Artigo
Mikhail Khomenko · Nov. 19, 2021 15min de leitura

Open Exchange

Automatizando a criação do GKE em compilações do CircleCI

Da última vez, lançamos uma aplicação IRIS no Google Cloud usando seu serviço GKE.

E, embora criar um cluster manualmente (ou por meio do gcloud) seja fácil, a abordagem de Infraestrutura como Código (IaC) moderna recomenda que a descrição do cluster Kubernetes também seja armazenada no repositório como código. Como escrever este código é determinado pela ferramenta que é usada para IaC.

No caso do Google Cloud, existem várias opções, entre elas o Deployment Manager e o Terraform. As opiniões estão divididas quanto o que é melhor: se você quiser saber mais, leia este tópico no Reddit Opiniões sobre Terraform vs. Deployment Manager? e o artigo no Medium Comparando o GCP Deployment Manager e o Terraform.

Para este artigo, escolheremos o Terraform, já que ele está menos vinculado a um fornecedor específico e você pode usar seu IaC com diferentes provedores em nuvem.

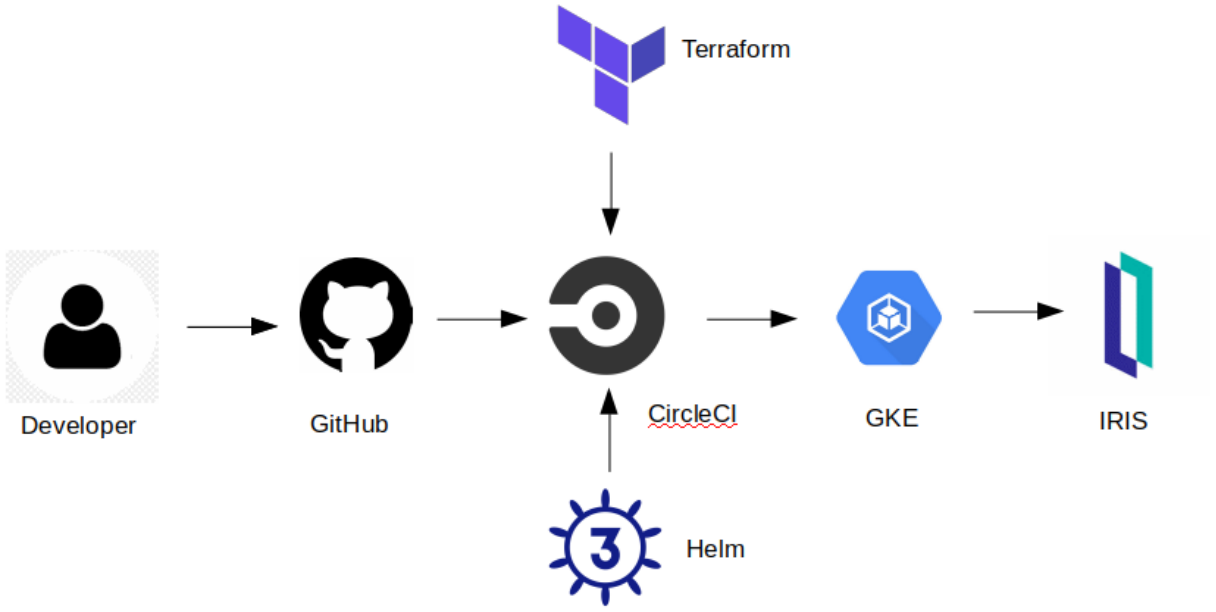
Suporemos que você leu o artigo anterior e já tem uma conta do Google, e que criou um projeto chamado " Desenvolvimento " , como no artigo anterior. Neste artigo, seu ID é mostrado como . Nos exemplos abaixo, altere-o para o ID de seu próprio projeto.

Lembre-se de que o Google não é gratuito, embora tenha um nível gratuito. Certifique-se de controlar suas despesas.

Também presumiremos que você já bifurcou o repositório original. Chamaremos essa bifurcação (fork) de " my-objectscript-rest-docker-template " e nos referiremos ao seu diretório raiz como"" ao longo deste artigo.

Todos os exemplos de código são armazenados neste repositório para simplificar a cópia e a colagem.

O diagrama a seguir descreve todo o processo de implantação em uma imagem:



Então, vamos instalar a versão mais recente do Terraform no momento desta postagem:
$ terraform version
Terraform v0.12.17

A versão é importante aqui, pois muitos exemplos na Internet usam versões anteriores, e a 0.12 trouxe muitas mudanças.

