

ASSIGNMENT 10 – RISULTATI AGGIORNATI

MAIN TOPICS: KUBERNETES, Helm, Istio, Prometheus, Grafana

MODULE 1: HELM

I've created a virtual machine called LAB12vm in order to perform the overall activity without interfering with other projects.

After having installed all the packages:

```
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-4 |  
VERIFY_CHECKSUM=false bash
```

```
helm version
```

```
helm install ciao-app helm/ --set customText="ciao!" -n ciao-app --create-  
namespace
```

If there's an **error** in the installation I have to run the following commands:

```
sudo k0s kubeconfig admin > ~/.kube/config HERE WE EXTRACT THE ADMINISTRATIVE  
CREDENTIALS GENERATED BY K0S TO SAVE THEM IN A CONGIF FILE IN THE HIDDEN DIRECTORY .KUBE (IN THE  
HOME DIRECTORY) IN ORDER TO LET KUBECTL AND HELM TALK TO THE CLUSTER
```

```
chmod 600 ~/.kube/config THIS MAKES THE PERMISSION SO THAT MY USER CAN READ OR WRITE THAT  
FILE.
```

At first I create a RELEASE based on a CHART with the parametrization (“—set ..”). This, by overwriting in the values.yaml default file with a personalized text without modifying the code of the Chart manually.

Creation of the namespace (“-n” ecc...) to isolate the resources and avoid conflicts.

Then, verification of the status and pods.

With “jsonpath...” I extract the IP address inside the pod directly from the API of Kubernetes (advanced way to filter the JSON data); the “curl” after demonstrates that the ConfigMap generated by Helm has injected the personalized text correctly in the web server Nginx.

```
ubuntu@LAB12vm:~/advanced-kubernetes$ cd ~/advanced-kubernetes  
ubuntu@LAB12vm:~/advanced-kubernetes$ helm install ciao-app ./helm --set customText="ciao!" -n ciao-app --create-namespace  
NAME: ciao-app  
LAST DEPLOYED: Tue Jan 27 11:26:39 2026  
NAMESPACE: ciao-app  
STATUS: deployed  
REVISION: 1  
TEST SUITE: None  
ubuntu@LAB12vm:~/advanced-kubernetes$ helm install hello-app helm/ --set customText="hello!" -n hello-app --create-namespace  
NAME: hello-app  
LAST DEPLOYED: Tue Jan 27 11:27:57 2026  
NAMESPACE: hello-app  
STATUS: deployed  
REVISION: 1  
TEST SUITE: None  
ubuntu@LAB12vm:~/advanced-kubernetes$ kubectl get pods -n ciao-app  
kubectl get pods -n hello-app  
helm list -n ciao-app  
helm list -n hello-app  
NAME READY STATUS RESTARTS AGE  
ciao-app-854f56985c-f8ng 1/1 Running 0 85s  
NAME READY STATUS RESTARTS AGE  
hello-app-5ccf5d479-m9rqm 1/1 Running 0 7s  
NAME NAMESPACE REVISION UPDATED STATUS CHART APP VERSION  
ciao-app ciao-app 1 2026-01-27 11:26:39.862817934 +0100 CET deployed simple-nginx-0.1.0 1.0  
NAME NAMESPACE REVISION UPDATED STATUS CHART APP VERSION  
hello-app hello-app 1 2026-01-27 11:27:57.931173631 +0100 CET deployed simple-nginx-0.1.0 1.0  
ubuntu@LAB12vm:~/advanced-kubernetes$ ciaoIP=$(kubectl get po -n ciao-app -o jsonpath=".items[*].status.podIP")  
curl $ciaoIP  
  
helloIP=$(kubectl get po -n hello-app -o jsonpath=".items[*].status.podIP")  
curl $helloIP  
<html>  
  <head>  
    <title>Simple Nginx Webapp</title>  
  </head>  
  <body>  
    <h1>ciao!</h1>  
  </body>  
</html>  
[<html>  
  <head>  
    <title>Simple Nginx Webapp</title>  
  </head>  
  <body>  
    <h1>hello!</h1>  
  </body>  
</html>  
ubuntu@LAB12vm:~/advanced-kubernetes$
```

To perform the Bonus exercise in Module 1 on Helm, I enter the file *helm/values.yaml* to include a new line *customPage*. And then I connect the variable to the HTML template of ConfigMap.

The image shows two terminal windows side-by-side. The left window displays the contents of a file named 'values.yaml' with the following content:

```

replicaCount: 1
image:
  repository: nginx
  tag: latest
  pullPolicy: IfNotPresent
service:
  type: ClusterIP
  port: 80
customText: "Welcome to the Advanced Kubernetes Class!"
customPage: "I've added this line!"
```

The right window displays the contents of a file named 'configmap.yaml' with the following content:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ include "simple-nginx.fullname" . }}-html
  labels:
    app: {{ include "simple-nginx.name" . }}
data:
  index.html: |
    <html>
      <head>
        <title>{{ .Values.customPage }}</title>
      </head>
      <body>
        <h1>{{ .Values.customText }}</h1>
      </body>
    </html>
```

And finally I have installed the Goodbye application:

The image shows a terminal window with the following command history:

```

ubuntu@LAB12vm:~/advanced-kubernetes$ cd ..
ubuntu@LAB12vm:~/advanced-kubernetes$ nano helm/templates/configmap.yaml
ubuntu@LAB12vm:~/advanced-kubernetes$ helm install goodbye-app ./helm --set customPage="Goodbye!" --set customText="goodbye!" -n goodbye-app --create-namespace
NAME: goodbye-app
LAST DEPLOYED: Tue Jan 27 11:47:39 2026
NAMESPACE: goodbye-app
STATUS: deployed
REVISION: 1
TEST SUITE: None
ubuntu@LAB12vm:~/advanced-kubernetes$ goodbyeIP=$(kubectl get po -n goodbye-app -o jsonpath=".items[0].status.podIP")
curl $goodbyeIP
<html>
  <head>
    <title>Goodbye!</title>
  </head>
  <body>
    <h1>goodbye!</h1>
  </body>
</html>
```

Then I have uninstalled the packages as suggested by the Exercise text.

```

helm uninstall -n ciao-app ciao-app
helm uninstall -n hello-app hello-app
helm uninstall -n goodbye-app goodbye-app
kubectl delete ns ciao-app
kubectl delete ns hello-app
kubectl delete ns goodbye-app
```

MODULE 2: ISTIO

I have installed Istio, Prometheus and Grafana as requested by the exercise.

```
curl -L https://istio.io/downloadIstio | ISTIO_VERSION=1.28.0 sh - # Downloads and extracts the specified Istio version.
```

```
export PATH="$PWD/istio-1.28.0/bin:$PATH" # Adds Istio binaries to your PATH for easy access.
```

```
istioctl install --set profile=minimal -y # Installs Istio with the minimal profile, suitable for labs and quick demos.
```

