ElasticSearch Database Backup and Cloud Storage Upload

Introduction

We were tasked with backing up an ElasticSearch database and sending it to a cloud storage bucket. To achieve this, we utilized ElasticDump to export the database indices and mappings, and then uploaded them to Google Cloud Storage (GCS).

Pre-requisites

- Ensure ElasticDump is installed. Install it via npm:
- bashCopy code
- sudo npm install elasticdump -g
- Obtain the ElasticSearch credentials (username and password) and the GCS bucket details.

Script

Below is the script used to backup the ElasticSearch database and upload it to GCS:

bashCopy code

#!/bin/bash

Elasticsearch details

ELASTICSEARCH_HOST="ec-deploy-es-internal-http.default.svc.cluster.local"

ELASTICSEARCH_PORT="9200"

ELASTICSEARCH_USERNAME="elastic"

ELASTICSEARCH_PASSWORD="7Uod31i2NPw7D8tvz63VC50D"

GCS bucket details

GCS_BUCKET="disearch_k8_es_db_backup/es_backups"

```
# Get index names and save to a JSON file
curl -k -u "${ELASTICSEARCH_USERNAME}:${ELASTICSEARCH_PASSWORD}" -X GET "https://${ELASTICSEARCH_HOST}:${ELASTICSEARCH_PORT}/_cat/indices" | awk '{print $3}' > indexname.json

# Iterate over each index and export
while IFS= read -r index; do
  echo "Exporting index: $index"

  # Export the index data
  echo "Exporting data for index: $index"
  NODE_TLS_REJECT_UNAUTHORIZED=0 elasticdump \
    --input="https://${ELASTICSEARCH_USERNAME}:${ELASTICSEARCH_PASSWORD}@${ELASTICSEARCH_HOST}:${ELASTICSEARCH_PORT}/${index}" \
    --output="${index}_data.json" \
    --type=data

  # Upload the exported index data to GCS
  gsutil cp "${index}_data.json" "gs://${GCS_BUCKET}/${index}_data.json"

  # Clean up the exported index data file
  rm "${index}_data.json"

  # Export the index mapping
```

```
    echo "Exporting mapping for index: $index"

    NODE_TLS_REJECT_UNAUTHORIZED=0 elasticdump \

      --
input="https://${ELASTICSEARCH_USERNAME}:${ELASTICSEARCH_PASSWORD}@${ELASTICS
EARCH_HOST}:${ELASTICSEARCH_PORT}/${index}" \

      --output="${index}_mapping.json" \

      --type=mapping


    # Upload the exported index mapping to GCS

    gsutil cp "${index}_mapping.json" "gs://${GCS_BUCKET}/${index}_mapping.json"


    # Clean up the exported index mapping file

    rm "${index}_mapping.json"

done < indexname.json


# Remove the temporary JSON file

rm indexname.json
```

Testing

We tested this script in our test environment by first port forwarding the cluster's ElasticSearch service:

bashCopy code

kubectl port-forward service/service-name oursetport:9200

Then, we set the ElasticSearch username and password and used localhost:9200 as the host and port in the script. We triggered the script, which successfully retrieved the indices data and mappings and sent them to the GCS bucket.

Kubernetes CronJob Deployment

To automate the backup process, we deployed the script as a Kubernetes CronJob. Below is the Dockerfile and CronJob configuration:

Dockerfile

```
FROM google/cloud-sdk:latest

WORKDIR /tmp

# Install dependencies

RUN apt-get update && \

    apt-get install -y npm && \

    npm install -g elasticdump

# Copy the script

COPY ./script.sh .

# Set execute permissions for the script

RUN chmod +x script.sh

# Define the command to run the script

CMD ["/bin/bash", "script.sh"]
```

Build the Docker image, tag it, and push it to Google Container Registry (GCR).

Kubernetes CronJob

yamlCopy code

```
apiVersion: batch/v1

kind: CronJob

metadata:

  name: cronjob-es-db-backup
```

```
spec:
 schedule: "53 05 * * *"
 jobTemplate:
  spec:
   template:
    spec:
     containers:
     - name: es-db-backup-script
      image: gcr.io/disearch/es_db_backup:latest
     restartPolicy: OnFailure
```

This CronJob is scheduled to run every day at 5:53 AM UTC, triggering the script within the Docker container to perform the ElasticSearch database backup and upload to GCS.

Service Account:

we need to grant access to gcp bucket to our pod. For this we need to create service account.

- go to GCP and create service account add role **roles/storage.objectAdmin.**
- get service account key from gcp and create secrets with that key.
- give service key to pod through volume and volume mount

kubectl create secret generic gcs-service-account --from-file=key.json=path/to/service-account-key.json

```
apiVersion: batch/v1
kind: CronJob
metadata:
 name: cronjob-es-db-backup
spec:
 schedule: "31 08 * * *"
```

```
 jobTemplate:
   spec:
     template:
       spec:
         containers:
         - name: es-db-backup-script
           image: gcr.io/disearch/es_db_backup:latest  # Replace with your Docker image
           #command: ["/bin/bash", "-c"]
           #  args:
           # - /tmp/script.sh  # Replace with the path to your script inside the container
         #  volumeMounts:
         #  - name: gcs-service-account
         #    mountPath: /tmp
         #    readOnly: true
         #volumes:
         #- name: gcs-service-account
         #  secret:
         #    secretName: gcs-service-account
         restartPolicy: OnFailure
```

Verify Cronjob:

For this use kubectl command to get the cronjob. Once get add watch on kubectl command to see the pod will come on your job schedule time. Take the pod name and see the logs with logs command.

- kubectl get cronjob
- kubectl get pods --watch or watch kubectl get pods
- kubectl logs pod/podname

This document outlines the steps we took to backup the ElasticSearch database and automate the process using a Kubernetes CronJob. It includes the script used, testing steps, and the deployment of the CronJob for scheduled execution.