Queremos que o Terraform execute certas ações (use certas APIs) em nossa conta do GCP. Para ativar isso, crie uma conta de serviço com o nome 'terraform' e ative a API do Kubernetes Engine. Não se preocupe sobre como vamos conseguir isso — basta continuar lendo e suas perguntas serão respondidas.

Vamos tentar um exemplo com o utilitário gcloud, embora também possamos usar o console web.

Usaremos alguns comandos diferentes nos exemplos a seguir. Consulte os tópicos da documentação a seguir para obter mais detalhes sobre esses comandos e recursos.

Como criar contas de serviço IAM no gcloud
Como atribuir papéis a uma conta de serviço para recursos específicos
Como criar chaves de conta de serviço IAM no gcloud
Como ativar uma API no projeto do Google Cloud
Agora vamos analisar o exemplo.
$ gcloud init

Como trabalhamos com o gcloud no artigo anterior, não discutiremos todos os detalhes de configuração aqui. Para este exemplo, execute os seguintes comandos:
$ cd
$ mkdir terraform; cd terraform
$ gcloud iam service-accounts create terraform --description "Terraform" --display-name "terraform"

Agora, vamos adicionar alguns papéis à conta de serviço do terraform além de " Administrador do Kubernetes

Engine ” (container.admin). Essas funções serão úteis para nós no futuro.

```
$ gcloud projects add-iam-policy-binding \
  --member serviceAccount:terraform@.iam.gserviceaccount.com \
  --role roles/container.admin
$ gcloud projects add-iam-policy-binding \
  --member serviceAccount:terraform@.iam.gserviceaccount.com \
  --role roles/iam.serviceAccountUser

$ gcloud projects add-iam-policy-binding \
  --member serviceAccount:terraform@.iam.gserviceaccount.com \
  --role roles/compute.viewer

$ gcloud projects add-iam-policy-binding \
  --member serviceAccount:terraform@.iam.gserviceaccount.com \
  --role roles/storage.admin

$ gcloud iam service-accounts keys create account.json \
--iam-account terraform@.iam.gserviceaccount.com
```

Observe que a última entrada cria o seu arquivo *account.json*. Certifique-se de manter este arquivo em segredo.

```
$ gcloud projects list
$ gcloud config set project
$ gcloud services list --available | grep 'Kubernetes Engine'
$ gcloud services enable container.googleapis.com
$ gcloud services list --enabled | grep 'Kubernetes Engine'
container.googleapis.com Kubernetes Engine API
```

A seguir, vamos descrever o cluster GKE na linguagem HCL do Terraform. Observe que usamos vários placeholders aqui; substitua-os por seus valores:

Placeholder	Significado	Exemplo
	ID do projeto do GCP	possible-symbol-254507
	Armazenamento para estado/bloqueio do Terraform - deve ser único	circleci-gke-terraform-demo
	Região onde os recursos serão criados	europe-west1
	Zona onde os recursos serão criados	europe-west1-b
	Nome do cluster GKE	dev-cluster
	Nome do pool de nós de trabalho do GKE	dev-cluster-node-pool

Aqui está a configuração HCL para o cluster na prática:

```
$ cat main.tf
terraform {
  requiredversion = ">= 0.12"
  backend "gcs" {
  bucket = ""
  prefix = "terraform/state"
  credentials = "account.json"
  }
}
provider "google" {
```

```
  credentials = file("account.json")
  project = ""
  region = ""
}

resource "google_container_cluster" "gke-cluster" {
  name = ""
  location = ""
  remove_default_node_pool = true
  # No cluster regional (localização é região, não zona)
  # este é um número de nós por zona
  initial_node_count = 1
}

resource "google_container_node_pool" "preemptible_node_pool" {
  name = ""
  location = ""
  cluster = google_container_cluster.gke-cluster.name
  # No cluster regional (localização é região, não zona)
  # este é um número de nós por zona
  node_count = 1

  node_config {
  preemptible = true
  machine_type = "n1-standard-1"
  oauth_scopes = [
  "storage-ro",
  "logging-write",
  "monitoring"
  ]
  }
}
```

Para garantir que o código HCL esteja no formato adequado, o Terraform fornece um comando de formatação útil que você pode usar:

```
  $ terraform fmt
```

O fragmento de código (snippet) mostrado acima indica que os recursos criados serão fornecidos pelo Google e os próprios recursos são google_container_cluster e google_container_node_pool, que designamos como preemptivos para economia de custos. Também optamos por criar nosso próprio pool em vez de usar o padrão.