```
kubectl get pods -n istio-system
```

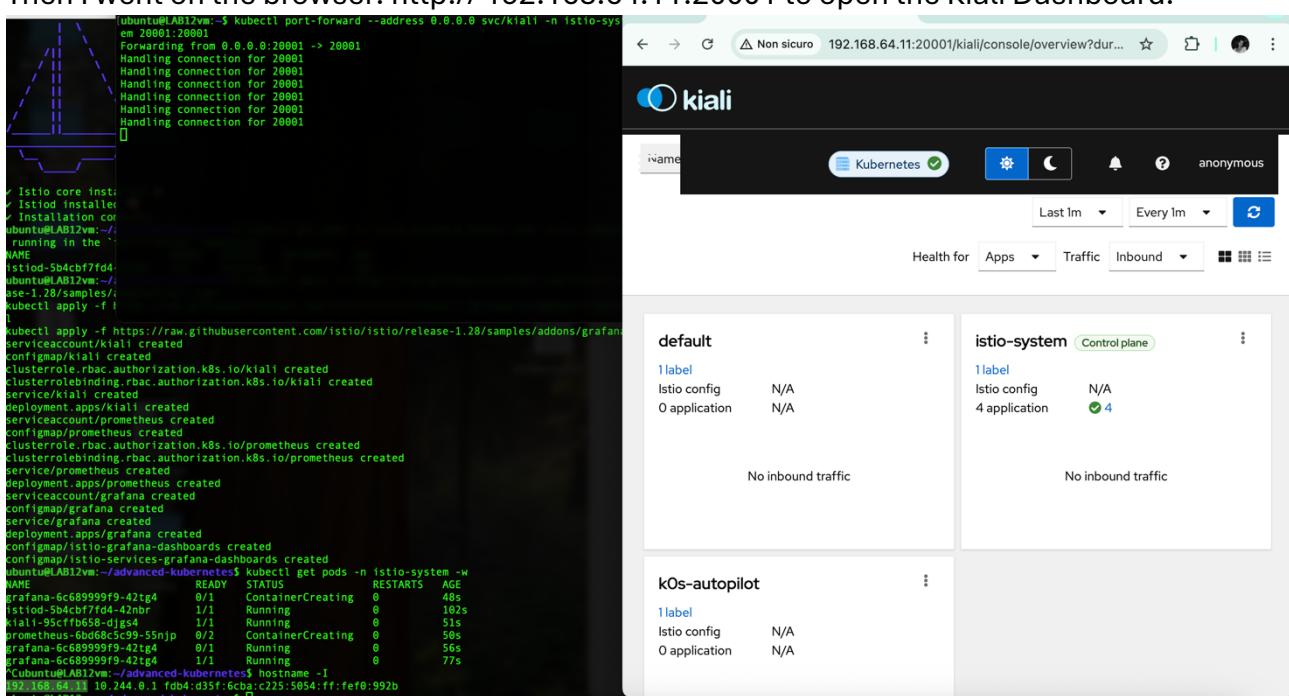
```
kubectl apply -f https://raw.githubusercontent.com/istio/istio/release-1.28/samples/addons/kiali.yaml  
kubectl apply -f https://raw.githubusercontent.com/istio/istio/release-1.28/samples/addons/prometheus.yaml  
kubectl apply -f https://raw.githubusercontent.com/istio/istio/release-1.28/samples/addons/grafana.yaml
```

```
configmap/istio-services.grafana dashboards created  
ubuntu@LAB12vm:~/advanced-kubernetes$ kubectl get pods -n istio-system -w  
NAME READY STATUS RESTARTS AGE  
grafana-6c68999f9-42tg4 0/1 ContainerCreating 0 48s  
istiod-5b4cbf7fd4-42nbr 1/1 Running 0 102s  
kiali-95cffb658-djgs4 1/1 Running 0 51s  
prometheus-6bd68c5c99-55njp 0/2 ContainerCreating 0 50s  
grafana-6c68999f9-42tg4 0/1 Running 0 56s  
grafana-6c68999f9-42tg4 1/1 Running 0 77s  
^Cubuntu@LAB12vm:~/advanced-kubernetes$
```

I have checked which was my IP address for my VM with :

```
hostname -I
```

Discovered that it was 192.168.64.11. So I opened another terminal (up on the left in the photo) with that command that temporarily forwards the Kiali service to my machine. Then I went on the browser: http:// 192.168.64.11:20001 to open the Kiali Dashboard.



Then I have created the demo namespace with the helloworld sample app in it. The sample app (helloworld.yaml) is a simple HTTP service used for demonstration and testing Istio features.

```
ubuntu@LAB1vm:~/advanced-kubernetes$ kubectl get pods -n istio-system -w
NAME                               READY   STATUS    RESTARTS   AGE
grafana-6c689999f9-42tg4          0/1     ContainerCreating   0          48s
istio-id-5b4cf7fd4-42nbr          0/1     Running   0          102s
kiali-95cff658-djgs4              0/1     Running   0          51s
prometheus-5bd6ac5c99-55njp      0/2     ContainerCreating   0          50s
grafana-6c689999f9-42tg4          0/1     Running   0          56s
grafana-6c689999f9-42tg4          1/1     Running   0          77s
^Cubuntu@LAB1vm:~/advanced-kubernetes$ hostname -I
192.168.64.11 10.244.8.1 fdb4:d35f:6cha:c225:5954:ff:fe0:992b
ubuntu@LAB1vm:~/advanced-kubernetes$ kubectl create namespace demo
kubectl label namespace demo istio-injection=enabled
namespace/demo created
namespace/demo labeled
ubuntu@LAB1vm:~/advanced-kubernetes$ kubectl apply -n demo -f https://raw.githubusercontent.com/istio/release-1.28/samples/helloworld/helloworld.yaml
kubectl get pods -n demo -w
service/helloworld created
deployment.apps/helloworld-v1 created
deployment.apps/helloworld-v2 created
NAME                               READY   STATUS    RESTARTS   AGE
helloworld-v2-86b89467fc-7zbk5    0/2     Pending   0          0s
helloworld-v1-7c56bd705-rpk5b     0/2     Pending   0          0s
helloworld-v2-86b89467fc-7zbk5    0/2     Pending   0          0s
helloworld-v1-7c56bd705-rpk5b     0/2     Pending   0          0s
helloworld-v2-86b89467fc-7zbk5    0/2     Pending   0          0s
helloworld-v1-7c56bd705-rpk5b     0/2     Init:0/2  0          0s
helloworld-v2-86b89467fc-7zbk5    0/2     Init:0/2  0          0s
helloworld-v1-7c56bd705-rpk5b     0/2     Init:0/2  0          0s
helloworld-v2-86b89467fc-7zbk5    0/2     Init:0/2  0          27s
helloworld-v2-86b89467fc-7zbk5    0/2     Init:0/2  0          29s
helloworld-v1-7c56bd705-rpk5b     0/2     Init:0/2  0          34s
helloworld-v2-86b89467fc-7zbk5    0/2     Init:0/2  0          36s
helloworld-v1-7c56bd705-rpk5b     0/2     Init:0/2  0          37s
helloworld-v2-86b89467fc-7zbk5    0/2     PodInitializing 0          39s
helloworld-v1-7c56bd705-rpk5b     0/2     PodInitializing 0          48s
helloworld-v2-86b89467fc-7zbk5    0/2     PodInitializing 0          41s
helloworld-v1-7c56bd705-rpk5b     0/2     PodInitializing 0          46s
helloworld-v2-86b89467fc-7zbk5    2/2     Running   0          72s
helloworld-v2-86b89467fc-7zbk5    2/2     Running   0          77s
ubuntu@LAB1vm:~/advanced-kubernetes$
```

Namespace	Service	Labels	Applications	Inbound Traffic
default	demo	Istio config, N/A	1 application	No inbound traffic
demo	helloworld	Istio config, N/A	4 applications	No inbound traffic

Each pod should have two containers: the application and the istio-proxy. The Istio sidecar (Envoy) has been injected automatically! Istio uses automatic sidecar injection to add an Envoy proxy container to each pod in a labeled namespace. This enables traffic management, security, and observability features for all services in that namespace.

