TECH CHALLENGE

TLDR

Unset

```
- install minikube
- install docker
- install helm
- install act
- clone repo
- cd into repo
- run docker build -t print-timestamp:alpine .
- minikube start
- minikube image load print-timestamp:alpine
- minikube addons enable metrics-server
- helm repo add prometheus-community
https://prometheus-community.github.io/ helm-charts
- helm upgrade --install prometheus
prometheus-community/prometheus
- kubectl expose service prometheus-server --type=NodePort
--target-port=9090 --name=prometheus-server-np
- minikube service prometheus-server-np --url
- helm repo add grafana https://grafana.github.io/helm-charts
- helm upgrade --install grafana grafana/grafana
- kubectl expose service grafana --type=NodePort
--target-port=3000 --name=grafana-np
## get and record password - user is admin
- kubectl get secret --namespace default grafana -o
jsonpath="{.data.admin-password}" | base64 --decode ; echo
- minikube service grafana-np --url
- helm upgrade --install timestamp print-timestamp
- kb get pods
- kb port-forward
pod/print-timestamp-<from-prev-command-1234567890> 4430:4430
- curl 127.0.0.1:4430/timestamp
- time output in json
```

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇ MY PROGRESS ⬇⬇⬇⬇⬇⬇⬇⬇⬇

Unset

```
- Python created
- Docker image created
- Helm chart completed for print-timestamp server
- Print-timestamp using flask serves timestamp from K3s
- Grafana and Prometheus up and running
- Started ci-cd using git hub actions
- todo is at he end of readme
```

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇ MY PROCESS ⬇⬇⬇⬇⬇⬇⬇⬇⬇

I created a Repo and cloned this locally

PYTHON SCRIPT

In this repo I firstly created the python script and named this `print_timestamp.py`

Unset

```
import time

# ts stores the time in seconds
ts = time.time()

# print the current timestamp
print(ts)
```

Running with `python3 print_timestamp.py` allowed me to test that the script was successful and ensured that the ask of attaining console type output can be cross-checked and referenced to the output of the kubernetes cluster once deployed.

The naming format utilising an underscore was retained as important so that all understood the name to be referenced.

Unset

```
> python3 print_timestamp.py
1716319109.759373
```

SERVER

To ensure a small image size, it was decided to use the built-in Python server to return the time.

Unset

```
#import need libraries
import time
from http.server import BaseHTTPRequestHandler, HTTPServer

#Class to be used
class RequestHandler(BaseHTTPRequestHandler):
def do_GET(self):
    # Get the current timestamp
    ts = time.time()

    # Send response status code and headers
    self.send_response(200)
    self.send_header('Content-type', 'text/plain')
    self.end_headers()

    # Send the timestamp as a response
    self.wfile.write(str(ts).encode())

#Run the server and allow response
def run(server_class=HTTPServer,
handler_class=RequestHandler, port=4430):
    server_address = ('', port)
    httpd = server_class(server_address, handler_class)
    print(f'Starting httpd server on port {port}...')
    httpd.serve_forever()

if __name__ == '__main__':
    run()
```

This was run in one terminal and in a second running `curl 127.0.0.1:4430` calls to the server and returned `1716319485.656218%`

This was different to the original output, adding %. After research, this was due to curl adding this to indicate the end of line as python was not returning a new line. New line was added to the script and not the % was no longer seen `self.wfile.write((str(ts) + '\n').encode())`

DOCKER

Now we need to create the docker image to be used in K8s.

Unset

```
# Use a lightweight Python image
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /app

# Copy the Python script into the container
COPY print_timestamp.py .

# Command to run the Python script
CMD ["python", "print_timestamp.py"]
```

Running `docker build -t print_timestamp:latest .` this will build the image locally for us to run.

Unset

```
> docker images
REPOSITORY            TAG       IMAGE ID       CREATED
SIZE
print_timestamp      latest    6b323af908dc   55 minutes ago
152MB
```

Running `docker run -p 4430:4430 print_timestamp` this will now run the image locally to allow testing

Unset

```
> docker ps -a
CONTAINER ID    IMAGE           COMMAND           CREATED
STATUS          PORTS
780f9746a412     print_timestamp   "python print_timest…"
35 seconds ago     Up 34 seconds  0.0.0.0:4430->4430/tcp
```

Again running the curl command will call the running server that is open on port 4430, this giving the expected and same reply as the original script output.

K8s

MINIKUBE

Now to look at the K8s section and to allow local testing, I used minikube. This can be built from here

Once installed we can start minikube with `minikube start`

Running `kb get namespace` will show minikube running.

Unset

```
> kb get namespaces
NAME              STATUS   AGE
default           Active   100m
kube-node-lease   Active   100m
kube-public       Active   100m
kube-system       Active   100m
```

We created a deployment yaml.

