Deploying to Google Kubernetes Engine

In this article

You can deploy to Google Kubernetes Engine as part of your continuous deployment (CD) workflows.

Note: GitHub-hosted runners are not currently supported on GitHub Enterprise Server. You can see more information about planned future support on the GitHub public roadmap.

Introduction

This guide explains how to use GitHub Actions to build a containerized application, push it to Google Container Registry (GCR), and deploy it to Google Kubernetes Engine (GKE) when there is a push to the `main` branch.

GKE is a managed Kubernetes cluster service from Google Cloud that can host your containerized workloads in the cloud or in your own datacenter. For more information, see Google Kubernetes Engine.

Prerequisites

Before you proceed with creating the workflow, you will need to complete the following steps for your Kubernetes project. This guide assumes the root of your project already has a `Dockerfile` and a Kubernetes Deployment configuration file.

Creating a GKE cluster

To create the GKE cluster, you will first need to authenticate using the `gcloud` CLI. For more information on this step, see the following articles:

- `gcloud auth login`
- `gcloud` CLI
- `gcloud` CLI and Cloud SDK

For example:

Bash

```
$ gcloud container clusters create $GKE_CLUSTER \
        --project=$GKE_PROJECT \
        --zone=$GKE_ZONE
```

Enabling the APIs

Enable the Kubernetes Engine and Container Registry APIs. For example:

Bash

```
$ gcloud services enable \
        containerregistry.googleapis.com \
        container.googleapis.com
```

Configuring a service account and storing its credentials

This procedure demonstrates how to create the service account for your GKE integration. It explains how to create the account, add roles to it, retrieve its keys, and store them as a base64-encoded encrypted repository secret named `GKE_SA_KEY`.

1. Create a new service account:

   ```
   $ gcloud iam service-accounts create $SA_NAME
   ```

2. Retrieve the email address of the service account you just created:

   ```
   $ gcloud iam service-accounts list
   ```

3. Add roles to the service account. Note: Apply more restrictive roles to suit your requirements.

   ```
   $ gcloud projects add-iam-policy-binding $GKE_PROJECT \
    --member=serviceAccount:$SA_EMAIL \
    --role=roles/container.admin
   $ gcloud projects add-iam-policy-binding $GKE_PROJECT \
    --member=serviceAccount:$SA_EMAIL \
    --role=roles/storage.admin
   $ gcloud projects add-iam-policy-binding $GKE_PROJECT \
    --member=serviceAccount:$SA_EMAIL \
    --role=roles/container.clusterViewer
   ```

4. Download the JSON keyfile for the service account:

   ```
   $ gcloud iam service-accounts keys create key.json --iam-account=$SA_EMAIL
   ```

5. Store the service account key as a secret named `GKE_SA_KEY`:

   ```
   $ export GKE_SA_KEY=$(cat key.json | base64)
   ```

   For more information about how to store a secret, see "Encrypted secrets."

Storing your project name

Store the name of your project as a secret named `GKE_PROJECT`. For more information about how to store a secret, see "Encrypted secrets."

(Optional) Configuring kustomize

Kustomize is an optional tool used for managing YAML specs. After creating a `kustomization` file, the workflow below can be used to dynamically set fields of the image and pipe in the result to `kubectl`. For more information, see kustomize usage.

(Optional) Configure a deployment environment

Environments are used to describe a general deployment target like `production`, `staging`, or `development`. When a GitHub Actions workflow deploys to an environment, the environment is displayed on the main page of the repository. You can use environments to require approval for a job to proceed, restrict which branches can trigger a workflow, or limit access to secrets. For more information about creating environments, see "Using environments for deployment."

Creating the workflow

Once you've completed the prerequisites, you can proceed with creating the workflow.

The following example workflow demonstrates how to build a container image and push it to GCR. It then uses the Kubernetes tools (such as `kubectl` and `kustomize`) to pull the image into the cluster deployment.

Under the `env` key, change the value of `GKE_CLUSTER` to the name of your cluster, `GKE_ZONE` to your cluster zone, `DEPLOYMENT_NAME` to the name of your deployment, and `IMAGE` to the name of your image.

If you configured a deployment environment, change the value of `environment` to be the name of your environment. If you did not configure an environment, delete the `environment` key.

YAML

```yaml
# This workflow uses actions that are not certified by GitHub.
# They are provided by a third-party and are governed by
# separate terms of service, privacy policy, and support
# documentation.

# GitHub recommends pinning actions to a commit SHA.
# To get a newer version, you will need to update the SHA.
# You can also reference a tag or branch, but the action may change without warning

name: Build and Deploy to GKE

on:
  push:
    branches:
      - main

env:
  PROJECT_ID: ${{ secrets.GKE_PROJECT }}
  GKE_CLUSTER: cluster-1    # Add your cluster name here.
  GKE_ZONE: us-central1-c   # Add your cluster zone here.
  DEPLOYMENT_NAME: gke-test # Add your deployment name here.
  IMAGE: static-site

jobs:
```

```yaml
  setup-build-publish-deploy:
    name: Setup, Build, Publish, and Deploy
    runs-on: ubuntu-latest
    environment: production

    steps:
    - name: Checkout
      uses: actions/checkout@v2

    # Setup gcloud CLI
    - uses: google-github-actions/setup-gcloud@1bee7de035d65ec5da40a31f8589e240eba8
      with:
        service_account_key: ${{ secrets.GKE_SA_KEY }}
        project_id: ${{ secrets.GKE_PROJECT }}

    # Configure Docker to use the gcloud command-line tool as a credential
    # helper for authentication
    - run: |-
        gcloud --quiet auth configure-docker

    # Get the GKE credentials so we can deploy to the cluster
    - uses: google-github-actions/get-gke-credentials@db150f2cc60d1716e61922b832eae
      with:
        cluster_name: ${{ env.GKE_CLUSTER }}
        location: ${{ env.GKE_ZONE }}
        credentials: ${{ secrets.GKE_SA_KEY }}

    # Build the Docker image
    - name: Build
      run: |-
        docker build \
          --tag "gcr.io/$PROJECT_ID/$IMAGE:$GITHUB_SHA" \
          --build-arg GITHUB_SHA="$GITHUB_SHA" \
          --build-arg GITHUB_REF="$GITHUB_REF" \
          .

    # Push the Docker image to Google Container Registry
    - name: Publish
      run: |-
        docker push "gcr.io/$PROJECT_ID/$IMAGE:$GITHUB_SHA"

    # Set up kustomize
    - name: Set up Kustomize
      run: |-
        curl -sfLo kustomize https://github.com/kubernetes-sigs/kustomize/releases/
        chmod u+x ./kustomize

    # Deploy the Docker image to the GKE cluster
    - name: Deploy
      run: |-
        ./kustomize edit set image gcr.io/PROJECT_ID/IMAGE:TAG=gcr.io/$PROJECT_ID/$I
        ./kustomize build . | kubectl apply -f -
        kubectl rollout status deployment/$DEPLOYMENT_NAME
        kubectl get services -o wide
```

Additional resources

For more information on the tools used in these examples, see the following documentation:

- For the full starter workflow, see the "Build and Deploy to GKE" workflow.
- The Kubernetes YAML customization engine: Kustomize.
- "Deploying a containerized web application" in the Google Kubernetes Engine documentation.