Premisa:

Esta prueba técnica pretende evaluar el proceso de razonamiento y la capacidad de resolución de problemas.

Parte I – Análisis y solución de problemas

***** DOCKER + TROUBLESHOOTING *****

Se está corriendo una aplicación en Python Flask (El código es irrelevante para este ejercicio, puede ser cualquiera) usando **gunicorn** y **haproxy** todo dentro de un mismo contenedor.

El **dockerfile** está como se muestra:

Archivo: Dockerfile

```
FROM alpine
RUN apk add py3-pip build-base python3-dev libffi-dev openssl-dev
RUN apk add nginx
RUN mkdir -p /opt/api
WORKDIR /opt/api
ADD api/requirements.txt /opt/api
RUN pip3 install --no-cache-dir -r requirements.txt
ADD api/. /opt/api
ADD ./docker-entrypoint.sh /bin/docker-entrypoint
ADD ./haproxy.conf /etc/haproxy/haproxy.cfg
EXPOSE 80
CMD ["/bin/docker-entrypoint"]
```

Archivo: docker-entrypoint.sh

```
#!/bin/sh

echo"Starting gunicorn..."
gunicorn -w 5 -b 127.0.0.1:9000 app:app –daemon
sleep 3

echo"Starting haproxy..."
haproxy -f "/etc/haproxy/haproxy.cfg" &
```

Archivo: haproxy.cfg

```
Global
maxconn 8192

defaults
log stdout format raw local0
```

```
mode    http
option  httplog
option  forwardfor
option  httpclose
option  dontlognull

timeout connect 10s
timeout client  150s
timeout server  150s

errorfile 400 /usr/local/etc/haproxy/errors/400.http
errorfile 403 /usr/local/etc/haproxy/errors/403.http
errorfile 408 /usr/local/etc/haproxy/errors/408.http
errorfile 500 /usr/local/etc/haproxy/errors/500.http
errorfile 502 /usr/local/etc/haproxy/errors/502.http
errorfile 503 /usr/local/etc/haproxy/errors/503.http
errorfile 504 /usr/local/etc/haproxy/errors/504.http

maxconn 8192

frontend app-http
bind *:80

acl is_app  path_beg -i  /
use_backend flask_backend  if is_app

backend flask_backend
server docker-app cll-dsb-auth:9000 check verify none
```

La aplicación funciona correctamente, sin embargo, en algunos momentos se muestra inestable o el servicio deja de responder retornando errores 500,el contenedor sigue corriendo, aunque el servicio no esté operativo.

Con lo anterior:

- Hacer un análisis con lo expuesto e identificar que errores considera que se están cometiendo.
- Que mejoras y/u optimizaciones se pueden llevar a cabo.
- Como SysAdmin, ¿Cómo se manejaría los logs y de qué forma se presentarán para ser usado en una herramienta de monitoreo?

Parte II – Ejercicio práctico

***** KUBERNETES *****

Cumplir los siguientes requerimientos:

✓Tener un usuario GitHub. ✓Tener un usuario de DockerHub. ✓Kubernetes local (minikube) para ejecutar validar lo entregado. ✓Haber instalado en su equipo kubectl y helm.

Con el punto anterior (DOCKER + TROUBLESHOOTING) desarrolle el siguiente ejercicio:

- Aplique las mejoras y/o correcciones necesarias para desplegar este servicio usando docker. Para el ejemplo en flask puede usar un simple "hello world" que encuentre en internet.
- Con el o los contenedores creados en el paso previo, genere los manifiestos de Kubernetes necesarios para desplegar este servicio.
- Incluya los probes de kubernetes necesarios para validar que el servicio este sirviendo correctamente.
- Usando el siguiente enlace: https://www.magalix.com/blog/monitoring-of-kubernetes-cluster-through-prometheus-and-grafana y usando minikube, despliegue el prometheus y visualice las métricas del servicio creado.

Una vez que ha finalizado con el desarrollo del ejercicio, debe:

1. subir a “DockerHub” los contenedores creados en un proyecto público para que el calificador los pueda descargar y evaluar.
2. Todos los archivos fuente generados deben ser colocados en el “GitHub” del aspirante en un proyecto público para su verificación.
3. Los manifiestos de Kubernetes que despliegue Pods en sus diferentes tipos de objetos, deben apuntar a la dirección de “DockerHub” del primer paso.