SISTEMAS OPERATIVOS 1

MANUAL TÉCNICO

Grupo No. 9 - Integrantes

Pablo Gerardo Garcia Perusina – 200511819

Ludwin Romario Burrion Imuchac – 201314001

Christian Enrique Ramos Alvarez – 201504444

Aplicación web

Para la aplicación web se utilizó React utilizando Hooks para los diferente componentes a utilizar, también se hizo uso de las siguientes librerías:

react-bootstrap: Esta librería se utilizó para la maquetación y para que la aplicación sea de manera responsiva https://react-bootstrap.github.io/

Chart.js: Gracias a esta potente librería que genera diferentes tipos de gráficos estadísticos, se pudo implementar en la aplicación web los gráficos de los datos consultados en las bases de datos https://www.chartjs.org/

(API)

La aplicación encargada de entregar los datos a la aplicación web se realizó con node Js utilizando el framework express.

Para la conexión con la base de datos de mongodb se utilizó el paquete de mongoose para manejar los Modelos que se utilizaron, con este paquete se realizaron los reportes utilizando la funcionalidad de aggregate nativo de mongodb.

Para la conexión con redis se utilizó el paquete redis, que nos permite ejecutar los comandos de redis desde nodejs.

Google Cloud Platform Kubernetes

A continuación encontrará los comandos utilizados para la creación de cada sección del proyecto, cluster, namespaces, deployments, servicios, ingresses y observabilidad.

Cluster

- *gcloud container clusters create covid19 --num-nodes=2 --tags=allin,allout --enable-legacy-authorization --enable-basic-auth --issue-client-certificate --machine-type=n2-standard-2*

Se utilizaron 2 nodos de tipo n2-standard-2 para soportar la carga del sistema.

Namespaces

Se crearon 3 namespaces, **project**, **frontend** y **observability**.

- *kubectl create namespace* ***project***
- *Kubectl create namespace* ***frontend***
- *Kubectl create namespace* ***observability***

Deployments

Se crearon múltiples deployments en cada namespace. A continuación se listan los comandos usados para crearlos.

- **Namespace Project**
  - kubectl create deployment pyserver-grpc --image=registry.hub.docker.com/petzydrummer/python-app-grpc -n project
  - kubectl create deployment go-webserver-grpc --image=registry.hub.docker.com/petzydrummer/go-webserver-grpc -n project
  - kubectl create deployment go-webserver-rmq --image=registry.hub.docker.com/petzydrummer/go-webserver -n project

    - kubectl create deployment pyserver --image=registry.hub.docker.com/petzydrummer/python-app -n project

- **Namespace Frontend**
    - kubectl create deployment nodejs-api --image=registry.hub.docker.com/petzydrummer/nodejs-api -n frontend
    - kubectl create deployment react-web-app --image=registry.hub.docker.com/petzydrummer/react-web-app -n frontend

Servicios

Se crearon múltiples servicios en cada namespace para poder exponer interna o externamente los deployments creados. A continuación se listan los comandos usados para crearlos.

- **Namespace Project**
    - kubectl expose deployment go-webserver-rmq --name=goservicermq --port=80 --target-port=80 -n project
    - kubectl expose deployment go-webserver-grpc --name=go-service-grpc --port=80 --target-port=80 -n project
    - kubectl expose deployment pyserver-grpc --name=python-service-grpc --port=50051 --target-port=50051 -n project

- **Namespace Frontend**
    - kubectl expose deployment nodejs-api --name=nodejs-api-service --port=50501 --target-port=50501 --type=LoadBalancer -n frontend
    - kubectl expose deployment react-web-app --name=react-web-app-service --port=80 --target-port=5000 --type=LoadBalancer -n frontend

Ingress

Se crearon 2 ingress en el namespace **Project** con el fin de exponer externamente los servidores web creados en Golang. A continuación se listan los comandos usados para crearlos.

- **Gloo Ingress**
  - helm install gloo gloo/gloo --namespace project --set gateway.enabled=false,ingress.enabled=true
  - kubectl create -f gloo.yaml

Adjunto en el proyecto se encuentra el archivo **gloo.yaml** con la especificación para la creación del ingress gloo.

- **Gloo Nginx**
  - helm install nginx-ingress stable/nginx-ingress -n proyecto
  - kubectl create -f nginx.yaml

Adjunto en el proyecto se encuentra el archivo **nginx.yaml** con la especificación para la creación del ingress Nginx.

Observabilidad

Para implementar la observabilidad en el proyecto, se utilizó **Linkerd** y **Jaeger.** A continuación listamos los comandos utilizados para instalar e implementar cada una de las herramientas.

Linkerd

Instalación de Linkerd localmente

- curl -sL https://run.linkerd.io/install | sh

Agregar binario de Linkerd a Path del SO.

- export PATH=$PATH:$HOME/.linkerd2/bin #add line to ~/.profile

Instalación de Linkerd en el cluster covid19

- linkerd install | kubectl apply -f -

Comando para ver localmente el dashboard de Linkerd

- linkerd dashboard

Para inyectar cada deployment del proyecto, se colocó una anotación en la metadata de la sección "***template***" de cada archivo yaml.

```yaml
template:
  metadata:
    annotations:
      linkerd.io/inject: enabled
    labels:
      app: pyserver-grpc
```

De esta manera, al crear el deployment vía el archivo yaml, automáticamente agregamos el contenedor proxy de linkerd.

Jaeger

Se agrega repositorio de jaeger vía helm

- helm repo add jaegertracing https://jaegertracing.github.io/helm-charts

Instalación de Jaeger en el cluster

- helm install jaeger jaegertracing/jaeger \
  --set provisionDataStore.cassandra=false \
  --set provisionDataStore.elasticsearch=true \
  --set storage.type=elasticsearch \
  --set spark.enabled=true \
  --set spark.schedule="*/1 * * * *" \
  --set elasticsearch.minimumMasterNodes=1 \
  --set elasticsearch.replicas=1 \
  -n observability

Comando para observar Dashboard de Jaeger localmente

- kubectl port-forward --namespace observability $POD_NAME 8080:16686