Vamos nos concentrar brevemente na seguinte configuração:

```
  terraform {
  required_version = ">= 0.12"
  backend "gcs" {
  Bucket = ""
  Prefix = "terraform/state"
  credentials = "account.json"
  }
}
```

O Terraform grava tudo o que é feito no arquivo de status e usa esse arquivo para outro trabalho. Para um

compartilhamento conveniente, é melhor armazenar este arquivo em algum lugar remoto. Um lugar típico é um Google Bucket.

Vamos criar este bucket. Use o nome do seu bucket em vez do placeholder . Antes da criação do bucket, vamos verificar se está disponível, pois deve ser único em todo o GCP:

```
$ gsutil acl get gs://
```

Boa resposta:

```
BucketNotFoundException: 404 gs:// bucket does not exist
```

A resposta ocupado "Busy" significa que você deve escolher outro nome:

```
AccessDeniedException: 403 does not have storage.buckets.get access to
```

Também vamos habilitar o controle de versão, como o Terraform recomenda.

```
$ gsutil mb -l EU gs://
$ gsutil versioning get gs://
gs://: Suspended

$ gsutil versioning set on gs://

$ gsutil versioning get gs://
gs://: Enabled
```

O Terraform é modular e precisa adicionar um plugin de provedor do Google para criar algo no GCP. Usamos o seguinte comando para fazer isso:

```
$ terraform init
```

Vejamos o que o Terraform fará para criar um cluster do GKE:

```
$ terraform plan -out dev-cluster.plan
```

A saída do comando inclui detalhes do plano. Se você não tem objeções, vamos implementar este plano:

```
$ terraform apply dev-cluster.plan
```

A propósito, para excluir os recursos criados pelo Terraform, execute este comando a partir do diretório /terraform/:

```
$ terraform destroy -auto-approve
```

Vamos deixar o cluster como está por um tempo e seguir em frente. Mas primeiro observe que não queremos colocar tudo no repositório, então vamos adicionar vários arquivos às exceções:

```
$ cat /.gitignore
.DSStore
terraform/.terraform/
terraform/*.plan
```

terraform/*.json

Usando Helm

No artigo anterior, armazenamos os manifestos do Kubernetes como arquivos YAML no diretório /k8s/, que enviamos ao cluster usando o comando "kubectl apply".

Desta vez, tentaremos uma abordagem diferente: usando o gerenciador de pacotes Helm do Kubernetes, que foi atualizado recentemente para a versão 3. Use a versão 3 ou posterior porque a versão 2 tinha problemas de segurança do lado do Kubernetes (veja Executando o Helm na produção: melhores práticas de Segurança para mais detalhes). Primeiro, empacotaremos os manifestos Kubernetes de nosso diretório k8s/ em um pacote Helm, que é conhecido como chart. Um chart Helm instalado no Kubernetes é chamado de release. Em uma configuração mínima, um chart consistirá em vários arquivos:

```
 $ mkdir /helm; cd /helm
$ tree /helm/
helm/
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── helpers.tpl
│   └── service.yaml
└── values.yaml
```

Seu propósito está bem descrito no site oficial. As práticas recomendadas para criar seus próprios charts são descritas no Guia de Melhores Práticas do Chart na documentação do Helm.