Unset

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: print_timestamp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: print_timestamp
  template:
    metadata:
      labels:
        app: print_timestamp
    spec:
      containers:
        - name: print_timestamp
          image: print-timestamp:latest
          ports:
            - containerPort: 4430
```

On trying to run this (`kb apply -f k8s/deployment.yaml`) the underscore I had retained in the naming convention bit me with the error.

Unset

```
The Deployment "print_timestamp" is invalid:
spec.template.spec.containers[0].name: Invalid value:
"print_timestamp": a lowercase RFC 1123 label must consist of
lower case alphanumeric characters or '-', and must start and
end with an alphanumeric character (e.g. 'my-name',  or
'123-abc', regex used for validation is
'[a-z0-9]([-a-z0-9]*[a-z0-9])?')
```

I changed all naming to change the _ (snake) to - (kebab) to maintain the ease and consistency.

Unset

```
> cp print_timestamp.py print-timestamp.py
> docker build -t print-timestamp:latest .
> docker images
REPOSITORY        TAG       IMAGE ID       CREATED
SIZE
print-timestamp  latest    2cf3cf13630f   57 seconds ago
152MB
print_timestamp  latest    6b323af908dc   2 hours ago
152MB
> docker rmi 6b323af908dc
Untagged: print_timestamp:latest
Deleted:
sha256:6b323af908df86dc8c7d70c446683d4dc7b97145b0597a644352b2
f6b38bcc
```

I realised this image appeared to be a little chunky and so changed to use alpine as the base.

Unset

```
# Use a lightweight image
FROM alpine:latest

# Install Python3 and pip
RUN apk add --no-cache python3

# Set the working directory in the container
WORKDIR /app

# Copy the Python script into the container
COPY print-timestamp.py .

# Command to run the Python script
CMD ["python", "print-timestamp.py"]
```

Unset

```
> docker build -t print-timestamp:alpine .
> docker images
REPOSITORY                    TAG       IMAGE ID
CREATED          SIZE
print-timestamp               alpine    384f39153a36   3
minutes ago   76MB
```

```
print-timestamp              latest    2cf3cf13630f   8
minutes ago   152M
```

This was half the size so much more pleasing, it was tested to ensure it was working correctly.

I now updated the deployment yaml to kebab case

Unset

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: print-timestamp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: print-timestamp
  template:
    metadata:
      labels:
        app: print-timestamp
    spec:
      containers:
        - name: print-timestamp
          image: print-timestamp:latest
          ports:
            - containerPort: 4430
```

Again running

Unset

```
> kb apply -f k8s/deployment.yaml
deployment.apps/print-timestamp created
> kb get pods -n default
NAME                              READY   STATUS    RESTARTS
AGE
print-timestamp-ff5bb948f-9kpdr   0/1     Running   0
8m21s
```

We now create the service yaml

Unset

```
apiVersion: v1
kind: Service
metadata:
  name: print-timestamp
spec:
  selector:
    app: print-timestamp
  ports:
    - protocol: TCP
      port: 4430
      targetPort: 4430
  type: ClusterIP
```

We now run and check the services

Unset

```
> kb apply -f k8s/service.yaml
service/print-timestamp created
> kb get services
NAME              TYPE        CLUSTER-IP    EXTERNAL-IP
PORT(S)    AGE
kubernetes        ClusterIP   10.96.0.1     <none>
443/TCP    3h5m
print-timestamp   ClusterIP   10.99.34.31   <none>
4430/TCP   5m15s
```

We now use port forwarding to allow local connection to the cluster

Unset

```
kubectl port-forward pod/print-timestamp-5b76b58757-84z2k
4430:4430

Forwarding from 127.0.0.1:4430 -> 4430
Forwarding from [::1]:4430 -> 4430
```

Again using curl we are able to gain a time stamp

Unset

```
curl 127.0.0.1:4430
1716331479.8028572
```

With output given in the first terminal that the connection was made and served

```
Handling connection for 4430
```

HORIZONTAL POD AUTOSCALER

We create a scaling policy

Unset

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: print-timestamp
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: print-timestamp
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
```

We apply this and can now test.

Unset

```
kb apply -f
k8s/service.yaml
```

HELM

I created a folder for the helm chart and built out the files from the previous deployment, hpa and service yaml. I adapted to put in frequesntly used vars from the values file and created the Chart.yaml

FLASK

The python server was replaced by a flask server

Unset

```
from flask import Flask, jsonify
import time

app = Flask(__name__)
metrics = PrometheusMetrics(app)

@app.route('/timestamp', methods=['GET'])
def get_timestamp():
  ts = time.time()
  return jsonify({'timestamp': ts})

if __name__ == '__main__':
  app.run(host='0.0.0.0', port=4430)
```

The code checked

Unset

```
> python3 print_timestamp.py
> curl 127.0.0.1:4439/timestamp
```

The docker image was rebuilt, launched into minikube and again tested to ensure the output was still coming through.

PROMETHEUS and GRAFANA

I was able to _after alot of searching_ spin up these us in the cluster.

Using helm made this so easy to get started

Unset

```
helm repo add prometheus-community
https://prometheus-community.github.io/ helm-charts
- helm upgrade --install prometheus
prometheus-community/prometheus
- kubectl expose service prometheus-server --type=NodePort
--target-port=9090 --name=prometheus-server-np
- minikube service prometheus-server-np --url
- helm repo add grafana https://grafana.github.io/helm-charts
- helm upgrade --install grafana grafana/grafana
- kubectl expose service grafana --type=NodePort
--target-port=3000 --name=grafana-np
## get and record password - user is admin
- kubectl get secret --namespace default grafana -o
jsonpath="{.data.admin-password}" | base64 --decode ; echo
- minikube service grafana-np --url
```

and i had the set up running.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇ TODO ⬇⬇⬇⬇⬇⬇⬇⬇⬇

Unset

```
- I have tried to build github actions pipeline but as yet
not working (mac is causing issues)
- I am struggling to get the CPU output from my pod for the
metrics server to read and hpa to scale
- I am still to successfully pull the metrics to prometheus -
i have the config in
```

Running `hey -z 30s -c 100` `http://127.0.0.1:4430` places load on the cluster and should force scaling