```
kubectl describe pod -n demo | grep -E "container.*istio-proxy"
```

I am using this command to validate the automatic sidecar injection. Since I labeled the namespace with `istio-injection=enabled`, Istio intercepts the Pod creation and adds an Envoy proxy (`istio-proxy`). Without this container, the Pod would not be part of the Service Mesh, and I would be unable to monitor traffic or apply security policies like mTLS.

SO: Istio has injected automatically a second container (Proxy Envoy) inside the same pod.

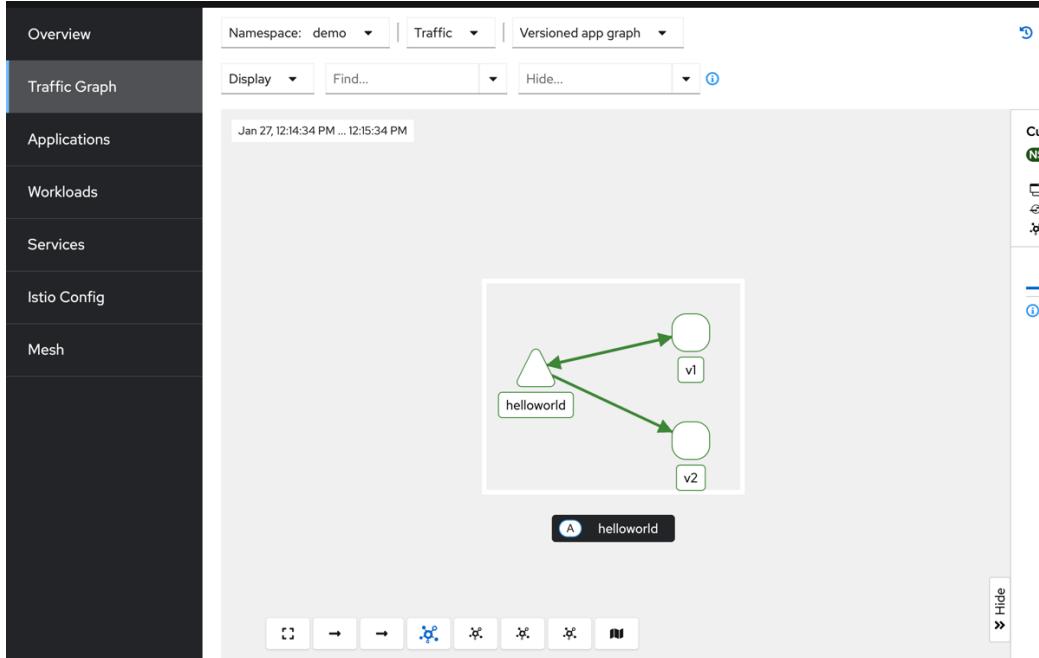
```
helloworld-v2-86b89467fc-7zbk5 2/2     Running   0          77s
^Cubuntu@LAB1vm:~/advanced-kubernetes$ kubectl describe pod -n demo | grep -E "container.*istio-proxy"
  Normal  Created  4m53s  kubelet           spec.initContainers(istio-proxy): Created container: istio-proxy
  Normal  Started  4m49s  kubelet           spec.initContainers(istio-proxy): Started container istio-proxy
  Normal  Created  4m52s  kubelet           spec.initContainers(istio-proxy): Created container: istio-proxy
  Normal  Started  4m49s  kubelet           spec.initContainers(istio-proxy): Started container istio-proxy
ubuntu@LAB1vm:~/advanced-kubernetes$
```

Now in the Kiali dashboard we can see in Applications the helloworld correctly created.

App Name	Name	Namespace	Labels	Details
helloworld		NS demo		app=helloworld service=helloworld version=v1/v2

And if I generate traffic with `istio/helloworld-traffic.sh`

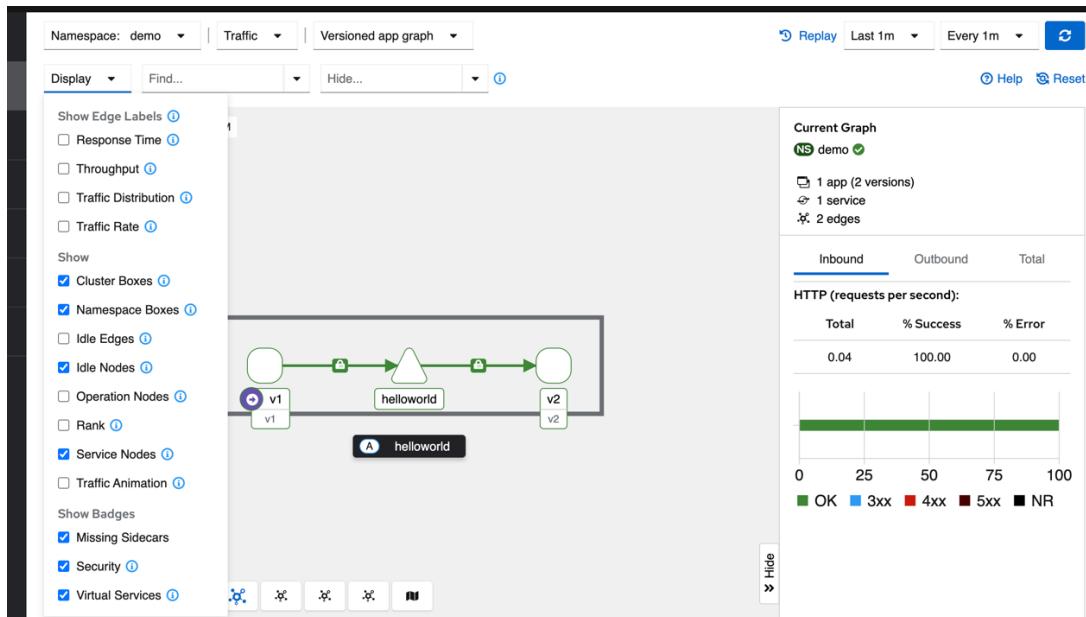
I can see from Kiali that the traffic graph has appeared and correctly working with green



links.

Then, if we enable the security mTLS given by Istio with **kubectl apply -f istio/mtls.yaml** And reload the application and the traffic with:

**kubectl rollout restart deployment -n demo helloworld-v1 helloworld-v2
istio/helloworld-traffic.sh**



The PeerAuthentication policy configures Istio to require mTLS for all workloads in the namespace.

Now let's try to elaborate complex application traces:

(form the text...)

The travel demo application consists of several microservices deployed across three namespaces. Label each namespace to enable automatic Envoy sidecar injection for observability and traffic management.

```
kubectl create namespace travel-agency
```

```
kubectl create namespace travel-portal
```

```
kubectl create namespace travel-control
```

```
kubectl label namespace travel-agency istio-injection=enabled
```

```
kubectl label namespace travel-portal istio-injection=enabled
```

```
kubectl label namespace travel-control istio-injection=enabled
```

Apply the manifests for each part of the application. These will create multiple interconnected services for a realistic microservices scenario.

