Kubernetes GCP: Deploy HA kafka cluster on GKE with DR through terraform

Check GitHub for helpful DevOps tools:

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics

Ask Personal AI Document assistant to learn interactively (FASTER)!

1 Download PDF

2 Go to website

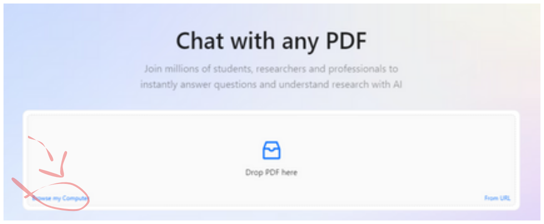
3 Browse file

4 Chat with Document

1 https://github.com/MichaelRobotics/DevOpsTools/blob/main/KubernetesGCP.pdf

Click there to go to ChatPdf website 2

3



Chat with any PDF

Join millions of students, researchers and professionals to instantly answer questions and understand research with AI

Ask questions about document! 4

Complety new to Linux and Networking?

Essential for this PDF is a thorough knowledge of networking. I highly recommend the HTB platform's networking module, which offers extensive information to help build a comprehensive understanding.

HTB - Your Cyber Performance Center

We provide a human-first platform creating and maintaining high performing cybersecurity individuals and organizations.

https://www.hackthebox.com/

What is Kubernetes?

Kubernetes is an open-source platform that automates the deployment, scaling, and management of containerized applications. It helps manage clusters of nodes running containers, ensuring efficient and reliable operation.

How Kubernetes clusters are made?

Kubernetes clusters consist of a control plane and multiple worker nodes. The control plane manages cluster operations, while worker nodes run the actual container workloads.

Why and When use Kubernetes

Kubernetes is ideal for deploying scalable, resilient, and automated containerized applications. It is used when managing multiple containers across different environments is necessary.

Example: Running a microservices-based e-commerce platform that scales up during peak hours.

System Requirements

- RAM: 2 GB per node (1 GB can work for testing but may lead to limited performance)
- 10 GB free storage
- Ubuntu

Kubernetes: Main components & packages

- **kube-apiserver:** Central management component that exposes the Kubernetes API; acts as the front-end for the cluster.
- **etcd:** Distributed key-value store for storing all cluster data, ensuring data consistency across nodes.
- **kube-scheduler:** Assigns pods to available nodes based on resource requirements and policies.
- **kube-controller-manager:** Manages core controllers that handle various functions like node status, replication, and endpoints.
- **kubelet:** Agent that runs on each node, responsible for managing pods and their containers.
- **kube-proxy:** Manages networking on each node, ensuring communication between pods and services within the cluster.