Esta é a aparência do conteúdo de nossos arquivos:

```
 $ cat Chart.yaml
apiVersion: v2
name: iris-rest
version: 0.1.0
appVersion: 1.0.3
description: Helm for ObjectScript-REST-Docker-template application
sources:
- https://github.com/intersystems-community/objectscript-rest-docker-template
- https://github.com/intersystems-community/gke-terraform-circleci-objects...
```

```
 $ cat templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ template "iris-rest.name" . }}
  labels:
  app: {{ template "iris-rest.name" . }}
  chart: {{ template "iris-rest.chart" . }}
  release: {{ .Release.Name }}
  heritage: {{ .Release.Service }}
spec:
  replicas: {{ .Values.replicaCount }}
  strategy:
  {{- .Values.strategy | nindent 4 }}
  selector:
  matchLabels:
  app: {{ template "iris-rest.name" . }}
```

```
  release: {{ .Release.Name }}
  template:
  metadata:
  labels:
  app: {{ template "iris-rest.name" . }}
  release: {{ .Release.Name }}
  spec:
  containers:
  - image: {{ .Values.image.repository }}:{{ .Values.image.tag }}
  name: {{ template "iris-rest.name" . }}
  ports:
  - containerPort: {{ .Values.webPort.value }}
  name: {{ .Values.webPort.name }}


  $ cat templates/service.yaml
{{- if .Values.service.enabled }}
apiVersion: v1
kind: Service
metadata:
  name: {{ .Values.service.name }}
  labels:
  app: {{ template "iris-rest.name" . }}
  chart: {{ template "iris-rest.chart" . }}
  release: {{ .Release.Name }}
  heritage: {{ .Release.Service }}
spec:
  selector:
  app: {{ template "iris-rest.name" . }}
  release: {{ .Release.Name }}
  ports:
  {{- range $key, $value := .Values.service.ports }}
  - name: {{ $key }}
{{ toYaml $value | indent 6 }}
  {{- end }}
  type: {{ .Values.service.type }}
  {{- if ne .Values.service.loadBalancerIP "" }}
  loadBalancerIP: {{ .Values.service.loadBalancerIP }}
  {{- end }}
{{- end }}


  $ cat templates/helpers.tpl
{{/* vim: set filetype=mustache: */}}
{{/*
Expande o nome do chart.
*/}}
{{- define "iris-rest.name" -}}
{{- default .Chart.Name .Values.nameOverride | trunc 63 | trimSuffix "-" -}}
{{- end -}}

{{/*
Cria o nome e a versão do chart conforme usado pelo rótulo do chart.
*/}}
{{- define "iris-rest.chart" -}}
{{- printf "%s-%s" .Chart.Name .Chart.Version | replace "+" "_" | trunc 63 | trimSuffix "-" -}}
{{- end -}}
```

```
$ cat values.yaml
namespaceOverride: iris-rest
replicaCount: 1

strategy: |
  type: Recreate

image:
  repository: eu.gcr.io/iris-rest
  tag: v1

webPort:
  name: web
  value: 52773

service:
  enabled: true
  name: iris-rest
  type: LoadBalancer
  loadBalancerIP: ""
  ports:
  web:
  port: 52773
  targetPort: 52773
  protocol: TCP
```

Para criar os charts do Helm, instale o cliente Helm e o utilitário de linha de comando kubectl.

```
$ helm version
version.BuildInfo{Version:"v3.0.1", GitCommit:"7c22ef9ce89e0ebeb7125ba2ebf7d421f3e82ffa",
GitTreeState:"clean", GoVersion:"go1.13.4"}
```

Crie um namespace chamado iris. Seria bom se isso fosse criado durante a implantação, mas até agora não é o caso.

Primeiro, adicione credenciais para o cluster criado pelo Terraform ao kube-config:

```
$ gcloud container clusters get-credentials --zone --project
$ kubectl create ns iris
```

Confirme (sem iniciar uma implantação real) se o Helm criará o seguinte no Kubernetes:

```
$ cd /helm
$ helm upgrade iris-rest \
  --install \
  . \
  --namespace iris \
  --debug \
  --dry-run
```

A saída — os manifestos do Kubernetes — foi omitida por causa do espaço aqui. Se tudo estiver certo, vamos implantar:

```
$ helm upgrade iris-rest --install . --namespace iris
$ helm list -n iris --all
Iris-rest iris 1 2019-12-14 15:24:19.292227564 +0200 EET deployed iris-rest-0.1.0 1.0.3
```

Vemos que o Helm implantou nossa aplicação, mas como ainda não criamos a imagem Docker eu.gcr.io/iris-rest:v1, o Kubernetes não pode extraí-la (ImagePullBackOff):

```
$ kubectl -n iris get po
NAME READY STATUS RESTARTS AGE
iris-rest-59b748c577-6cnrt 0/1 ImagePullBackOff 0 10m
```