```
kubectl apply -f <(curl -L  
https://raw.githubusercontent.com/kiali/demos/master/travels/travel_agency.yaml)  
-n travel-agency  
kubectl apply -f <(curl -L  
https://raw.githubusercontent.com/kiali/demos/master/travels/travel_portal.yaml)  
-n travel-portal  
kubectl apply -f <(curl -L  
https://raw.githubusercontent.com/kiali/demos/master/travels/travel_control.yaml  
) -n travel-control
```

Namespace: 3 | Traffic | Workload graph | Replay Last 1m Every 1m

Display Find... Hide... Help Reset

Jan 27, 01:04:07 PM ... 01:05:07 PM

Current Graph

- travel-portal
- istio-system
- travel-agency

15 services | 14 workloads

Inbound | Outbound | Total

No inbound traffic.

Grid - non-boxing layout

Pod	Image	Replicas	Status	Last Seen	Age
travel-agency	flights-v1-8dc97889b-zbcd5	1/2	CrashLoopBackOff	5 (18s ago)	28m
travel-agency	mysqlDb-v1-55c7575545-dbsld	2/2	Running	6 (2m51s ago)	30m
travel-agency	mysqlDb-v1-55c7575545-dbsld	1/2	Error	6 (2m54s ago)	31m
travel-agency	flights-v1-8dc97889b-zbcd5	1/2	Error	6 (2m48s ago)	31m
travel-agency	mysqlDb-v1-55c7575545-dbsld	1/2	CrashLoopBackOff	6 (15s ago)	31m
travel-agency	flights-v1-8dc97889b-zbcd5	1/2	CrashLoopBackOff	6 (13s ago)	31m
travel-agency	cars-v1-55df97679d-hm2rx	1/2	Error	7 (5m17s ago)	32m
travel-agency	hotels-v1-686f9d5986-h2vxx	1/2	Error	7 (5m2s ago)	32m
travel-agency	hotels-v1-686f9d5986-h2vxx	1/2	CrashLoopBackOff	7 (22s ago)	32m
travel-agency	cars-v1-55df97679d-hm2rx	1/2	CrashLoopBackOff	7 (55s ago)	33m
travel-agency	travels-v1-784fb9596b-sgdbf	1/2	CrashLoopBackOff	7 (43s ago)	33m
travel-portal	voyages-6458cdb7cf-hcwgl	2/2	Running	7 (5m39s ago)	32m
travel-portal	viaggi-875847696-htpk6	1/2	CrashLoopBackOff	7 (41s ago)	32m
travel-agency	discounts-v1-79965fc986-ljc5v	2/2	Running	7 (5m48s ago)	33m
travel-portal	travels-645cb75748-7d8mb	1/2	Error	7	32m
travel-agency	discounts-v1-79965fc986-ljc5v	1/2	CrashLoopBackOff	7 (29s ago)	33m
travel-agency	insurances-v1-9d9f49d65-xrkgd	2/2	Running	7 (5m22s ago)	33m
travel-portal	travels-645cb75748-7d8mb	1/2	CrashLoopBackOff	7 (52s ago)	32m
travel-portal	voyages-6458cdb7cf-hcwgl	1/2	CrashLoopBackOff	7 (36s ago)	32m
travel-control	control-69c9b9869d-s84ts	1/2	CrashLoopBackOff	7 (37s ago)	32m
travel-agency	insurances-v1-9d9f49d65-xrkgd	1/2	Error	7 (5m32s ago)	33m
travel-agency	insurances-v1-9d9f49d65-xrkgd	1/2	CrashLoopBackOff	7 (18s ago)	33m
travel-agency	mysqlDb-v1-55c7575545-dbsld	1/2	Error	7 (5m8s ago)	36m
travel-agency	mysqlDb-v1-55c7575545-dbsld	1/2	CrashLoopBackOff	7 (13s ago)	36m
travel-agency	flights-v1-8dc97889b-zbcd5	1/2	Error	7 (5m12s ago)	36m
travel-agency	flights-v1-8dc97889b-zbcd5	1/2	CrashLoopBackOff	7 (12s ago)	36m
travel-agency	cars-v1-55df97679d-hm2rx	1/2	Error	8 (5m2s ago)	37m
travel-agency	hotels-v1-686f9d5986-h2vxx	1/2	Error	8 (5m2s ago)	37m
travel-agency	hotels-v1-686f9d5986-h2vxx	1/2	CrashLoopBackOff	8 (15s ago)	37m
travel-agency	cars-v1-55df97679d-hm2rx	1/2	CrashLoopBackOff	8 (16s ago)	37m
travel-agency	travels-v1-784fb9596b-sgdbf	1/2	Error	8 (5m11s ago)	37m
travel-portal	travels-645cb75748-7d8mb	1/2	Error	8 (5m11s ago)	36m
travel-agency	travels-v1-784fb9596b-sgdbf	1/2	CrashLoopBackOff	8 (15s ago)	37m
travel-portal	viaggi-875847696-htpk6	1/2	Error	8 (5m14s ago)	37m
travel-portal	travels-645cb75748-7d8mb	1/2	CrashLoopBackOff	8 (16s ago)	37m
travel-portal	voyages-6458cdb7cf-hcwgl	2/2	Running	8 (5m10s ago)	37m
travel-portal	voyages-6458cdb7cf-hcwgl	1/2	Error	8 (5m16s ago)	37m
travel-portal	viaggi-875847696-htpk6	1/2	CrashLoopBackOff	8 (21s ago)	37m
travel-agency	discounts-v1-79965fc986-ljc5v	2/2	Running	8 (5m19s ago)	38m
travel-control	control-69c9b9869d-s84ts	2/2	Running	8 (5m17s ago)	36m
travel-control	control-69c9b9869d-s84ts	1/2	Error	8 (5m19s ago)	36m
travel-agency	discounts-v1-79965fc986-ljc5v	1/2	Error	8 (5m22s ago)	38m
travel-portal	voyages-6458cdb7cf-hcwgl	1/2	CrashLoopBackOff	8 (17s ago)	37m
travel-agency	insurances-v1-9d9f49d65-xrkgd	1/2	Error	8 (5m8s ago)	38m
travel-agency	discounts-v1-79965fc986-ljc5v	1/2	CrashLoopBackOff	8 (16s ago)	38m
travel-control	control-69c9b9869d-s84ts	1/2	CrashLoopBackOff	8 (19s ago)	37m
travel-agency	insurances-v1-9d9f49d65-xrkgd	1/2	CrashLoopBackOff	8 (17s ago)	38m
travel-agency	mysqlDb-v1-55c7575545-dbsld	1/2	Error	8 (5m15s ago)	41m
travel-agency	flights-v1-8dc97889b-zbcd5	1/2	Error	8 (5m9s ago)	41m
travel-agency	mysqlDb-v1-55c7575545-dbsld	1/2	CrashLoopBackOff	8 (14s ago)	41m
travel-agency	flights-v1-8dc97889b-zbcd5	1/2	CrashLoopBackOff	8 (14s ago)	41m

I had to reload the overall VM because the quantity of logs to run is very high so it gives me the CrashLoopBackOff error, signi of very small memory in my VM so that the processing is very difficult to perform for my PC.

MODULE 3: TELEMETRY

I have installed Grafana and Prometheus using Helm as the guide suggested. Prometheus was successfully installed but Grafana gave me a CrashLoopBack and, by checking the logs with:

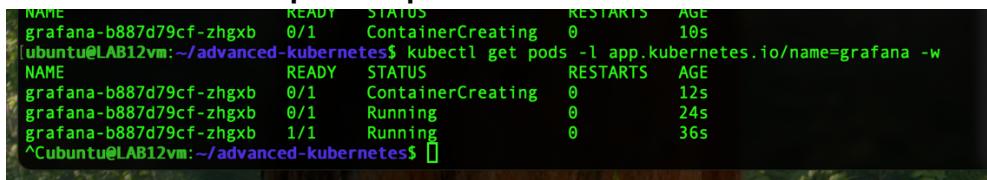
```
kubectl logs grafana-7f4875785c-z7pkr
```

```
kubectl describe pod grafana-7f4875785c-z7pkr
```

I performed a clean reinstallation because the Exit Code 139 suggested a segmentation fault or binary incompatibility. By redeploying the chart and disabling persistence, I ruled out potential conflicts with the VM's file system or corrupted configuration states.

The latest Grafana image (12.3.1) was triggering a Segmentation Fault (Exit Code 139) due to an instruction set incompatibility within the emulated ARM64 environment of this VM. I resolved this by performing a downgrade to version 10.4.0 and explicitly defining Resource Requests, ensuring the container starts with a stable binary and guaranteed memory allocation.

```
helm install grafana grafana/grafana \
--set image.tag=10.4.0 \
--set adminPassword=admin \
--set service.type=NodePort \
--set service.nodePort=30405 \
--set persistence.enabled=false \
--set resources.requests.memory=256Mi \
--set resources.requests.cpu=100m
```



A terminal window showing the output of the command `kubectl get pods -l app.kubernetes.io/name=grafana -w`. The output lists three pods: `grafana-b887d79cf-zhgb`, `grafana-b887d79cf-zhgb`, and `grafana-b887d79cf-zhgb`. The first two are in a `ContainerCreating` state, while the third is `Running`.

NAME	READY	STATUS	RESTARTS	AGE
grafana-b887d79cf-zhgb	0/1	ContainerCreating	0	10s
grafana-b887d79cf-zhgb	0/1	ContainerCreating	0	12s
grafana-b887d79cf-zhgb	0/1	Running	0	24s
grafana-b887d79cf-zhgb	1/1	Running	0	36s

In order to retrieve how to access to Prometheus and Grafana I have applied the following commands suggested by the guide:

```
export PNODE_PORT=$(kubectl get --namespace default -o
jsonpath=".spec.ports[0].nodePort" services prometheus-server)
export PNODE_IP=$(kubectl get nodes --namespace default -o
jsonpath=".items[0].status.addresses[0].address")
echo "Prometheus UI: http://$PNODE_IP:$PNODE_PORT"

