*In this lab, we will use the solution folder of the previous lab. Download it from the Moodle.*

**Deploy microase with K8s**

1. Configure a minikube cluster
2. Launch[[1]](#footnote-1) minikube start --nodes 3
3. Open a new terminal and launch minikube tunnel

*Browse the docs and explain what you achieved with the above commands.*

|  |
| --- |
| **Output:**  ✅ Tunnel successfully started  📌 NOTE: Please do not close this terminal as this process must stay alive for the tunnel to be accessible ...  **Explanation:**  minikube tunnel runs as a process, creating a network route on the host to the service CIDR (Classless Inter-Domain Routing is a method for allocating IP addresses and for IP routing) of the cluster using the cluster’s IP address as a gateway. The tunnel command exposes the external IP directly to any program running on the host operating system.  <https://minikube.sigs.k8s.io/docs/handbook/accessing/>  and after creating service.yml we can see  🏃 Starting tunnel for service gateway.  So only our gateway service will be able to talk to external world because while defining gateway service we select its type as  type: LoadBalancer |

1. Write a deployment.yaml file and a service.yaml file to realise the orchestration sketched below. Note that each microservice is associated to a Deployment and that math-service and string-service have two replicas each. Each Deployment needs to be associated to a Service with a suitable [type](https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types).

Service

Service

Service

Deployment

Deployment

Deployment

G

Pod

Pod

Pod

Pod

Pod

gateway

string-service

math-service

Note that images of each service are available on DockerHub as:

* jacopomassa97/math\_py:latest
* jacopomassa97/string\_rust:latest
* jacopomassa97/gateway:latest

1. Launch the orchestration through commands:

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

**observations:**

PS C:\Users\hassa\Downloads\Compressed\crash\_test>

kubectl get pods

NAME READY STATUS RESTARTS AGE

gateway-7c79b75694-nv48v 1/1 Running 0 4m7s

math-service-7cb7f756fd-cr5pk 1/1 Running 0 4m7s

math-service-7cb7f756fd-q8x59 1/1 Running 0 4m7s

string-service-7dc4778dfb-kpc8x 1/1 Running 0 4m7s

string-service-7dc4778dfb-wjq7c 1/1 Running 0 4m7s

*we can see as per our deployment configuration we have 2 pods for maths and string service and only 1 pod for gateway service.*

kubectl get nodes

NAME STATUS ROLES AGE VERSION

minikube Ready control-plane 57m v1.25.2

minikube-m02 Ready <none> 57m v1.25.2

minikube-m03 Ready <none> 56m v1.25.2

*as per our initial configuration we have stared our kubectl with 3 nodes out of them 1 node is serving as control-plan.*

PS C:\Users\hassa\Downloads\Compressed\crash\_test>

kubectl get svc

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

gateway LoadBalancer 10.103.97.223 127.0.0.1 5000:32247/TCP 7m40s

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 61m

math-service ClusterIP 10.96.108.125 <none> 5000/TCP 7m40s

string-service ClusterIP 10.107.229.100 <none> 5000/TCP 7m40s

*from here we can see that only service with the name gateway is accessible via localhost and 32247 port which will be mapped to container port 5000*

1. Query the service interface from a browser or via curl.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

*our application is working and only accessible via gaetway service we can not directly access math and string service from local host*

lets try some more observations

PS C:\Users\hassa\Downloads\Compressed\crash\_test>

kubectl get pods -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

gateway-7c79b75694-nv48v 1/1 Running 0 22m 10.244.1.3 minikube-m02 <none> <none>

math-service-7cb7f756fd-cr5pk 1/1 Running 0 22m 10.244.2.2 minikube-m03 <none> <none>

math-service-7cb7f756fd-q8x59 1/1 Running 0 22m 10.244.1.2 minikube-m02 <none> <none>

string-service-7dc4778dfb-kpc8x 1/1 Running 0 22m 10.244.1.4 minikube-m02 <none> <none>

string-service-7dc4778dfb-wjq7c 1/1 Running 0 22m 10.244.2.3 minikube-m03 <none> <none>

