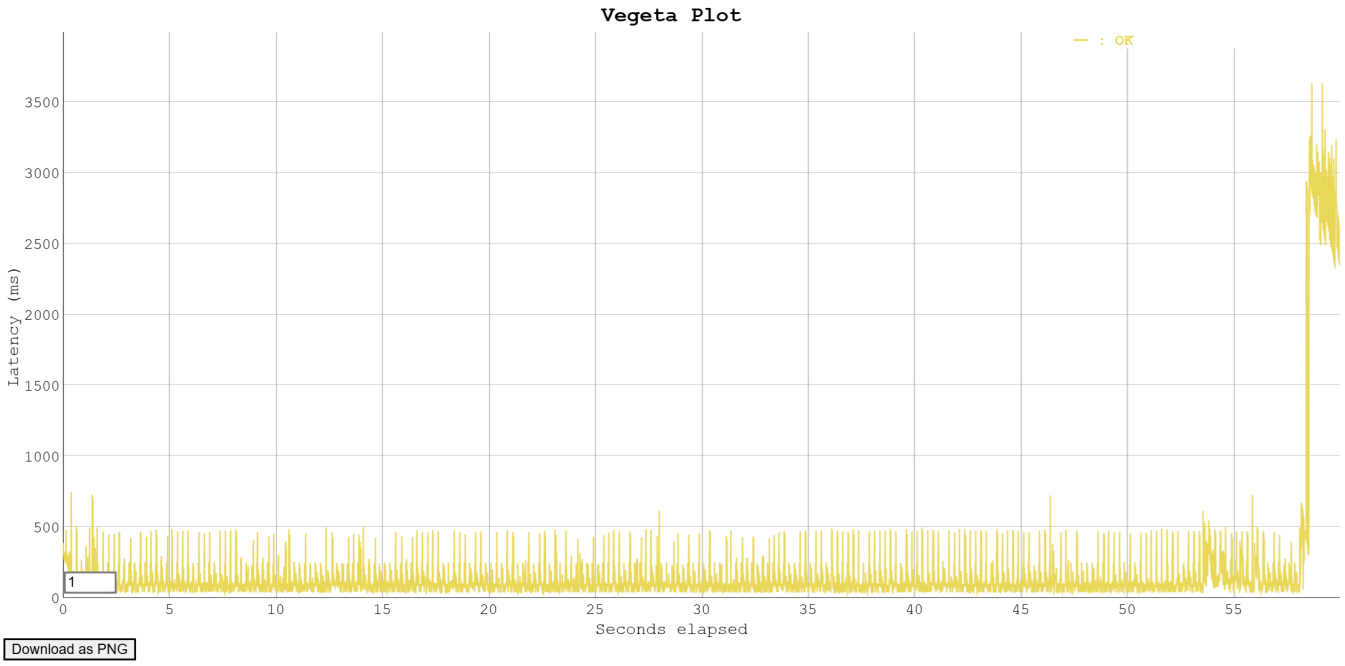
```
Status Codes  [code:count]                  200:6000
Error Set:
```



The graph indicates the frontend service functioned under load as we can for the 10 first seconds the latency is high around 4500ms and after that it have stabilize around 200 and 300 ms. It is the demonstration of the auto-scaling on the cluster.

rate 250

```
echo "GET http://34.78.207.78/all" |
vegeta attack -duration=60s -rate=250 | tee results2.bin | vegeta report
Requests      [total, rate, throughput]         15000, 250.02, 237.88
Duration      [total, attack, wait]             1m3s, 59.995s, 3.062s
Latencies     [min, mean, 50, 90, 95, 99, max]  36.224ms, 180.703ms, 90.217ms,
238.871ms, 451.805ms, 2.886s, 3.634s
Bytes In      [total, mean]                     9465000, 631.00
Bytes Out     [total, mean]                     0, 0.00
Success       [ratio]                           100.00%
Status Codes  [code:count]                      200:15000
Error Set:
```



The small increase in latency at the start suggests that the autoscaler scaled the pods quickly enough to handle the load, which stabilised for more than 50 seconds. However, in the last 10 seconds, it is possible that the existing pods were overloaded due to the influx of requests, leading to resource exhaustion. This could explain the significant increase in latency as pods struggled to respond to requests.

In conclusion using an autoscaler to dynamically adjust the number of replicas for an application is generally more advantageous than manually setting a fixed number.