Vamos terminar com isso por agora:

```
$ helm delete iris-rest -n iris
```

O Lado do CircleCI

Agora que experimentamos o Terraform e o cliente Helm, vamos colocá-los em uso durante o processo de implantação no lado do CircleCI.

```
$ cat /.circleci/config.yml
version: 2.1
orbs:
  gcp-gcr: circleci/gcp-gcr@0.6.1

jobs:
  terraform:
  docker:
  # A versão da imagem do Terraform deve ser a mesma de quando
  # você executa o terraform antes da máquina local
  - image: hashicorp/terraform:0.12.17
  steps:
  - checkout
  - run:
  name: Create Service Account key file from environment variable
  workingdirectory: terraform
  command: echo ${TFSERVICEACCOUNTKEY} > account.json
  - run:
  name: Show Terraform version
  command: terraform version
  - run:
  name: Download required Terraform plugins
  workingdirectory: terraform
  command: terraform init
  - run:
  name: Validate Terraform configuration
  workingdirectory: terraform
  command: terraform validate
  - run:
  name: Create Terraform plan
  workingdirectory: terraform
  command: terraform plan -out /tmp/tf.plan
  - run:
  name: Run Terraform plan
  workingdirectory: terraform
```

```
  command: terraform apply /tmp/tf.plan
  k8sdeploy:
  docker:
  - image: kiwigrid/gcloud-kubectl-helm:3.0.1-272.0.0-218
  steps:
  - checkout
  - run:
  name: Authorize gcloud on GKE
  workingdirectory: helm
  command: |
  echo ${GCLOUDSERVICEKEY} > gcloud-service-key.json
  gcloud auth activate-service-account --key-file=gcloud-service-key.json
  gcloud container clusters get-credentials ${GKECLUSTERNAME} --zone ${GOOGLECOMPUTEZONE}
--project ${GOOGLEPROJECTID}
  - run:
  name: Wait a little until k8s worker nodes up
  command: sleep 30 # It ' s a place for improvement
  - run:
  name: Create IRIS namespace if it doesn't exist
  command: kubectl get ns iris || kubectl create ns iris
  - run:
  name: Run Helm release deployment
  workingdirectory: helm
  command: |
  helm upgrade iris-rest \
  --install \
  . \
  --namespace iris \
  --wait \
  --timeout 300s \
  --atomic \
  --set image.repository=eu.gcr.io/${GOOGLEPROJECTID}/iris-rest \
  --set image.tag=${CIRCLESHA1}
  - run:
  name: Check Helm release status
  command: helm list --all-namespaces --all
  - run:
  name: Check Kubernetes resources status
  command: |
  kubectl -n iris get pods
  echo
  kubectl -n iris get services
workflows:
  main:
  jobs:
  - terraform
  - gcp-gcr/build-and-push-image:
  dockerfile: Dockerfile
  gcloud-service-key: GCLOUDSERVICEKEY
  google-compute-zone: GOOGLECOMPUTEZONE
  google-project-id: GOOGLEPROJECTID
  registry-url: eu.gcr.io
  image: iris-rest
  path: .
  tag: ${CIRCLESHA1}
  - k8sdeploy:
  requires:
  - terraform
  - gcp-gcr/build-and-push-image
```

Você precisará adicionar várias variáveis de ambiente ao seu projeto no lado do CircleCI:

O GCLOUDSERVICEKEY é a chave da conta de serviço CircleCI e o TFSERVICEACCOUNTKEY é a chave da conta de serviço Terraform. Lembre-se de que a chave da conta de serviço é todo o conteúdo do arquivo *account.json*.

A seguir, vamos enviar nossas alterações para um repositório:

```
$ cd
$ git add .circleci/ helm/ terraform/ .gitignore
$ git commit -m "Add Terraform and Helm"
$ git push
```

O painel da IU do CircleCI deve mostrar que tudo está bem:

Terraform é uma ferramenta idempotente e se o cluster GKE estiver presente, o trabalho "terraform" não fará nada. Se o cluster não existir, ele será criado antes da implantação do Kubernetes.
Por fim, vamos verificar a disponibilidade de IRIS:

```
$ gcloud container clusters get-credentials --zone --project
$ kubectl -n iris get svc
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
Iris-rest LoadBalancer 10.23.249.42 34.76.130.11 52773:31603/TCP 53s

$ curl -XPOST -H "Content-Type: application/json" -u system:SYS 34.76.130.11:52773/person/ -d '{"Name":"John Dou"}'

$ curl -XGET -u system:SYS 34.76.130.11:52773/person/all
[{"Name":"John Dou"},]
```

Conclusão

Terraform e Helm são ferramentas DevOps padrão e devem ser perfeitamente integrados à implantação do IRIS.

Eles exigem algum aprendizado, mas depois de alguma prática, eles podem realmente economizar seu tempo e esforço.

#Conteinerização #DevOps #Docker #Google Cloud Platform (GCP) #Kubernetes #Nuvem #InterSystems IRIS #Open Exchange
Confira o aplicativo relacionado no InterSystems Open Exchange

URL de origem:https://pt.community.intersystems.com/post/automatizando-cria%C3%A7%C3%A3o-do-gke-em-compila%C3%A7%C3%B5es-do-circleci