Lab 13: Monitoring
ID720 Virtualisation

Introduction

As we move our applications into containers running in an environment like Kubernetes, we need some extra machinery to give us some visibility into our systems. *Prometheus* is a typical tool for this.

Note that this lab is an updated and simplified version of the tutorial found at `https://cloud.google.com/architecture/monitoring-apps-running-on-multiple-gke-clusters-using-prometheus-and-stackdriver`

1 Setup

To begin, create a project called "lab13" in GCP, and be sure that GKE and monitoring are enabled for this project.

Clone the files needed for this lab from `https://github.com/tclark/lab13-files` into your cloud shell.

In your cloud shell, you will need to install *Helm*, a package manager for Kubernetes app. Do this by checking `https://helm.sh/docs/intro/install/` and following the directions for installing with Apt.

Launch a cluster:

```
gcloud container clusters create lab13 \
--zone us-west2-a \
--num-nodes 3 \
--machine-type n1-standard-2
```

2 Create Service Account

You will need a service account with permissions to write data to the GCP monitoring service. Create the account with

```
gcloud iam service-accounts create prometheus --display-name prometheus-service-account
```

and then save some values in environment variables

```
export PROJECT_ID=$(gcloud info --format='value(config.project)')

export PROMETHEUS_SA_EMAIL=$(gcloud iam service-accounts list \
   --filter="displayName:prometheus-service-account" \
   --format='value(email)')
```

and finally give the service account a role to write metrics data

```
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
--role roles/monitoring.metricWriter \
--member serviceAccount:${PROMETHEUS_SA_EMAIL}
```

3 Install Prometheus

Be sure to `cd` into the `lab13-files` directory.

Create a namespace for the prometheus resources

```
kubectl create namespace prometheus
```

Create a Kubernetes service account associated with the service account you created above.

```
kubectl apply -f prometheus-service-account.yaml
```

Load the configMap that will supply configuration to the Prometheus server.

```
kubectl apply -f prometheus-service-account.yaml
```

Set up some environment variables that we will apply to the Prometheus deployment.

```
export KUBE_NAMESPACE=prometheus
export KUBE_CLUSTER=lab13
export GCP_LOCATION=us-west2-a
export GCP_PROJECT=$(gcloud info --format='value(config.project)')
export DATA_DIR=/prometheus
export DATA_VOLUME=prometheus-storage-volume
export SIDECAR_IMAGE_TAG=0.8.2
export PROMETHEUS_VER_TAG=v2.19.3
```

Now we combine these values into our deployment config and apply it.

```
envsubst < gke-prometheus-deployment.yaml | kubectl apply -f -
```

After a few minutes, verify that the Prometheus deployment is ready.

```
kubectl get pods -n prometheus
```

4 Check the Prometheus UI

Prometheus' role is to collect data from targets in our cluster and store the data in a time series database. We'll use that data to populate our dashboard later. First let's inspect Prometheus' web UI.

Set up port forwarding.

```
export PROMETHEUS_POD_GKE=$(kubectl get pods \
--namespace prometheus -l "app=prometheus-server" \
  -o jsonpath="{.items[0].metadata.name}")

kubectl --context gke port-forward --namespace prometheus\
$PROMETHEUS_POD_GKE 9090:9090 >> /dev/null &
```

In cloud shell, click web preview > change port and enter 9090 for the new port number. Click "Change and preview". You should see the Prometheus web UI. In it, click Status > Service Discovery to see the services that Prometheus has found.

5 Install Postgres

We will use Postgresql to give us something interesting to monitor. We will install it with Helm.

First, add a repository and update Helm.

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update
```

Next, install Postgres.

```
helm install pg bitnami/postgresql \
--set metrics.enabled=true \
--set postgresqlDatabase=prometheusdb \
--set auth.database=labdb
```

Finally, prepare the database for later testing.

```
# get the pg password from a k8s secret
export POSTGRES_PASSWORD=$(kubectl get secret pg-postgresql -o
jsonpath="{.data.postgres-password}" | base64 --decode)

# port-forward the database
kubectl --context gke port-forward \
svc/gke-postgresql 5432:5432 >> /dev/null &

# log into the database
 PGPASSWORD="$POSTGRES_PASSWORD" psql --host 127.0.0.1 -p 5432 -U postgres

# in psql, create a database
CREATE DATABASE promtest
```

6 Create a Monitoring Dashboard

In the cloud console, select Monitoring from the main menu (you'll have to scroll down a bit). Once it's loaded, choose the Metrics explorer.

1. In the Find resource type and metric field, enter `pg_stat_database_blks_read`.
2. Select the metric that starts with `external.googleapis.com/prometheus/`
3. For the resource, select Kubernetes Container
4. In the Group By drop-down list, select `cluster_name`.
5. For Aggregator, select `sum`.
6. Click Save Chart. Name the chart.
7. In the Dashboard drop-down list, select New Dashboard. Enter the dashboard name Postgres and click Save.
8. Click the Dashboards link, and then click the PostgreSQL dashboard.

You should see a dashboard that plots a graph with our Postgres metric.

7 Generate Postgres Traffic

Use the tool `pgbench` to generate traffic on the database. Install it, if necessary, by installing the `postgresql-contrib` package.

In your cloud shell initialise the database with

```
PGPASSWORD="$POSTGRES_PASSWORD" pgbench -i \
-h localhost -p 5431 -U postgres -d promtest
```

And then run the following test, which will run for ten minutes.

```
PGPASSWORD="$POSTGRES_PASSWORD" pgbench -c 10 -T 600 \
-h localhost -p 5431 -U postgres -d promtest
```

While this is running, check the monitoring dashboard. You should see the load from the test on the graph. Toggle auto-refresh on so that the graph will refresh.

8 Clean up

This lab uses a number of billable resources, so be sure to remove everything after you finish. The easiest way to do this is to delete the entire lab13 project.