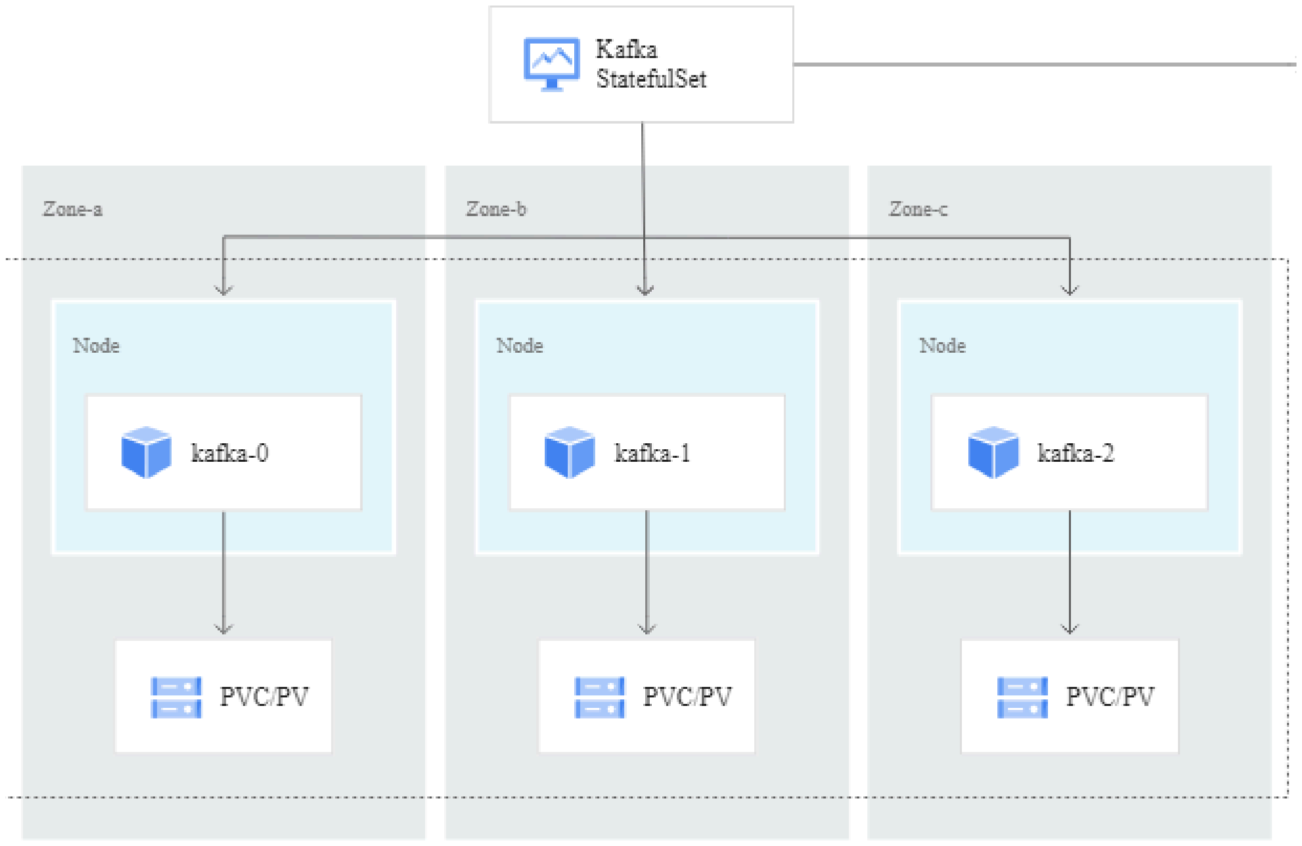
Kubernetes GCP: Project Introduction

1) What is Kafka

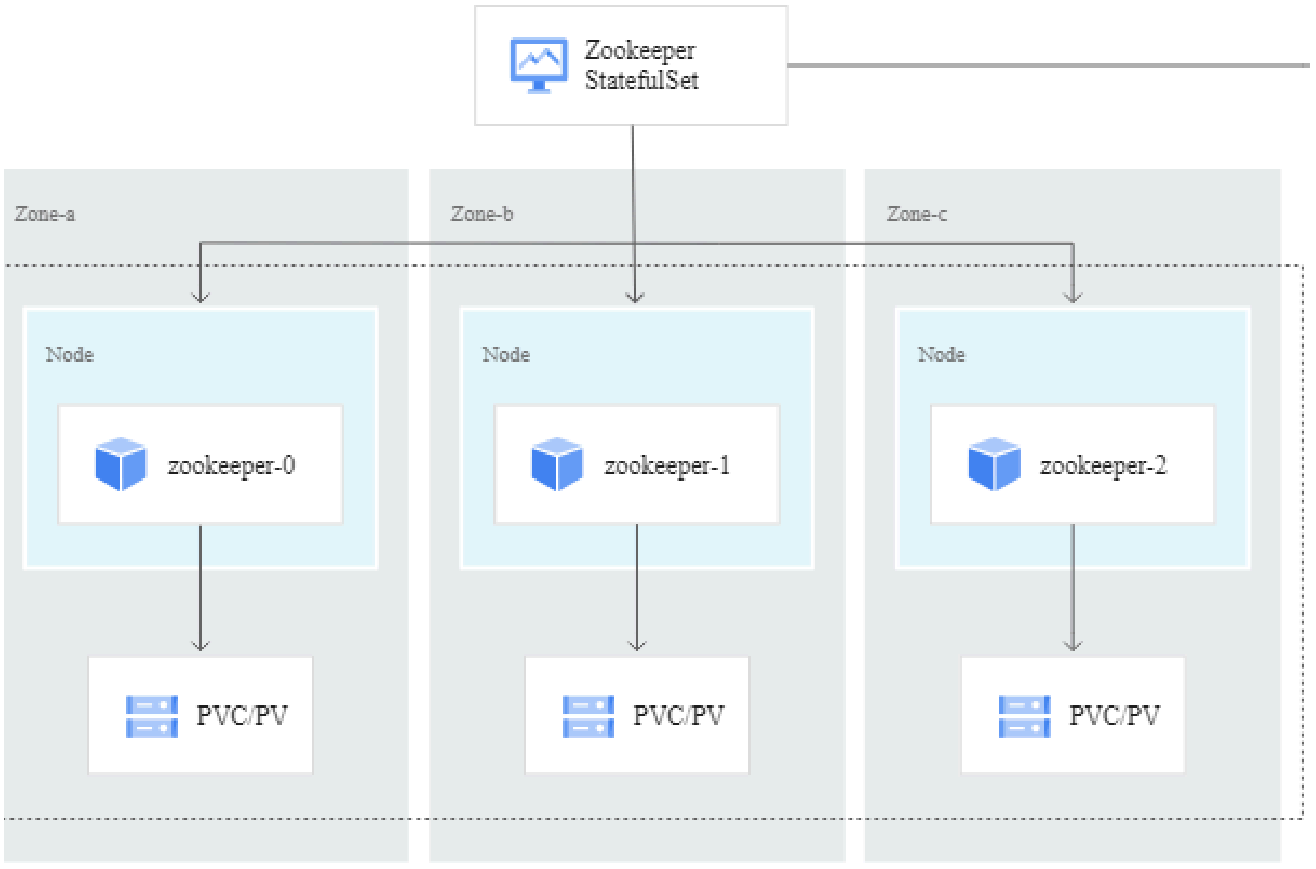
Kafka is an open-source, distributed messaging system for high-volume, real-time data streaming. It enables reliable data pipelines for processing and analysis across systems.

2) Introduction

A Kafka cluster consists of brokers managing data streams and publish-subscribe messaging for consumers. Each partition has a leader broker for reads/writes and follower brokers for replication.



you'll deploy Kafka clusters on GKE, configuring Kafka brokers and ZooKeeper as StatefulSets. To ensure high availability and disaster recovery, you'll place these StatefulSets in separate node pools and zones.