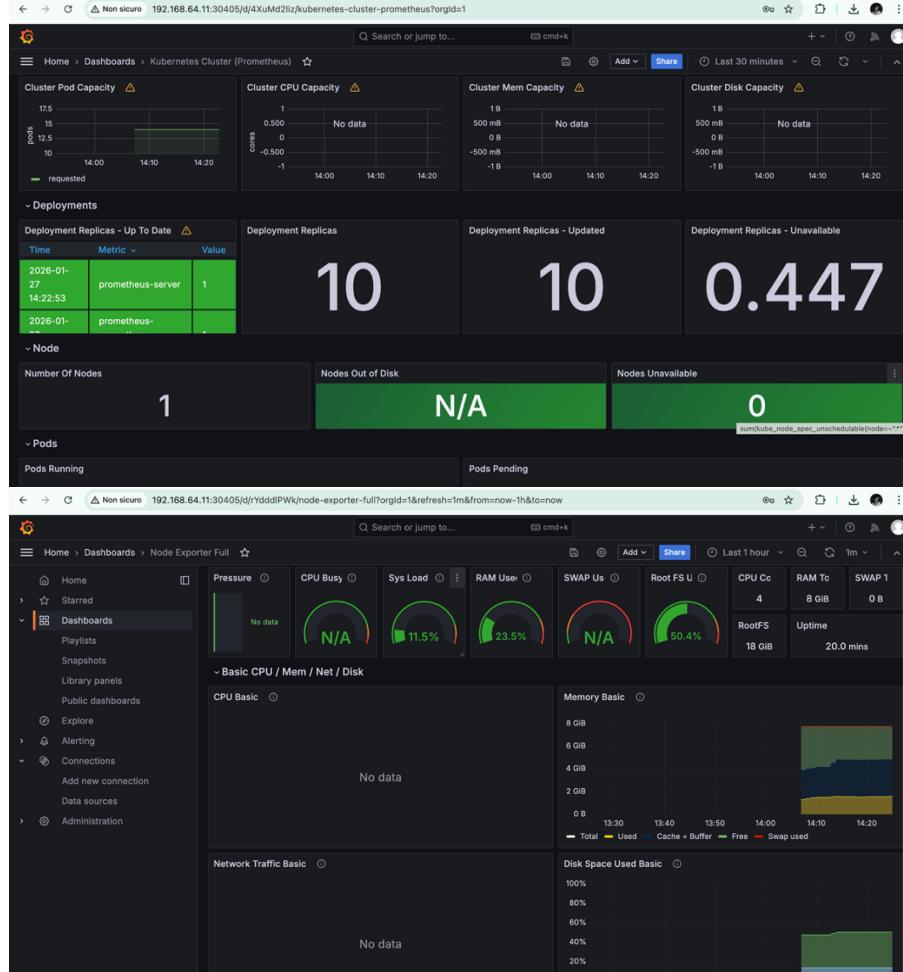
# Get the Grafana URL to visit by running these commands in the same shell:
export GNODE_PORT=$(kubectl get --namespace default -o
jsonpath=".spec.ports[0].nodePort" services grafana)
export GNODE_IP=$(kubectl get nodes --namespace default -o
jsonpath=".items[0].status.addresses[0].address")
echo "Grafana UI: http://$GNODE_IP:$GNODE_PORT"
```

Prometheus UI: <http://192.168.64.11:30303>

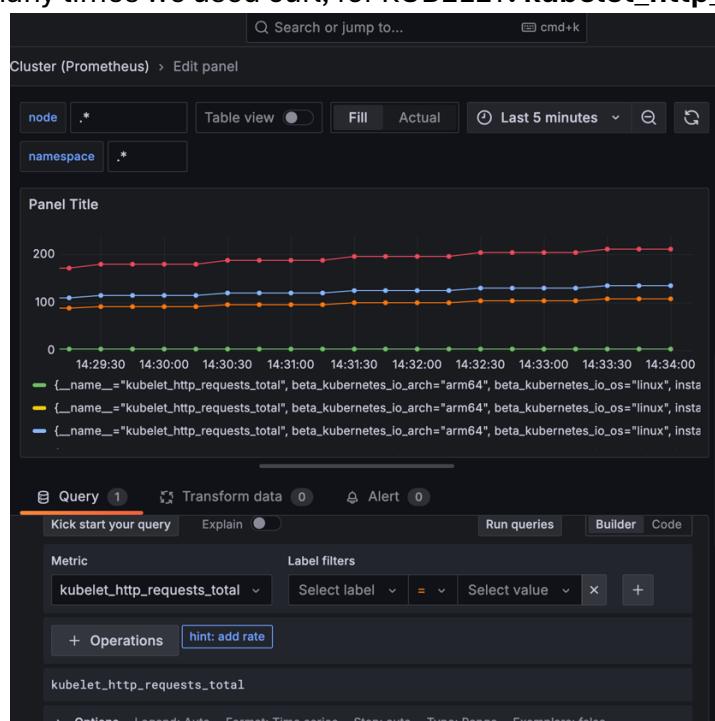
Grafana UI: <http://192.168.64.11:30405>

So I open Grafana with that link and add Prometheus as a Data Source in Connection>data sources>Add data source and by pasting the previous link given by Prometheus.

Then, I have added with their JSON file the two dashboards: the Cluster monitoring and the Node Exporter Full.



Then, if we go to Add (in alto a destra) and Visualization, we can add some metrics. For example how many times we used curl, for KUBELET: `kubelet_http_requests_total`.



We can create an Alert rule (in the left section Alerts) for example to advise if the CPU reaches an high value (over 80%) with:

```
sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_irate)>0.8
```

MODULE 4: HELLO OPERATOR

Here I had to face an architectural mismatch:

```
ubuntu@LAB12vm:~/advanced-kubernetes$ helm install hello-operator operator/helm-hello-operator
NAME: hello-operator
LAST DEPLOYED: Tue Jan 27 14:36:16 2026
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
ubuntu@LAB12vm:~/advanced-kubernetes$ kubectl get pods -l app=hello-operator
NAME                  READY   STATUS            RESTARTS   AGE
hello-operator-9fcf564b9-htm7g  0/1    ContainerCreating   0          5s
[ubuntu@LAB12vm:~/advanced-kubernetes$ kubectl get pods -l app=hello-operator
NAME                  READY   STATUS            RESTARTS   AGE
hello-operator-9fcf564b9-htm7g  0/1    ContainerCreating   0          10s
[ubuntu@LAB12vm:~/advanced-kubernetes$ kubectl get pods -l app=hello-operator -w
NAME                  READY   STATUS            RESTARTS   AGE
hello-operator-9fcf564b9-htm7g  0/1    ContainerCreating   0          14s
hello-operator-9fcf564b9-htm7g  0/1    Error             0          29s
hello-operator-9fcf564b9-htm7g  0/1    Error             1 (2s ago)  30s
hello-operator-9fcf564b9-htm7g  0/1    CrashLoopBackOff  1 (2s ago)  31s
[^Cubuntu@LAB12vm:~/advanced-kubernetes$ kubectl logs $(kubectl get pods -l app=hello-operator -o jsonpath=".items[0].metadata.name")"]
exec /usr/local/bin/kopf: exec format error
[ubuntu@LAB12vm:~/advanced-kubernetes$ kubectl logs app=hello-operator
error: error from server (NotFound): pods "app=hello-operator" not found in namespace "default"
[ubuntu@LAB12vm:~/advanced-kubernetes$ kubectl logs hello-operator
error: error from server (NotFound): pods "hello-operator" not found in namespace "default"
[ubuntu@LAB12vm:~/advanced-kubernetes$ cd helm/
[ubuntu@LAB12vm:~/advanced-kubernetes$ helm$ kubectl get pods -l app=hello-operator -w
NAME                  READY   STATUS            RESTARTS   AGE
hello-operator-9fcf564b9-htm7g  0/1    CrashLoopBackOff  3 (23s ago)  96s
[^Cubuntu@LAB12vm:~/advanced-kubernetes$ helm$ helm uninstall hello-operator
release "hello-operator" uninstalled
[ubuntu@LAB12vm:~/advanced-kubernetes$ helm$ helm install hello-operator operator/helm-hello-operator --set image.repository=ghcr.io/unusualfor/hello-operator-arm64 --set image.tag=latest
Error: INSTALLATION FAILED: repo operator not found
```

In fact, by looking at the pods I'm facing a problem between arm64 and x86_64. even though the Operator pod is currently crashing due to architecture incompatibility I have successfully extended the Kubernetes API. By running kubectl get crd, I can show that the cluster now 'understands' what a Hello resource is. I have applied a Custom Resource named francesco, and Kubernetes has stored it in its database (etcd).

