CI/CD with GKE and Google Cloud Build

Nilesh Jayanandana
Follow
Feb 12, 2019 · 5 min read

204

Google Cloud Build is a really powerful tool. You can configure your entire CI/CD pipeline with it. In this article, I'm going to guide you through on creating a simple CI/CD pipeline with Google Cloud Build and GKE.

Pre-requisites

You need to have a GCP account. If you don't sign up here and subscribe for the free $300 Google gives you to try out the Google Cloud Platform. Once you are logged in, you need to create a Kubernetes Cluster, which can be created here under Kubernetes engine in the main menu.

Dockerize the application

To get started, first, you need to Dockerize your application. I'm going to assume that the Dockerfile and the context will be in your root directory of the application. When it's done, your application should look like this. In the sample Node application I'm going to use here, it would look like this.

```
node_modules/
index.js
package.json
package-lock.json
Dockerfile
```

Make sure the Dockerfile works as expected by building it and running it locally. You can bind your ports and test it in whatever way you like. Here are the basic commands.

```
docker build -t <app-name> .
docker run -p <host-port>:<container-port> <app-name>
```

Write the Kubernetes Configs

Okay. Now you are done with the Dockerfile, let's get to the good stuff. Create a folder called **k8s.** We are going to add all the Kubernetes Yaml files into this folder. Let's add a Service and a Deployment.

Create a file called deployment.yaml inside the k8s folder.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    run: <app-name>
  name: <app-name>
spec:
  replicas: 2
  selector:
    matchLabels:
      run: <app-name>
  template:
    metadata:
      labels:
        run: <app-name>
    spec:
      containers:
      - image: gcr.io/<google-project-id>/<app-name>:latest
        name: <app-name>
        ports:
        - containerPort: <container-port>
```

Now let's create the service for this in service.yaml inside the k8s folder. I'm going to use the LoadBalancer type service for this demo. But you are free to use whatever you like.

```
kind: Service
apiVersion: v1
metadata:
  name: <app-name>
spec:
  selector:
     run: <app-name>
  ports:
  - protocol: TCP
    port: 80
    targetPort: <container-port>
  type: LoadBalancer
```

Creating Cloudbuild Config

Okay. Now with this configured, we have to go to the next step of writing the Google Cloud Build Configuration. To do this, create a new file called cloudbuild.yaml in the root directory of your application and add the following lines.

```
steps:
#step 1
- name: 'gcr.io/cloud-builders/docker'
  entrypoint: 'bash'
  args: [
   '-c',
   'docker pull gcr.io/$PROJECT_ID/<app-name>:latest || exit 0'
  ]
#step 2
- name: gcr.io/cloud-builders/docker
  args: [
   'build',
   '-t',
   'gcr.io/$PROJECT_ID/<app-name>:$TAG_NAME',
   '-t',
   'gcr.io/$PROJECT_ID/<app-name>:latest',
   '.'
  ]
#step 3
- name: 'gcr.io/cloud-builders/kubectl'
  args: ['apply', '-f', 'k8s/']
  env:
  - 'CLOUDSDK_COMPUTE_ZONE=<cluster-zone>'
  - 'CLOUDSDK_CONTAINER_CLUSTER=<cluster-name>'
#step 4
- name: 'gcr.io/cloud-builders/kubectl'
  args: [
   'set',
   'image',
   'deployment',
   '<app-name>',
   '<app-name>=gcr.io/$PROJECT_ID/<app-name>:$TAG_NAME'
  ]
  env:
  - 'CLOUDSDK_COMPUTE_ZONE=<cluster-zone>'
  - 'CLOUDSDK_CONTAINER_CLUSTER=<cluster-name>'
# push images to Google Container Registry with tags
images: [
   'gcr.io/$PROJECT_ID/<app-name>:$TAG_NAME',
   'gcr.io/$PROJECT_ID/<app-name>:latest'
  ]
```

This cloud build is configured to initiate a CI/CD pipeline targeting a Tag Push Trigger. If you want Branch push, you will have to replace **$TAG_NAME** with
$BRANCH_NAME-$COMMIT_SHA.

I will explain what happens in each of the above steps.
Step 1, we are trying to pull the latest existing image of the application we are trying to build so that our build will be faster as docker uses the cached layers of the old images to build new images. The reason for adding *|| exit 0* is in case the docker pull returns an error (When running this build for the first time there will be no latest image to pull from the repository), the entire pipeline would fail. That is why *|| exit 0* is added to ignore and continue with the build even if an error occurs in that step.

Step 2, we build the docker image of our application.

Step 3, we apply all the configuration yamls that exist in the k8s/ folder of our application. Kubectl is a really powerful tool and it will automatically create or update missing configs and resources in the cluster according to the yamls you have provided in the k8s folder. **<cluster-zone>** is the zone of your kubernetes cluster. You cluster maybe regional, but still if you go to the Kubernetes Engine in Google, you would see the default zone name of your cluster. You have to add that in. **<cluster-name>** is the name of your cluster which you gave when you created the cluster.

Step 4, Update the app deployment image in GKE with the latest version of the image you built.

Set up IAM

Now on a coding perspective, you are all done. Now we need to configure a couple more resources to get this pipeline working in GCP. First is setting up IAM for cloud builder so that it has permissions to deploy the images in GKE.
To do this, navigate to IAM & Admin →IAM on the side menu.

Here, you would see an entry in the format,

```
<project-number>@cloudbuild.gserviceaccount.com
```

with the role, *Cloud Build Service Account.* Edit the role and add *Kubernetes Engine Admin* role to it as well.

Set up a Build Trigger

Okay, now we have to connect the dots. We are going to add a build trigger to initiate our pipeline. To do this, you have to navigate to Cloud Build → Triggers. Here, click the button *Add Trigger* and you will be given a choice to select from either Github or Bitbucket. (Sorry Gitlab fans, it's not supported at the moment. If you want Gitlab, try Platformer Cloud which provides a PaaS solution on top of Kubernetes with the underlying infrastructure running inside GCP).

Once you select your repository, You will be navigated to the build config page. Please fill the details as shown in the screenshot below and create the trigger.

All good! now everything is configured and you can do a Git Tag push in your repository and the pipeline will jump to action.

Bonus Content: What about tests?

You can configure Integration tests too in the cloud build yaml. Just add another step with your testing configurations. Or, better yet, I recommend you use a multi stage docker file, where you run your integration tests as a stage in your Docker build.

I hope this helped you. Clap for more. Cheers!

Kubernetes · Google Cloud Platform · Google Kubernetes Engine · Continuous Deployment · Cloud

204 claps

WRITTEN BY

Nilesh Jayanandana

Follow

Software Architect at Platformer

Platformer Cloud

Follow

Platformer.com Blog | Technical stories written by our team, partners and invited authors on Cloud, Containers, Kubernetes, Serverless, etc

See responses (7)