3) Costs

Following billable components of Google Cloud will be used:

- Artifact Registry
- Backup for GKE
- Compute Engine
- GKE
- Google Cloud Managed Service for Prometheus

Kubernetes GCP: Environment setup

1) Create account / login to GCP

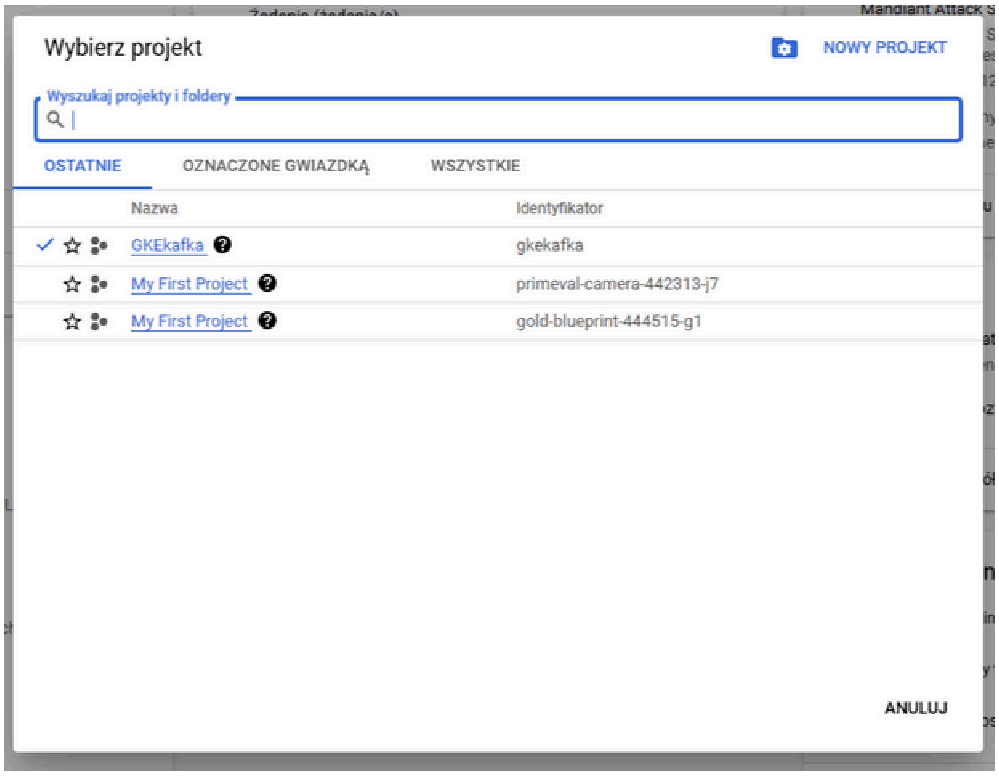
If you dont have an account already, create new and start free trial.

Start running workloads for free

Build what’s next. Better software. Faster.

https://cloud.google.com/gcp

In the Google Cloud console, on the project selector page, click Create project



Nazwa	Identyfikator
✓ GKEkafka	gkekafka
My First Project	primeval-camera-442313-j7
My First Project	gold-blueprint-444515-g1

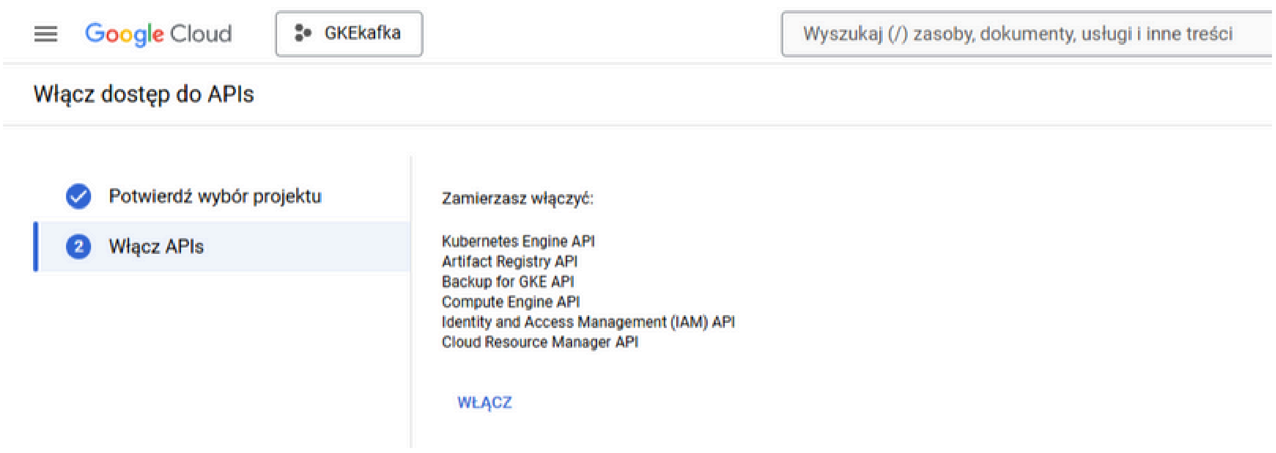
2) Add APIs

Enable the Google Kubernetes Engine, Backup for GKE, Artifact Registry, Compute Engine, and IAM APIs. Go to link below:

Start running workloads for free

Build what’s next. Better software. Faster.

https://cloud.google.com/gcp



Włącz dostęp do APIs

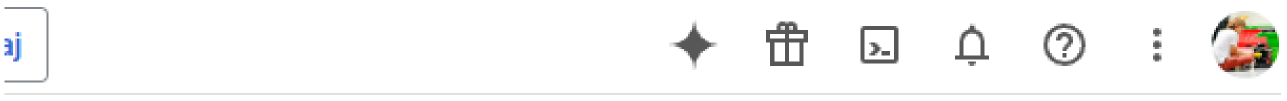
1. Potwierdź wybór projektu
2. Włącz APIs

Zamierzasz włączyć:

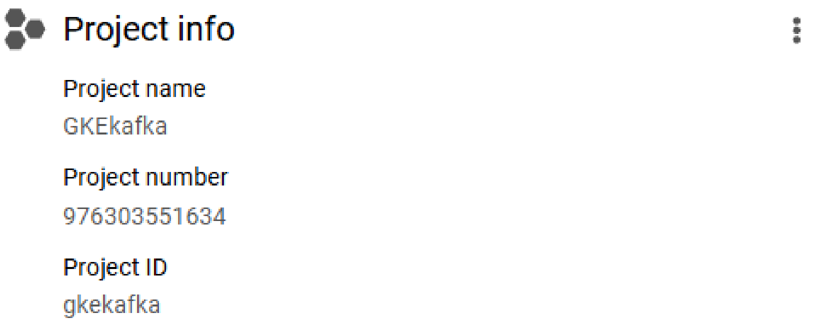
Kubernetes Engine API
Artifact Registry API
Backup for GKE API
Compute Engine API
Identity and Access Management (IAM) API
Cloud Resource Manager API

3) Set environment variables and download repo

Use cloudshell to manage resources. It comes with preinstalled software used in this tutorial.



Check your project ID Cloud Overview -> Dashboard:



Project info

Project name
GKEkafka

Project number
976303551634

Project ID
gkekafka

Assign variables

```
export PROJECT_ID=PROJECT_ID
export REGION=us-central1
export USER_IDENTIFIER=sodmasf
```

set default environment

```
gcloud config set project PROJECT_ID
```

Download repository

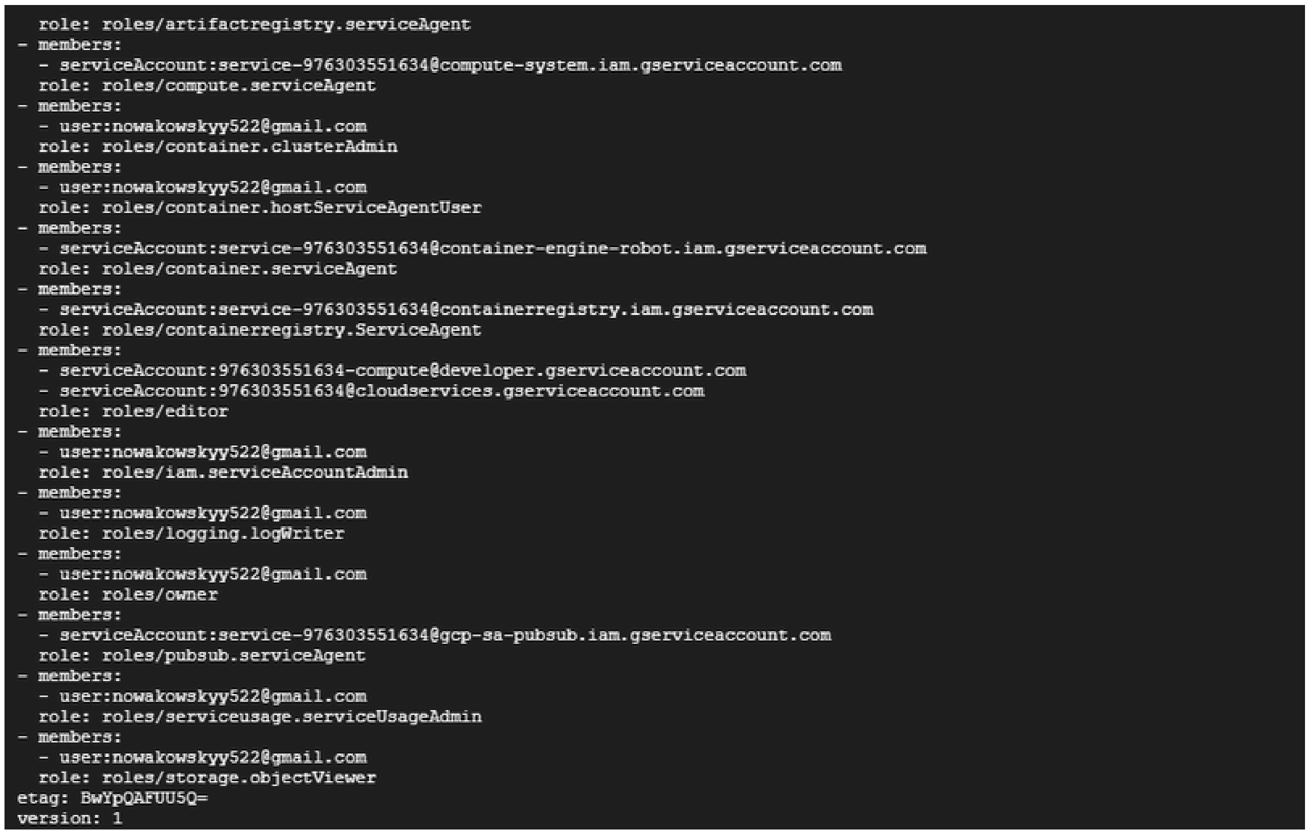
```
git clone https://github.com/GoogleCloudPlatform/kubernetes-engine-samples
cd kubernetes-engine-samples/streaming/gke-stateful-kafka
```