```
[^Cubuntu@LAB12vm:~/advanced-kubernetes$ kubectl get crd hellos.unusualfor.com
NAME                CREATED AT
hellos.unusualfor.com  2026-01-27T13:36:14Z
ubuntu@LAB12vm:~/advanced-kubernetes$ kubectl apply -f operator/hello-francesco.yaml
[kubectl get hello
hello.unusualfor.com/francesco created
NAME      AGE
francesco  0s
ubuntu@LAB12vm:~/advanced-kubernetes$
```

HOW DID I PERFORM THE TROUBLESHOOTING FOR THIS PART:

The default container image provided in the lab was compiled for x86_64, causing an Exec format error on my MacBook Air (M3). I performed a local build to ensure the binary was compatible with the ARM64 architecture of my host and VM.

In my normal terminal:

```
cd ~/advanced-kubernetes/operator/app
docker build -t hello-operator:local .
docker save hello-operator:local > hello-operator.tar
```

multipass transfer hello-operator.tar secondaVM:/home/ubuntu/

Since the local Docker image was not available inside the isolated Multipass VM, I transferred the image manually and imported it directly into the containerd runtime of the k0s cluster.

I uninstalled and installed another time helm hello operator:

```
[ubuntu@LAB12vm:~/advanced-kubernetes$ sudo k0s ctr images import /home/ubuntu/hello-operator.tar
unpacking docker.io/library/hello-operator:local (sha256:b5d705324973bbb4aa9326a4b035ae0608974d2e3a23a5c9f746819cac6391ff)...done
ubuntu@LAB12vm:~/advanced-kubernetes$ helm install hello-operator operator/helm-hello-operator \
--set image.repository=hello-operator \
--set image.tag=local \
--set image.pullPolicy=Never
Error: INSTALLATION FAILED: cannot re-use a name that is still in use
ubuntu@LAB12vm:~/advanced-kubernetes$ kubectl get pods -l app=hello-operator
NAME           READY   STATUS    RESTARTS   AGE
hello-operator-67c4f77f4d-7wwfj   0/1     ErrImageFull   0          58m
[ubuntu@LAB12vm:~/advanced-kubernetes$ helm uninstall hello-operator
release "hello-operator" uninstalled
ubuntu@LAB12vm:~/advanced-kubernetes$ helm install hello-operator operator/helm-hello-operator \
--set image.repository=hello-operator \
--set image.tag=local \
--set image.pullPolicy=Never
NAME: hello-operator
LAST DEPLOYED: Tue Jan 27 15:37:04 2026
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
ubuntu@LAB12vm:~/advanced-kubernetes$ kubectl get pods -l app=hello-operator
NAME           READY   STATUS    RESTARTS   AGE
hello-operator-7bbf5d5cb4-5g694   1/1     Running   0          4s
ubuntu@LAB12vm:~/advanced-kubernetes$
```

To prevent Kubernetes from trying to pull the incompatible image from the remote registry, I set the image.pullPolicy to Never, forcing the cluster to use the local sideloaded image.

I verified that the operator was running correctly. Upon applying a Custom Resource (CR), the operator's reconciliation loop successfully detected the event and automatically created a corresponding ConfigMap.

I tried to create a new resource from the template:

```
[ubuntu@LAB12vm:~/advanced-kubernetes$ kubectl logs -l app=hello-operator
[kubectl get configmap -l app=hello-operator
[2026-01-27 14:37:05.953] kopf._core.engines.a [INFO] Initial authentication has finished.
[2026-01-27 14:37:06.084] kopf.objects [INFO] [default/francesco] Timer 'periodic_reconcile' succeeded.
[2026-01-27 14:37:06.187] kopf.objects [INFO] [default/francesco] Handler 'reconcile_hello' succeeded.
[2026-01-27 14:37:06.187] kopf.objects [INFO] [default/francesco] Creation is processed: 1 succeeded; 0 failed.
[2026-01-27 14:38:06.154] kopf.objects [INFO] [default/francesco] Timer 'periodic_reconcile' succeeded.
[2026-01-27 14:39:06.253] kopf.objects [INFO] [default/francesco] Timer 'periodic_reconcile' succeeded.
[2026-01-27 14:40:06.347] kopf.objects [INFO] [default/francesco] Timer 'periodic_reconcile' succeeded.
[2026-01-27 14:41:06.435] kopf.objects [INFO] [default/francesco] Timer 'periodic_reconcile' succeeded.
[2026-01-27 14:42:06.504] kopf.objects [INFO] [default/francesco] Timer 'periodic_reconcile' succeeded.
[2026-01-27 14:43:06.609] kopf.objects [INFO] [default/francesco] Timer 'periodic_reconcile' succeeded.
NAME        DATA   AGE
hello-francesco  1   6m33s
[ubuntu@LAB12vm:~/advanced-kubernetes$ cp operator/hello-francesco.yaml operator/hello-mariapia.yaml
ubuntu@LAB12vm:~/advanced-kubernetes$ sed -i 's/francesco/mariapia/g' operator/hello-mariapia.yaml
[kubectl apply -f operator/hello-mariapia.yaml
hello.unusualfor.com/mariapia created
[ubuntu@LAB12vm:~/advanced-kubernetes$ kubectl get configmap hello-mariapia -o yaml
apiVersion: v1
data:
  greeting: Hello, Francesco!
kind: ConfigMap
metadata:
  creationTimestamp: "2026-01-27T14:45:48Z"
  labels:
    app: hello-operator
  name: hello-mariapia
  namespace: default
  resourceVersion: "19445"
  uid: 2967018a-519f-46d2-b5d0-f4560e3762a4
ubuntu@LAB12vm:~/advanced-kubernetes$
```

I modified the name of the YAML file and see the final output of the new ConfigMap specific for my name (new user).

I have deleted all to free my previous environment. The docker image too from my PC:

helm uninstall hello-operator

```
kubectl delete crd hellos.unusualfor.com  
docker rmi hello-operator:local
```

And from the VM:

```
kubectl get configmap -l app=hello-operator  
helm uninstall hello-operator  
kubectl delete crd hellos.unusualfor.com  
rm /home/ubuntu/hello-operator.tar  
sudo k0s ctr images rm docker.io/library/hello-operator:local  
*****
```

ASSIGNMENT

Assignment: Telemetry

Objective: Deploy a custom telemetry app, ensure Prometheus scrapes its metrics, and create a Grafana dashboard for visualization and analysis.

Prerequisites:

- Prometheus and Grafana are installed and running as per [Module 3](#)
- You have access to your cluster via kubectl and Helm

Steps:

1. Deploy the telemetry app using Helm in the assignment namespace:

- This command installs the telemetry app using the provided Helm chart, creating the assignment namespace if it does not exist.

```
helm install assignment-app oci://ghcr.io/unusualfor/demo-app -n assignment --  
create-namespace
```

2. Verify the app is running in the assignment namespace:

- Check that the pod and service for the demo app are present and running.
3. kubectl get pods -n assignment -l app=demo-app
kubectl get svc -n assignment demo-app

4. Test metrics endpoint:

- Retrieve the ClusterIP of the demo-app service:

```
CLUSTER_IP=$(kubectl get svc -n assignment demo-app -o  
jsonpath='{.spec.clusterIP}')
```

- Access the metrics endpoint exposed by the app:

```
curl http://$CLUSTER_IP:8000/metrics
```

- You should see Prometheus-formatted metrics output. If not, check the pod logs and service configuration.

5. Check Prometheus targets:

- Prometheus is exposed via NodePort. Retrieve the Prometheus server URL:

- ```
export PNODE_PORT=$(kubectl get --namespace default -o jsonpath='{.spec.ports[0].nodePort}' services prometheus-server)
```
- ```
export PNODE_IP=$(kubectl get nodes --namespace default -o jsonpath='{.items[0].status.addresses[0].address}')
```


`echo "Prometheus UI: http://$PNODE_IP:$PNODE_PORT"`
- Query Prometheus targets to confirm your app appears in activeTargets and is up:
`curl http://$PNODE_IP:$PNODE_PORT/api/v1/targets | jq . | grep assignment`

or just access the following and filter per *assignment*:

```
echo "Prometheus UI - Targets: http://$PNODE_IP:$PNODE_PORT/targets"
```

- If your app does not appear or is not up, check the app's service annotations and Prometheus scrape configuration.

6. Build a dashboard in Grafana:

- After confirming metrics are available, create a custom dashboard or panel in Grafana using the metrics exposed by your app (e.g., request count, latency, memory usage). Check again with

```
7. CLUSTER_IP=$(kubectl get svc -n assignment demo-app -o jsonpath='{.spec.clusterIP}')
curl http://$CLUSTER_IP:8000/metrics
```

which are the available metrics and use them appropriately (hint: some of them start with *demo_app*).

- Example tasks:
 - Visualize request rate over time
 - Show average or maximum latency
 - Display current memory usage
- Use Prometheus as the data source and select your app's metrics for visualization.
- Make sure to create multiple panels and, for each panel itself try to relate some metrics together (e.g. how many requests were created and which was the request latency, shown in the same panel).

As in the previous operation part, because of mismatch of architectures with the creation of the new demo app, I had to create an image on my personal docker and copy it on the VM in this way:

```
Host-006:esercizidistr mariapiabuonomo$ cd advanced-kubernetes/
Host-006:advanced-kubernetes mariapiabuonomo$ docker build -t demo-app:local .
[+] Building 0.0s (1/1) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 2B
ERROR: failed to build: failed to solve: failed to read dockerfile: open Dockerfile: no such file or directory
Host-006:advanced-kubernetes mariapiabuonomo$ cd app
-bash: cd: app: No such file or directory
Host-006:advanced-kubernetes mariapiabuonomo$ ls
assignment      helm      infra      istio      operator      README.md
Host-006:advanced-kubernetes mariapiabuonomo$ cd assignment/
Host-006:assignment mariapiabuonomo$ ls
app      helm
Host-006:assignment mariapiabuonomo$ cd app/
Host-006:app mariapiabuonomo$ docker build -t demo-app:local .
[+] Building 6.5s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 152B
=> [internal] load metadata for docker.io/library/python:3.10-slim
=> [auth] library/python:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/4] FROM docker.io/library/python:3.10-slim@sha256:f5d029fe39146b08200bcc73595795ac19b85997ad0e5001a0
=> => resolve docker.io/library/python:3.10-slim@sha256:f5d029fe39146b08200bcc73595795ac19b85997ad0e5001a0
=> [internal] load build context
=> => transferring context: 901B
=> CACHED [2/4] WORKDIR /app
=> [3/4] COPY app.py .
=> [4/4] RUN pip install prometheus_client psutil
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:2a65a547a67b54e05c83410e692b1e52d0ef1f4829c4dc6a8ff3c0628f506aab
=> => exporting config sha256:a65a67fe4d8061b5e1f726c4851d61b133fef660beed82ce9f82c6bbdf024a14
=> => exporting attestation manifest sha256:e98061a27883f6ecf7d30e7aba370d8a8cee8629a600c60a3fb22105e83f5
=> => exporting manifest list sha256:575d5eac34dcc9c7b8061965c16cdd08dbe146567c046f85fe0a5ac44c6721b
=> => naming to docker.io/library/demo-app:local
=> => unpacking to docker.io/library/demo-app:local
Host-006:app mariapiabuonomo$ docker save demo-app:local > demo-app.tar
Host-006:app mariapiabuonomo$ multipass transfer demo-app.tar LAB12vm:/home/ubuntu
Host-006:app mariapiabuonomo$
```

Then I went back to the VM in order to open the docker files and use the demo app by uninstalling and re-installing it. (in the following screenshot I put the wrong name at first)

```
[ubuntu@LAB12vm:~/advanced-kubernetes]$ sudo k8s ctr images import /home/ubuntu/demo-app.tar
unpacking docker.io/library/demo-app:local (sha256:575d5eac34dcc9c7b8061965c16cdd08dbe146567c046f85fe0a5ac44c6721b)...done
[ubuntu@LAB12vm:~/advanced-kubernetes]$ helm uninstall demo-app
Error: uninstall: Release not loaded: demo-app: release: not found
[ubuntu@LAB12vm:~/advanced-kubernetes]$ helm install assignment-app oci://ghcr.io/unusualfor/demo-app -n assignment --create-namespace
Pulled: ghcr.io/unusualfor/demo-app:0.1.0
Digest: sha256:001f32c6dfa671ff3ca8fb88683ba2755c9ae1d32997c81c465d29f0f471581
Error: INSTALLATION FAILED: cannot re-use a name that is still in use
[ubuntu@LAB12vm:~/advanced-kubernetes]$ helm uninstall demo-app
Error: uninstall: Release not loaded: demo-app: release: not found
[ubuntu@LAB12vm:~/advanced-kubernetes]$ helm uninstall assignment-app -n assignment
release "assignment-app" uninstalled
[ubuntu@LAB12vm:~/advanced-kubernetes]$ helm install assignment-app oci://ghcr.io/unusualfor/demo-app -n assignment \
--set image.repository=demo-app \
--set image.tag=local \
--set image.pullPolicy=Never
Pulled: ghcr.io/unusualfor/demo-app:0.1.0
Digest: sha256:001f32c6dfa671ff3ca8fb88683ba2755c9ae1d32997c81c465d29f0f471581
NAME: assignment-app
LAST DEPLOYED: Tue Jan 27 16:06:02 2026
NAMESPACE: assignment
STATUS: deployed
REVISION: 1
TEST SUITE: None
[ubuntu@LAB12vm:~/advanced-kubernetes]$ kubectl get pods -n assignment -l app=demo-app -w
NAME           READY   STATUS    RESTARTS   AGE
demo-app-545c9b6cc6-ll2gv  1/1     Running   0          18s
^C[ubuntu@LAB12vm:~/advanced-kubernetes]$
```

Now that is running I first test the endpoint:

```
Cubuntu@LAB12vm:~/advanced-kubernetes$ (kubectl get svc -n assignment demo-app -o jsonpath='{.spec.clusterIP}')$ curl http://$CLUSTER_IP:8000/metrics
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 391.0
python_gc_objects_collected_total{generation="1"} 133.0
python_gc_objects_collected_total{generation="2"} 0.0
# HELP python_gc_objects_uncollectable_total Uncollectable objects found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generation="0"} 0.0
python_gc_objects_uncollectable_total{generation="1"} 0.0
python_gc_objects_uncollectable_total{generation="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 46.0
python_gc_collections_total{generation="1"} 4.0
python_gc_collections_total{generation="2"} 0.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython",major="3",minor="10",patchlevel="19",version="3.10.19"} 1.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 1.84029184e+08
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 2.2786048e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.76952636198e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.23
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 6.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 999999.0
# HELP demo_app_requests_total Total requests to demo app
# TYPE demo_app_requests_total counter
demo_app_requests_total 37.0
# HELP demo_app_requests_created Total requests to demo app
# TYPE demo_app_requests_created gauge
demo_app_requests_created 1.769526362925001e+09
# HELP demo_app_request_latency_seconds Request latency in seconds
# TYPE demo_app_request_latency_seconds histogram
demo_app_request_latency_seconds_bucket{le="0.005"} 0.0
demo_app_request_latency_seconds_bucket{le="0.01"} 0.0
demo_app_request_latency_seconds_bucket{le="0.025"} 0.0
demo_app_request_latency_seconds_bucket{le="0.05"} 0.0
demo_app_request_latency_seconds_bucket{le="0.075"} 0.0
```

And then I find the Prometheus and Grafana URL as done in the previous module 3.