*here we can see that each pod has assiagnd ip addresses lets try to delete pod*

*kubectl delete pod math-service-7cb7f756fd-cr5pk*

*pod "math-service-7cb7f756fd-cr5pk" deleted*

*PS C:\Users\hassa\Downloads\Compressed\crash\_test> kubectl get pods -o wide*

*NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES*

*gateway-7c79b75694-nv48v 1/1 Running 0 27m 10.244.1.3 minikube-m02 <none> <none>*

*math-service-7cb7f756fd-q8x59 1/1 Running 0 27m 10.244.1.2 minikube-m02 <none> <none>*

*math-service-7cb7f756fd-rf7rb 1/1 Running 0 58s 10.244.2.4 minikube-m03 <none> <none>*

*string-service-7dc4778dfb-kpc8x 1/1 Running 0 27m 10.244.1.4 minikube-m02 <none> <none>*

*string-service-7dc4778dfb-wjq7c 1/1 Running 0 27m 10.244.2.3 minikube-m03 <none> <none>*

*as I tried to delete one pod which is running math service, but we can see that even if it got deleted k8s is trying to maintain copies of this pod as per our manifest. But pod is assigned with new Ip address as per k8s specifications.*

1. To stop the orchestration issue:

minikube delete --all

Deletes a local Kubernetes cluster. This command deletes the VM, and removes all associated files.

minikube stop --all

Stops a running local Kubernetes cluster

Graphical user interface, application

Description automatically generated

When we restarted cluster we observed that each pod got new Ip address and cluster Ip address are same

PS C:\Users\hassa\Downloads\Compressed\crash\_test>

kubectl get pods -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

gateway-7c79b75694-bxfns 1/1 Running 0 50s 10.244.1.2 minikube-m02 <none> <none>

math-service-7cb7f756fd-g2gfl 1/1 Running 0 50s 10.244.1.4 minikube-m02 <none> <none>

math-service-7cb7f756fd-rrk82 1/1 Running 0 70s 10.244.0.4 minikube <none> <none>

string-service-7dc4778dfb-6kmvz 1/1 Running 0 50s 10.244.1.3 minikube-m02 <nminikube-m02 minikube-m02 Ready minikube-m02 Ready <none> 2m2s v1.25.2

minikube-m03 Ready <none> 102s v1.25.2

**Deploy microase with Docker Swarm**

1. Initialise the Docker Swarm mode via

docker swarm init --advertise-addr 127.0.0.1:5001

1. Write a suitable stack.yaml file to orchestrate the application with the same replicas as above. **Hint:** you should adapt your Docker Compose file by exploiting the images available on DockerHub for each service.
2. Launch your orchestration via suitable [docker stack](https://docs.docker.com/engine/reference/commandline/stack/) commands.

**Bonus**

We have added endpoints to make our services crash:

* /crash, to make the gateway crash
* /crashall, to make all services crash
* /math/crash, to make the math-service crash
* /string/crash, to make the string-service crash

*Try to query them both in the K8s and the Docker Swarm orchestrations. Do the two deployments behave differently? How?*

|  |
| --- |
| Crash observation with k8s:  First we tried to crash math and string service but we did not faced any service down time as after crash command we tried to run math functions it worked and same for string services because k8s we have 2 replicas for these services.  Second we tried to crash gateway and as the result we lost connection to our math and string services for some time and then they start working this is because we have 1 replica for gateway and if it is down k8s will try to maintain number of replicas which is 1 in gateway case so services will be inaccessible until gateway pod will be created and start running our container  Third we tried crashall services and k8s nicely managed that crash automatically but we faced some service downtime. |

1. For Windows users, add the option flag -- driver=docker [↑](#footnote-ref-1)