4) Setup roles

Grant roles to your user account.

```
gcloud projects add-iam-policy-binding PROJECT_ID --member="user:USER_IDENTIFIER" --
role=roles/storage.objectViewer

gcloud projects add-iam-policy-binding PROJECT_ID --member="user:USER_IDENTIFIER" --
role=roles/logging.logWriter

gcloud projects add-iam-policy-binding PROJECT_ID --member="user:USER_IDENTIFIER" --
role=roles/artifactregistry.admin

gcloud projects add-iam-policy-binding PROJECT_ID --member="user:USER_IDENTIFIER" --
role=roles/container.clusterAdmin
```

```
gcloud projects add-iam-policy-binding PROJECT_ID --member="user:USER_IDENTIFIER" --
role=roles/container.hostServiceAgentUser

gcloud projects add-iam-policy-binding PROJECT_ID --member="user:USER_IDENTIFIER" --
role=roles/iam.serviceAccountAdmin

gcloud projects add-iam-policy-binding PROJECT_ID --member="user:USER_IDENTIFIER" --
role=roles/serviceusage.serviceUsageAdmin

gcloud projects add-iam-policy-binding PROJECT_ID --member="user:USER_IDENTIFIER" --
role=roles/iam.serviceAccountAdmin
```



```
  role: roles/artifactregistry.serviceAgent
- members:
  - serviceAccount:service-976303551634@compute-system.iam.gserviceaccount.com
  role: roles/compute.serviceAgent
- members:
  - user:nowakowskyy522@gmail.com
  role: roles/container.clusterAdmin
- members:
  - user:nowakowskyy522@gmail.com
  role: roles/container.hostServiceAgentUser
- members:
  - serviceAccount:service-976303551634@container-engine-robot.iam.gserviceaccount.com
  role: roles/container.serviceAgent
- members:
  - serviceAccount:service-976303551634@containerregistry.iam.gserviceaccount.com
  role: roles/containerregistry.ServiceAgent
- members:
  - serviceAccount:976303551634-compute@developer.gserviceaccount.com
  - serviceAccount:976303551634@cloudservices.gserviceaccount.com
  role: roles/editor
- members:
  - user:nowakowskyy522@gmail.com
  role: roles/iam.serviceAccountAdmin
- members:
  - user:nowakowskyy522@gmail.com
  role: roles/logging.logWriter
- members:
  - user:nowakowskyy522@gmail.com
  role: roles/owner
- members:
  - serviceAccount:service-976303551634@gcp-sa-pubsub.iam.gserviceaccount.com
  role: roles/pubsub.serviceAgent
- members:
  - user:nowakowskyy522@gmail.com
  role: roles/serviceusage.serviceUsageAdmin
- members:
  - user:nowakowskyy522@gmail.com
  role: roles/storage.objectViewer
etag: BwYpQAFUU5Q=
version: 1
```

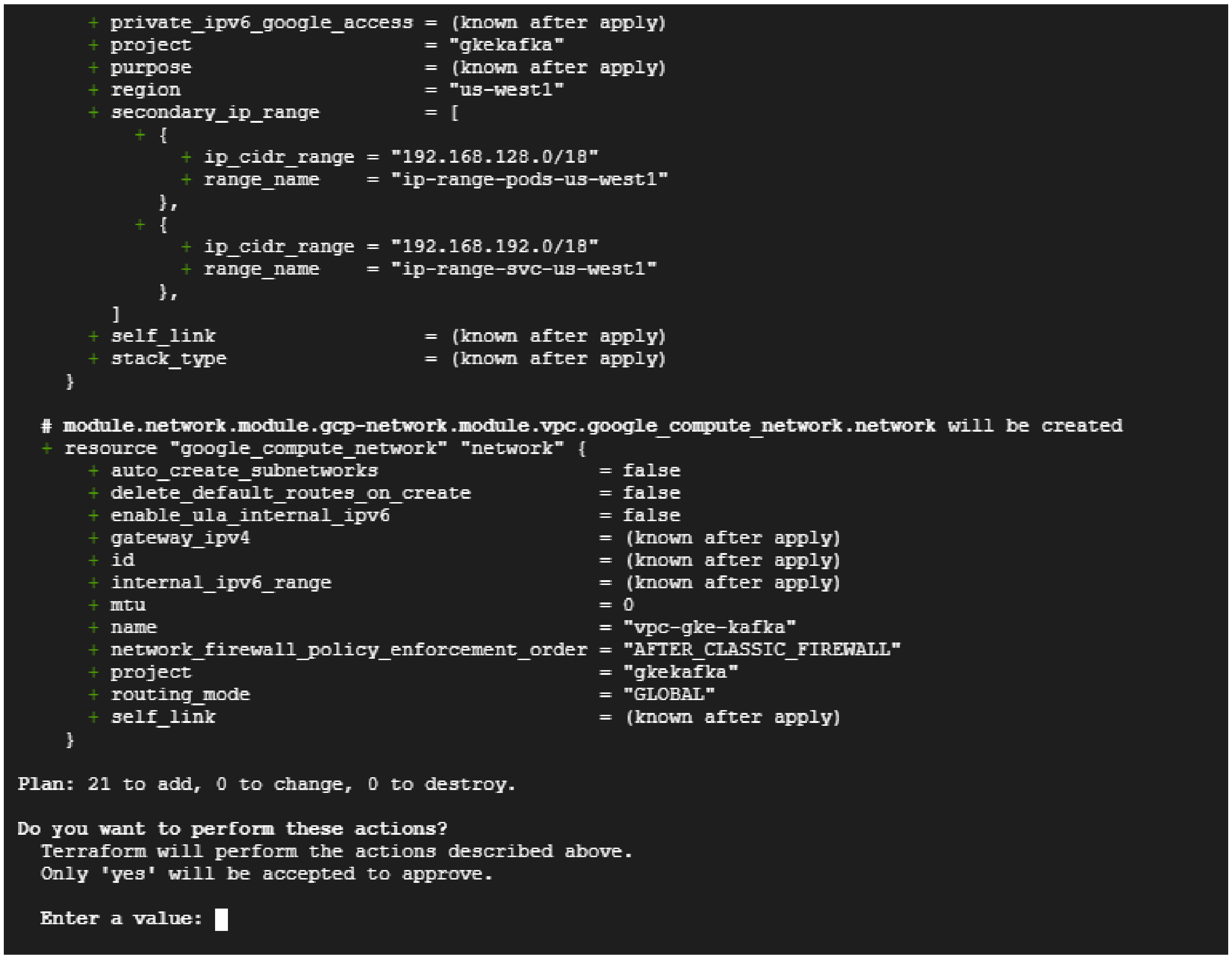
Kubernetes GCP: Create infrastructure & HA GKE clusters with Terraform

Terraform script to creates two regional GKE clusters. The primary cluster will be deployed in us-central1. In Cloud Shell, run:

```
terraform -chdir=terraform/gke-standard init
terraform -chdir=terraform/gke-standard apply -var project_id=$PROJECT_ID
```

Script creates Creates a Artifact Registry repository to store the Docker images.

VPC network, subnet for the VM's network interface and two GKE clusters.



```
      + private_ipv6_google_access = (known after apply)
      + project                    = "gkekafka"
      + purpose                    = (known after apply)
      + region                     = "us-west1"
      + secondary_ip_range         = [
          + {
              + ip_cidr_range = "192.168.128.0/18"
              + range_name    = "ip-range-pods-us-west1"
            },
          + {
              + ip_cidr_range = "192.168.192.0/18"
              + range_name    = "ip-range-svc-us-west1"
            },
        ]
      + self_link                  = (known after apply)
      + stack_type                 = (known after apply)
    }

  # module.network.module.gcp-network.module.vpc.google_compute_network.network will be created
  + resource "google_compute_network" "network" {
      + auto_create_subnetworks                   = false
      + delete_default_routes_on_create           = false
      + enable_ula_internal_ipv6                  = false
      + gateway_ipv4                              = (known after apply)
      + id                                        = (known after apply)
      + internal_ipv6_range                       = (known after apply)
      + mtu                                       = 0
      + name                                      = "vpc-gke-kafka"
      + network_firewall_policy_enforcement_order = "AFTER_CLASSIC_FIREWALL"
      + project                                   = "gkekafka"
      + routing_mode                              = "GLOBAL"
      + self_link                                 = (known after apply)
    }

Plan: 21 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value:
```

Kubernetes GCP: Deploy kafka on your cluster

deploy Kafka on GKE using a Helm chart. The operation creates the following resources:

- The Kafka and Zookeeper StatefulSets.
- A Kafka exporter deployment. The exporter gathers Kafka metrics for Prometheus consumption.

Configure Docker access.

```
gcloud auth configure-docker us-docker.pkg.dev
```

Populate Artifact Registry with the Kafka and Zookeeper images.

```
./scripts/gcr.sh bitnami/kafka 3.3.2-debian-11-r0
./scripts/gcr.sh bitnami/kafka-exporter 1.6.0-debian-11-r52
./scripts/gcr.sh bitnami/jmx-exporter 0.17.2-debian-11-r41
./scripts/gcr.sh bitnami/zookeeper 3.8.0-debian-11-r74
```

Configure kubectl command line access to the primary cluster.

```
gcloud container clusters get-credentials gke-kafka-us-central1 \
    --region=${REGION} \
    --project=${PROJECT_ID}
```

Create a namespace.

```
export NAMESPACE=kafka
kubectl create namespace $NAMESPACE
```

Install Kafka using the Helm chart version 20.0.6.

```
cd helm
../scripts/chart.sh kafka 20.0.6 && \
rm -rf Chart.lock charts && \
helm dependency update && \
helm -n kafka upgrade --install kafka . \
--set global.imageRegistry="us-docker.pkg.dev/$PROJECT_ID/main"
```

The output is similar to the following:

```
NAME: kafka
LAST DEPLOYED: Thu Feb 16 03:29:39 2023
NAMESPACE: kafka
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Verify that your Kafka replicas are running (this might take a few minutes).

```
kubectl get all -n kafka
```

The output is similar to the following:

```
NAME                  READY   STATUS    RESTARTS        AGE
pod/kafka-0            1/1     Running   2 (3m51s ago)   4m28s
pod/kafka-1            1/1     Running   3 (3m41s ago)   4m28s
pod/kafka-2            1/1     Running   2 (3m57s ago)   4m28s
pod/kafka-zookeeper-0   1/1     Running   0               4m28s
pod/kafka-zookeeper-1   1/1     Running   0               4m28s
pod/kafka-zookeeper-2   1/1     Running   0               4m28s

NAME                                TYPE        CLUSTER-IP        EXTERNAL-IP   PORT(S)
AGE
service/kafka                        ClusterIP   192.168.112.124   <none>        9092/TCP
4m29s
[...]
NAME                           READY   AGE
statefulset.apps/kafka           3/3     4m29s
statefulset.apps/kafka-zookeeper   3/3     4m29s
```

Kubernetes GCP: Test kafka

test the Kafka application and generate messages.

Create a consumer client Pod for interacting with the Kafka application.

```
kubectl run kafka-client -n kafka --rm -ti \
   --image us-docker.pkg.dev/$PROJECT_ID/main/bitnami/kafka:3.3.2-debian-11-r0 -- bash
```

Create a topic named topic1 with three partitions and a replication factor of three.

```
kafka-topics.sh \
   --create \
   --topic topic1 \
   --partitions 3  \
   --replication-factor 3 \
   --bootstrap-server kafka-headless.kafka.svc.cluster.local:9092
```

Verify that the topic partitions are replicated across all three brokers.

```
kafka-topics.sh \
   --describe \
   --topic topic1 \
   --bootstrap-server kafka-headless.kafka.svc.cluster.local:9092
```

The output is similar to the following:

```
Topic: topic1     TopicId: 1ntc4WiFS4-AUNlpr9hCmg PartitionCount: 3       ReplicationFactor: 3
Configs:
flush.ms=1000,segment.bytes=1073741824,flush.messages=10000,max.message.bytes=1000
012,retention.bytes=1073741824
      Topic: topic1    Partition: 0    Leader: 2       Replicas: 2,0,1 Isr: 2,0,1
      Topic: topic1    Partition: 1    Leader: 1       Replicas: 1,2,0 Isr: 1,2,0
      Topic: topic1    Partition: 2    Leader: 0       Replicas: 0,1,2 Isr: 0,1,2
```

topic1 has three partitions, each with a unique leader and replica set. Kafka's partitioning distributes data across brokers for scalability and fault tolerance. With a replication factor of three, each partition has three replicas, ensuring data availability even if one or two brokers fail.

Run the following command to generate message numbers in bulk into topic1.

```
ALLOW_PLAINTEXT_LISTENER=yes
for x in $(seq 0 200); do
  echo "$x: Message number $x"
done | kafka-console-producer.sh \
    --topic topic1 \
    --bootstrap-server kafka-headless.kafka.svc.cluster.local:9092 \
    --property parse.key=true \
    --property key.separator=":"
```

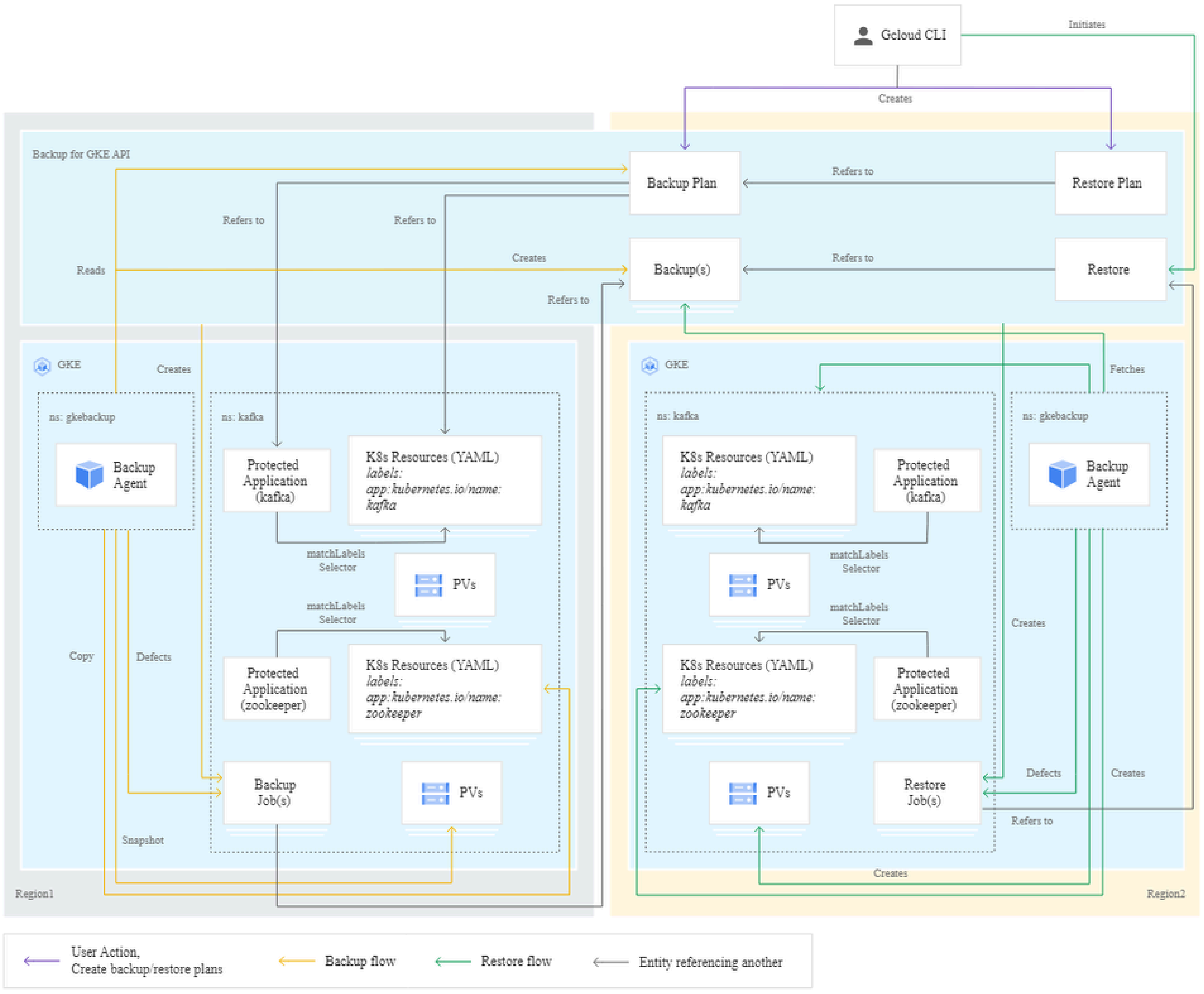
Run the following command to consume topic1 from all partitions.

```
kafka-console-consumer.sh \
    --bootstrap-server kafka.kafka.svc.cluster.local:9092 \
    --topic topic1 \
    --property print.key=true \
    --property key.separator=" : " \
    --from-beginning;
```

Kubernetes GCP: Prepare DR solution

Plan for backup is as follows:

1. Take backup of gke-kafka-us-central1
2. Restore the backup into gke-kafka-us-west1
3. perform the backup and restore operation at the application scope, using the ProtectedApplication Custom Resource.



Set up the environment variables.

```
export BACKUP_PLAN_NAME=kafka-protected-app
export BACKUP_NAME=protected-app-backup-1
export RESTORE_PLAN_NAME=kafka-protected-app
export RESTORE_NAME=protected-app-restore-1
export REGION=us-central1
export DR_REGION=us-west1
export CLUSTER_NAME=gke-kafka-$REGION
export DR_CLUSTER_NAME=gke-kafka-$DR_REGION
```

Verify the cluster is in a RUNNING state.

```
gcloud container clusters describe $CLUSTER_NAME --region us-central1 --format='value(status)'
```

Create a Backup Plan.

```
gcloud beta container backup-restore backup-plans create $BACKUP_PLAN_NAME \
   --project=$PROJECT_ID \
   --location=$DR_REGION \
   --cluster=projects/$PROJECT_ID/locations/$REGION/clusters/$CLUSTER_NAME \
   --selected-applications=kafka/kafka,kafka/zookeeper \
   --include-secrets \
   --include-volume-data \
   --cron-schedule="0 3 * * *" \
   --backup-retain-days=7 \
   --backup-delete-lock-days=0
```

Manually create a Backup. While scheduled backups are typically governed by the cron-schedule in the backup plan, the following example shows how you can initiate a one-time backup operation.

```
gcloud beta container backup-restore backups create $BACKUP_NAME \
   --project=$PROJECT_ID \
   --location=$DR_REGION \
   --backup-plan=$BACKUP_PLAN_NAME \
   --wait-for-completion
```

Create a Restore Plan.

```
gcloud beta container backup-restore restore-plans create $RESTORE_PLAN_NAME \
    --project=$PROJECT_ID \
    --location=$DR_REGION \
    --backup-
plan=projects/$PROJECT_ID/locations/$DR_REGION/backupPlans/$BACKUP_PLAN_NAME \
    --cluster=projects/$PROJECT_ID/locations/$DR_REGION/clusters/$DR_CLUSTER_NAME \
    --cluster-resource-conflict-policy=use-existing-version \
    --namespaced-resource-restore-mode=delete-and-restore \
    --volume-data-restore-policy=restore-volume-data-from-backup \
    --selected-applications=kafka/kafka,kafka/zookeeper \
    --cluster-resource-scope-selected-group-kinds="storage.k8s.io/StorageClass"
```

Manually restore from a Backup.

```
gcloud beta container backup-restore restores create $RESTORE_NAME \
    --project=$PROJECT_ID \
    --location=$DR_REGION \
    --restore-plan=$RESTORE_PLAN_NAME \
    --backup=projects/$PROJECT_ID/locations/$DR_REGION/backupPlans/
$BACKUP_PLAN_NAME/backups/$BACKUP_NAME
```

Watch the restored application come up in the backup cluster. It may take a few minutes for all the Pods to be running and ready.

```
gcloud container clusters get-credentials gke-kafka-us-west1 \
    --region us-west1
kubectl get pod -n kafka --watch
```

Validate that previous topics can be fetched by a consumer.

```
kubectl run kafka-client -n kafka --rm -ti \
    --image us-docker.pkg.dev/$PROJECT_ID/main/bitnami/kafka:3.4.0 -- bash

kafka-console-consumer.sh \
    --bootstrap-server kafka.kafka.svc.cluster.local:9092 \
    --topic topic1 \
    --property print.key=true \
    --property key.separator=" : " \
    --from-beginning;
```

The output is similar to the following:

```
192 :  Message number 192
193 :  Message number 193
197 :  Message number 197
200 :  Message number 200
Processed a total of 201 messages
```

Kubernetes GCP: Clean up

To avoid incurring charges to your Google Cloud account for the resources used in this tutorial, either delete the project that contains the resources, or keep the project and delete the individual resources.

Delete the project

The easiest way to avoid billing is to delete the project.

Delete a Google Cloud project:

```
gcloud projects delete PROJECT_ID
```

common troubleshooting

1) Terraform Apply Fails with IAM Errors

Cause: Missing permissions for the Terraform service account.
Solution: Assign roles like roles/container.admin and roles/compute.admin. Validate with gcloud projects get-iam-policy and update bindings with gcloud projects add-iam-policy-binding.

2) Kafka Pods Stuck in Pending State

Cause: Insufficient node resources or misconfigured node pool.
Solution: Check pod events with kubectl describe pod. Scale the node pool or adjust resource requests/limits in the StatefulSet.

3) PersistentVolumeClaims (PVCs) Not Binding

Cause: Misconfigured StorageClass or insufficient storage.
Solution: Confirm StorageClass with kubectl get pvc. Verify GCP storage capacity and adjust PersistentVolumes if needed.

4) Check my Kubernetes Troubleshooting series:

Michael Robotics
Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics

Learn more about Kubernetes

Check Kubernetes and piyushsachdeva - great docs!

Setup a Multi Node Kubernetes Cluster

kubeadm is a tool to bootstrap the Kubernetes cluster

https://github.com/piyushsachdeva/CKA-2024/tree/main/Resources/Day27

Kubernetes Documentation

This section lists the different ways to set up and run Kubernetes

https://kubernetes.io/docs/setup/

Share, comment, DM and check GitHub for scripts & playbooks created to automate process.

Check my GitHub

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics

PS.

If you need a playbook or bash script to manage KVM on a specific Linux distribution, feel free to ask me in the comments or send a direct message!