```
export PNODE_PORT=$(kubectl get -n default svc prometheus-server -o jsonpath='{.spec.ports[0].nodePort}')
export PNODE_IP=$(kubectl get nodes -o jsonpath='{.items[0].status.addresses[0].address}')
echo "Prometheus UI: http://\$PNODE\_IP:\$PNODE\_PORT/targets"
```

The output is:

Prometheus UI: <http://192.168.64.11:30303/targets>

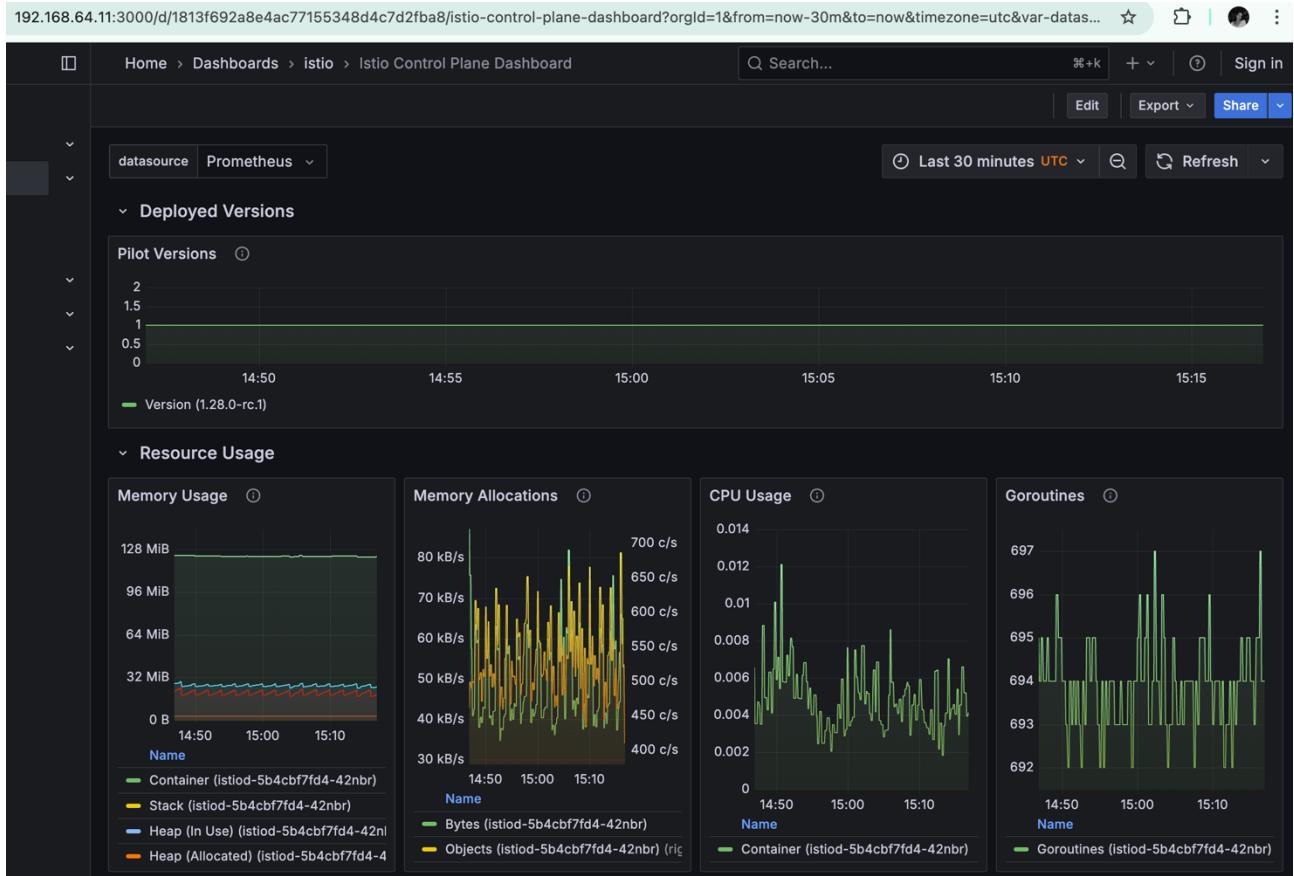
To open Prometheus I just click on the link.

So I open another terminal of my VM and start the forwarding to the wanted port.

For Grafana I have:

<http://192.168.64.11:3000>

Here I will find all the Istio dashboards.



Examples of results:

During the traffic generation phase, I performed connectivity testing using curl against the Service ClusterIP. I ensured the application was reachable on port 8000 by verifying the Pod readiness probe and the Service selector matching the assignment namespace. So I performed this while cycle and visualize possible results on Grafana:

The while loop used for traffic generation is a critical part of the observability test.

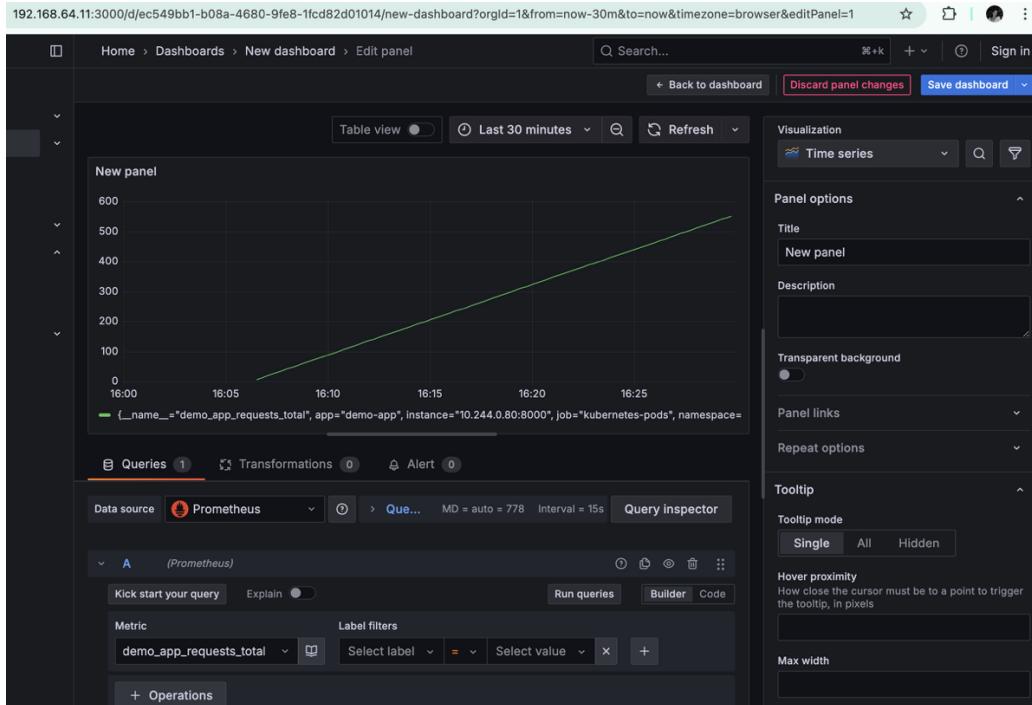
- curl -s -m 2 http://\$CLUSTER_IP:8000/hash: This command performs a silent request to the application's /hash endpoint with a 2-second timeout.
- || echo "Connection ERROR": This is a defensive programming practice to immediately signal any network failure or pod crash during the test.

By sending requests every second, I am creating a constant load that allows Prometheus to calculate stable rates (per-second averages) rather than showing erratic spikes.

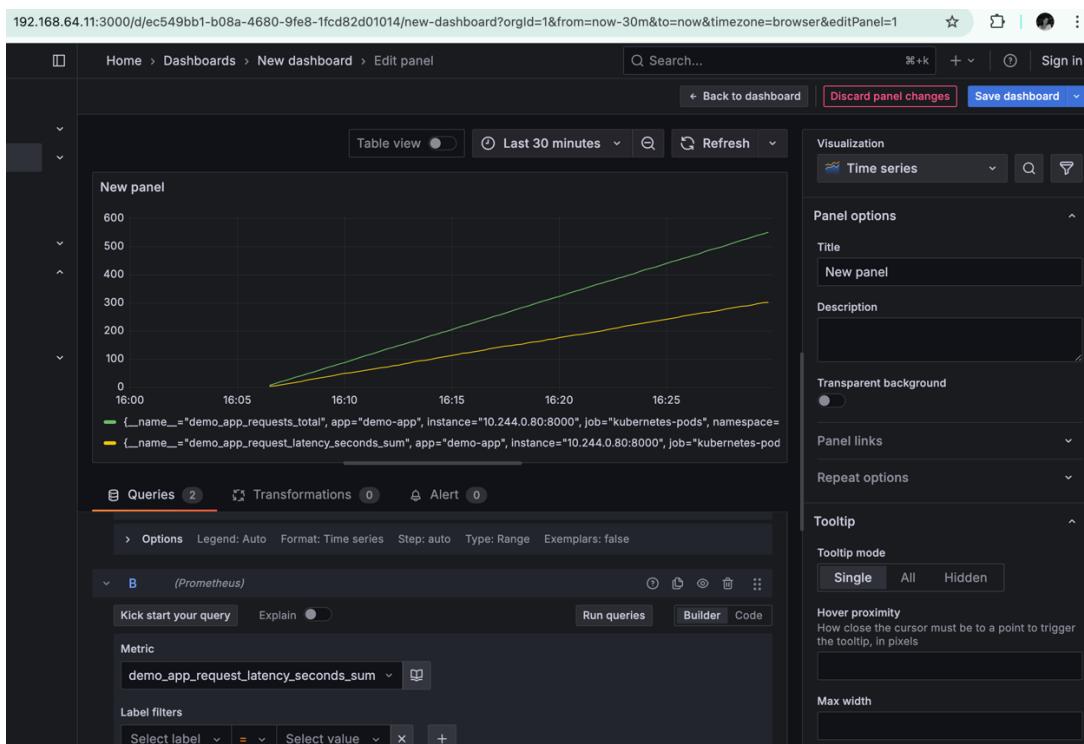
```
while true; do
echo -n "$(date +%H:%M:%S) - Sending the request... "
curl -s -m 2 http://$CLUSTER_IP:8000/hash || echo "Connection ERROR"
sleep 1
done
```

When running the curl command, the application returns raw Prometheus metrics.

demo_app_requests_total: This is a Counter. It only goes up. In the results, it shows over 500 requests, confirming the application has been active and stable.



The metric `demo_app_request_latency_seconds_sum` represents the cumulative time, in seconds, that the application has spent processing all incoming requests since the last restart.



This is how much the CPU was used:

