記事
Toshihiko Minamoto · 2020年11月26日 17m read

Open Exchange

# CircleCI ビルドで GKE の作成を自動化する

前回 は GKE サービスを使用して IRIS アプリケーションを Google Cloud 上で起動しました。

また、クラスタを手動で または gcloud を介して作成するのは簡単ですが、最新の Infrastructure-as-Code (IaC) 手法 では、Kubernetes クラスタの説明もコードとしてリポジトリに格納する必要があります。
このコードの記述方法は、IaC に使用されるツールによって決まります。

Google Cloud の場合は 複数のオプション が存在し、その中には Deployment Manager と Terraform があります。
どちらが優れているかについては意見が分かれています。詳細を知りたい場合は、この Reddit のスレッド「 Opinions on Terraform vs. Deployment Manager?」と Medium の記事「 Comparing GCP Deployment Manager and Terraform」を参照してください。

この記事では特定のベンダーへの結びつきが少なく、さまざまなクラウドプロバイダーで IaC を使用できる Terraform を選択します。

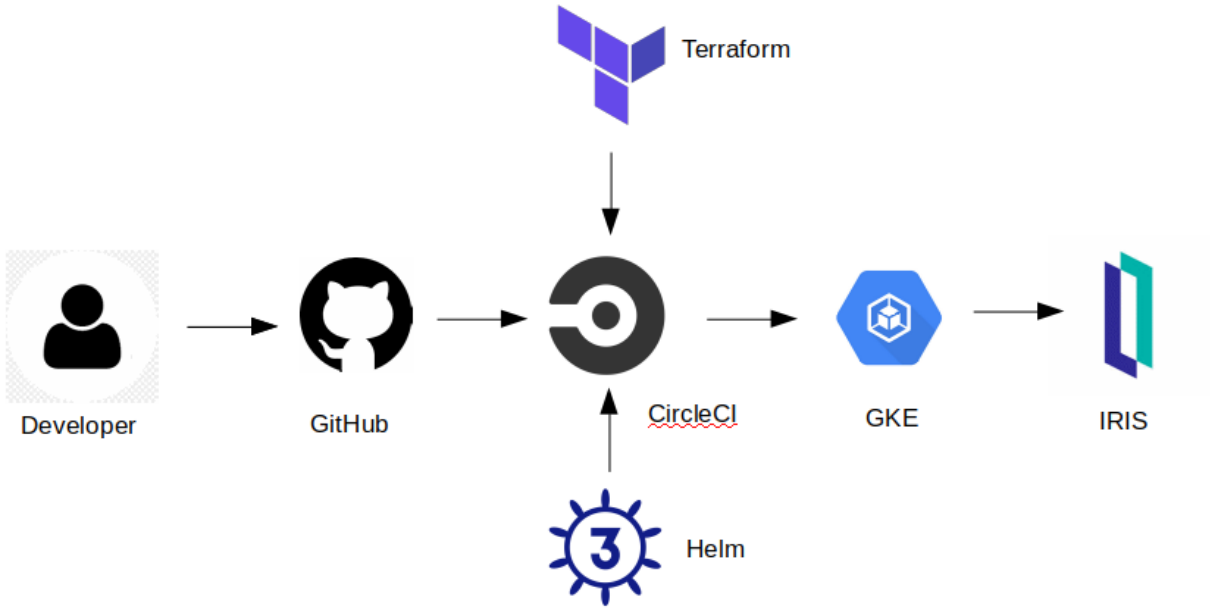
ここでは過去の記事を読み、 Google アカウント を作成、前回の記事と同様に「開発」という名前のプロジェクトを作成しているものと仮定します。
この記事ではその ID は <PROJECT_ID> として表示されます。以下の例では、それを 自分のプロジェクトの ID に変更してください。

Google には 無料枠 がありますが、無料ではないことに 注 意してください。
必ず出費をコントロール するようにしてください。

また、ここではすでに 元のリポジトリ をフォークしていることを前提にしています。この記事全体を通してこのフォークを「my-objectscript-rest-docker-template」と呼び、そのルートディレクトリを「<root_repo_dir>」として参照します。

コピー&貼り付けを簡単にするため、すべてのコードサンプルを このリポジトリ に格納しています。

次の図では、デプロイプロセス全体を 1 つの図で表しています。




では、次のように執筆時点での Terraform の最新バージョンを インストール しましょう。

```
$ terraform version
Terraform v0.12.17
```

インターネット上のほとんどの例では旧バージョンが使用されており、0.12
では ほとんどの変更 が加えられているため、ここではバージョンが重要になります。

ここでは GCP アカウントで Terraform に特定のアクションを実行（特定の API
を使用）させたいと考えています。
これを可能にするには「terraform」という名前の サービスアカウントを作成 し、Kubernetes Engine API
を有効にしてください。
その実施方法についてはご心配なく。この記事を読み進めるだけで、あなたの疑問は解消します。

Web Console を使用することもできますが、ここでは gcloud ユーティリティ を使った例を試してみましょう。

次の例では、数種類のコマンドを使用します。
これらのコマンドや機能の詳細については、次のドキュメントのトピックを参照 指定してください。

gcloud iam service-accounts create
特定のリソースのサービスアカウントへの役割の付与
gcloud iam service-accounts keys create
Google Cloud プロジェクトでの API の有効化
それでは、例を見ていきましょう。

```
$ gcloud init
```

前回の記事 で gcloud を取り上げましたので、ここではセットアップの詳細は説明しません。
この例では、次のコマンドを実行します。

```
$ cd <root_repo_dir>
$ mkdir terraform; cd terraform
```

```
$ gcloud iam service-accounts create terraform --description "Terraform" --display-
name "terraform"
```

次に、「Kubernetes Engine Admin」(container.admin)の他にいくつかのロールを terraform
サービスアカウントに追加しましょう。これらのロールは今後役に立つことでしょう。

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

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

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

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

最後の入力では、 *account.json* ファイルを作成ていることに 注 意してください。
このファイルは必ず秘密にしてください。

```
 $ gcloud projects list
$ gcloud config set project <PROJECT_ID>
$ 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
```

次に、Terraform の HCL 言語で GKE クラスタを記述しましょう。
ここではいくつかのプレースホルダを使用していますが、これらは実際の値に置き換えてください。

| プレースホルダ | 意味 |
|---|---|
| <PROJECT_ID> | GCP のプロジェクト ID |
| <BUCKET_NAME> | Terraform |

| | のステート/ロック用のストレージ(一意である必要があります) |
|---|---|
| <REGION> | リソースが作成されるリージョン |
| <LOCATION> | リソースが作成されるゾーン |
| <CLUSTER_NAME> | GKE クラスター名 |
| <NODES_POOL_NAME> | GKE ワーカーノードのプール名 |

以下に実際のクラスターの HCL 構成を示します。

```
 $ cat main.tf
terraform {
  required_version = "~> 0.12"
  backend "gcs" {
    bucket = "<BUCKET_NAME>"
    prefix = "terraform/state"
    credentials = "account.json"
  }
}
provider "google" {
  credentials = file("account.json")
  project = "<PROJECT_ID>"
  region = "<REGION>"
}

resource "google_container_cluster" "gke-cluster" {
  name = "<CLUSTER_NAME>"
  location = "<LOCATION>"
  remove_default_node_pool = true
  # In regional cluster (location is region, not zone)
  # this is a number of nodes per zone
  initial_node_count = 1
}

resource "google_container_node_pool" "preemptible_node_pool" {
  name = "<NODES_POOL_NAME>"
  location = "<LOCATION>"
  cluster = google_container_cluster.gke-cluster.name
  # In regional cluster (location is region, not zone)
  # this is a number of nodes per zone
  node_count = 1

  node_config {
    preemptible = true
    machine_type = "n1-standard-1"
```

```
    oauth_scopes = [
      "storage-ro",
      "logging-write",
      "monitoring"
    ]
  }
}
```

HCL コードを適切に整形できるよう、Terraform には次の便利な整形コマンドが用意されています。

```
 $ terraform fmt
```

上記のコードスニペットは、作成されたリソースが Google によって提供され、リソース自体 google_container_cluster と google_container_node_pool であることを示しています。また、ここではコスト削減のために preemptible を指定しています。また、デフォルトの代わりに 独自のプール を作成しています。

次の設定を簡単に説明します。

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

Terraform はすべての実行結果をステータスファイルに書き込み、このファイルを他の作業に使用します。このファイルは共有しやすいように、離れた場所に保存することをお勧めします。一般的には Google バケット に保存されます。

このバケットを作成しましょう。プレースホルダー <BUCKET_NAME> の代わりに自分のバケットの名前を使用してください。バケットを作成する前に、<BUCKET_NAME> が使用できるかどうかを次のコマンドで確認してください。すべての GCP で一意である必要があるためです。

```
 $ gsutil acl get gs://<BUCKET_NAME>
```

期待する応答:

```
 BucketNotFoundException: 404 gs://<BUCKET_NAME> bucket does not exist
```

「Busy」という応答があった場合、別の名前を選択する必要があります。

```
 AccessDeniedException: 403 <YOUR_ACCOUNT> does not have
```

storage.buckets.get access to <BUCKET_NAME>

Terraform の推奨 どおりにバージョン管理を有効にしましょう。

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

$ gsutil versioning set on gs://<BUCKET_NAME>

$ gsutil versioning get gs://<BUCKET_NAME>
gs://<BUCKET_NAME>: Enabled
```

Terraform はモジュール方式であり、GCP で何かを作成するにはGoogle Provider プラグインを追加する必要があります。これを行うには、次のコマンドを使用します。

```
 $ terraform init
```

Terraform が GKE クラスターを作成する際の実行計画を見てみましょう。

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

コマンドの出力には、計画の詳細が含まれています。
特に問題なければ、次のコマンドでこの計画を実行しましょう。

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

ちなみに、Terraform によって作成されたリソースを削除するには、このコマンドを <root_repo_dir>/terraform/ ディレクトリから実行してください。

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

しばらくクラスターから離れて先に進みましょう。
ただし、何かもリポジトリにプッシュされないように、先にいくつかのファイルを例外に追加しましょう。

```
 $ cat <root_repo_dir>/.gitignore
.DS_Store
terraform/.terraform/
terraform/*.plan
terraform/*.json
```

**Helm の使用**

前回の記事では、Kubernetes のマニフェストを yaml ファイルして <root_repo_dir>/k8s/

ディレクトリに保存し、それを「kubectl apply」コマンドを使用してクラスターに送信しました。

今回は別の手法を試してみましょう。最近 バージョン 3 にアップデートされた Kubernetes のパッケージマネージャーである Helm を使用します。バージョン 2 には Kubernetes 側のセキュリティの問題があったため、バージョン 3 以降を使用してください（詳細については、Running Helm in production: Security best practices を参照してください）。まず、Kubernetes のマニフェストを k8s/ ディレクトリから チャート として知られる Helm パッケージにまとめます。Kubernetes にインストールされている Helm チャートはリリースと呼ばれます。
最小構成は、チャートは次のような複数のファイルで構成されます。

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

これらのファイルの目的は、公式サイト で詳細に説明されています。
独自チャートを作成するためのベストプラクティスは、Helm ドキュメントの The Chart Best Practices Guide に記載されています。

次にファイルの内容を示します。

```
 $ 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: */}}
{{/*
Expand the name of the chart.
*/}}
{{- define "iris-rest.name" -}}
{{- default .Chart.Name .Values.nameOverride ¦ trunc 63 ¦ trimSuffix "-" -}}
{{- end -}}

{{/*
Create chart name and version as used by the chart label.
*/}}
{{- 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
```

Helm チャートを作成するには、 Helm クライアント と kubectl コマンドラインユーティリティをインストールします。

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

iris というネームスペースを作成ます。
デプロイ中にこれが作成れていれば良かったのですが、 現時点ではその動作は実装されていません 。

まず、Terraform によって作成れたクラスタの資格情報を kube-config に追加します。

```
 $ gcloud container clusters get-credentials <CLUSTER_NAME> --zone
<LOCATION> --project <PROJECT_ID>
$ kubectl create ns iris
```

Helm が Kubernetes で以下を作成ることを（実際にデプロイを開始せずに）確認します。

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

ここでは、出力（Kubernetes のマニフェスト）をスペースを確保るために省略しています。
特に問題がなければ、デプロイしましょう。

```
 $ 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
```

Helm がアプリケーションをデプロイしたことはわかりますが、Docker イメージ eu.gcr.io/iris-rest:v1

をまだ作成していないため、Kubernetes がそのイメージをプルすることはできません(ImagePullBackOff)。

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

とりあえず、今はここで終っておきましょう。

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

**CircleCI 側**

Terraform と Helm クライアントを試しましたので、それらを CircleCI 側のデプロイプロセスで使用できるようにしましょう。

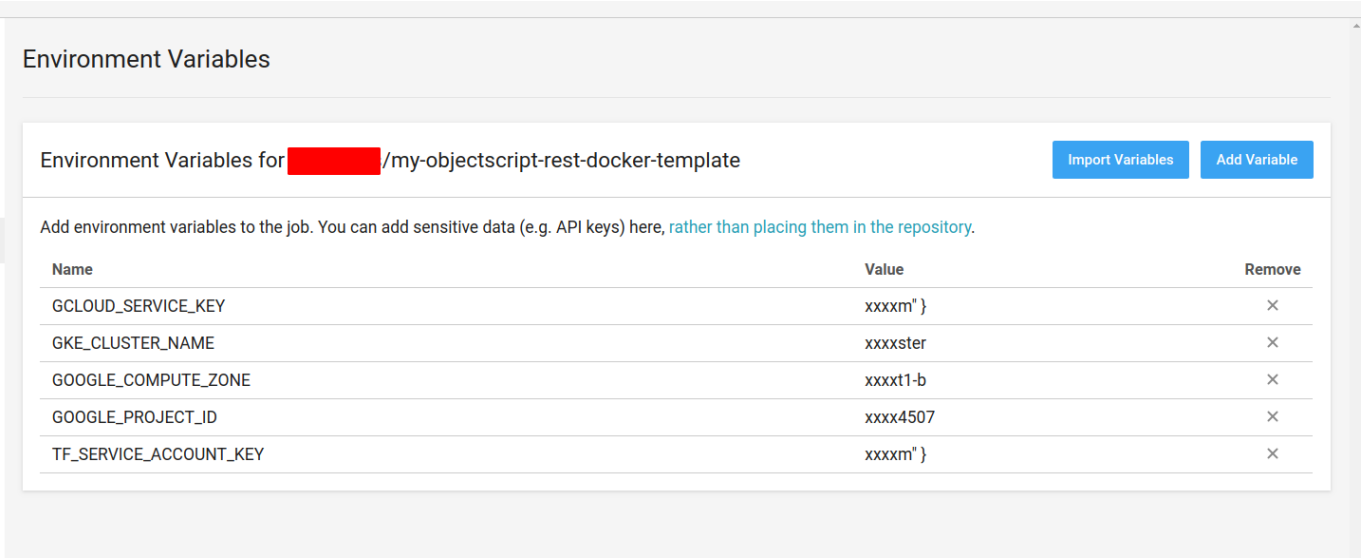
```
 $ cat <root_repo_dir>/.circleci/config.yml
version: 2.1
orbs:
  gcp-gcr: circleci/gcp-gcr@0.6.1

jobs:
  terraform:
    docker:
    # Terraform image version should be the same as when
    # you run terraform before from the local machine
      - image: hashicorp/terraform:0.12.17
    steps:
      - checkout
      - run:
          name: Create Service Account key file from environment variable
          working_directory: terraform
          command: echo ${TF_SERVICE_ACCOUNT_KEY} > account.json
      - run:
          name: Show Terraform version
          command: terraform version
      - run:
          name: Download required Terraform plugins
          working_directory: terraform
          command: terraform init
      - run:
          name: Validate Terraform configuration
          working_directory: terraform
          command: terraform validate
      - run:
```

```
        name: Create Terraform plan
        working_directory: terraform
        command: terraform plan -out /tmp/tf.plan
    - run:
        name: Run Terraform plan
        working_directory: terraform
        command: terraform apply /tmp/tf.plan
  k8s_deploy:
    docker:
      - image: kiwigrid/gcloud-kubectl-helm:3.0.1-272.0.0-218
    steps:
      - checkout
      - run:
          name: Authorize gcloud on GKE
          working_directory: helm
          command: |
            echo ${GCLOUD_SERVICE_KEY} > gcloud-service-key.json
            gcloud auth activate-service-account --key-file=gcloud-service-key.json
            gcloud container clusters get-credentials ${GKE_CLUSTER_NAME}
--zone ${GOOGLE_COMPUTE_ZONE} --project ${GOOGLE_PROJECT_ID}
      - run:
          name: Wait a little until k8s worker nodes up
          command: sleep 30 # It’s a place for improvement
      - run:
          name: Create IRIS namespace if it doesn't exist
          command: kubectl get ns iris || kubectl create ns iris
      - run:
          name: Run Helm release deployment
          working_directory: helm
          command: |
            helm upgrade iris-rest  \
              --install  \
              .  \
              --namespace iris  \
              --wait  \
              --timeout 300s  \
              --atomic  \
              --set image.repository=eu.gcr.io/${GOOGLE_PROJECT_ID}/iris-rest  \
              --set image.tag=${CIRCLE_SHA1}
      - 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: GCLOUD_SERVICE_KEY
          google-compute-zone: GOOGLE_COMPUTE_ZONE
          google-project-id: GOOGLE_PROJECT_ID
          registry-url: eu.gcr.io
          image: iris-rest
          path: .
          tag: ${CIRCLE_SHA1}
      - k8s_deploy:
          requires:
            - terraform
            - gcp-gcr/build-and-push-image
```

CircleCI 側のプロジェクトに次のようないくつかの 環境変数 を追加する必要があります。

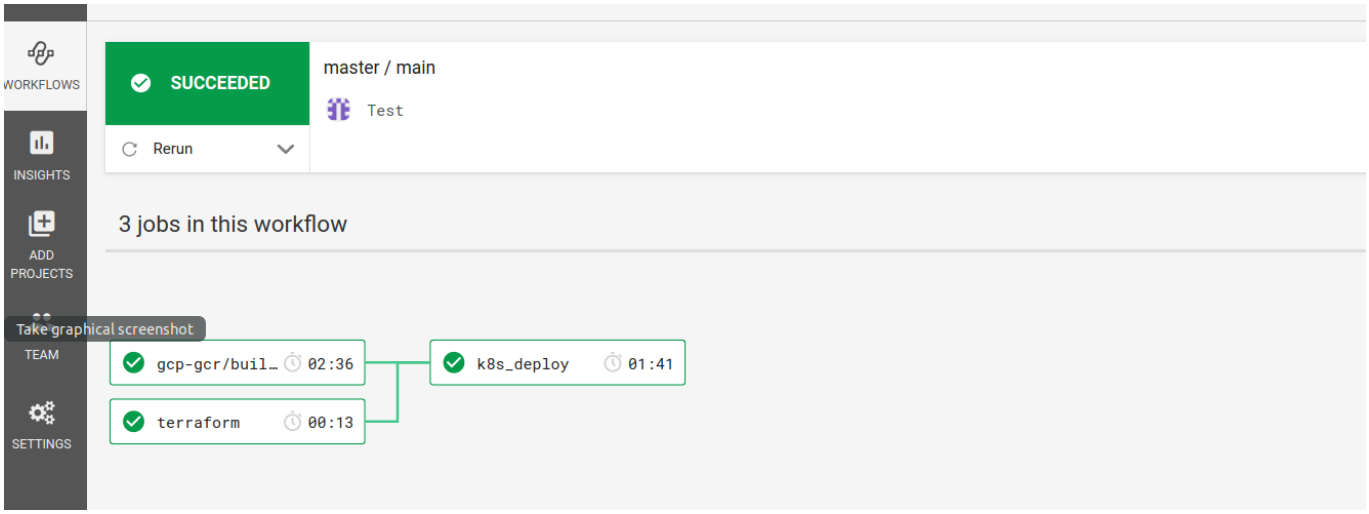



GCLOUD_SERVICE_KEY は CircleCI のサービスアカウントキーであり、TF_SERVICE_ACCOUNT_KEY は Terraform のサービスアカウントキーです。サービスアカウントキーが *account.json* ファイル全体内容であることを思い出してください。

次に、変更をリポジトリにプッシュしましょう。

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

CircleCI UI ダッシュボードには、次のようにすべてが正常であることが示されているはずです。




Terraform は冪等性あるツールであり、GKE クラスターが存在する場合、「terraform」ジョブは無実行しません。クラスターが存在しない場合は、Kubernetes をデプロイする前に作成されます。最後に、IRIS の可用性を確認しましょう。

```
 $ gcloud container clusters get-credentials <CLUSTER_NAME> --zone
<LOCATION> --project <PROJECT_ID>
$ 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"},]
```

**まとめ**

Terraform と Helm は標準の DevOps ツールであり、IRIS のデプロイと緊密に統合する必要があります。

これらはある程度の学習を必要としますが、一度か実践した後は大幅に時間と労力を節約できるようになります。

#DevOps #Docker #GCP #Kubernetes #クラウド #コンテナ #InterSystems IRIS #Open Exchange
InterSystems Open Exchangeで関連アプリケーションを確認してください

**ソースURL:** https://jp.community.intersystems.com/post/circleci-%E3%83%93%E3%83%AB%E3%83%89%E3%81%A7-gke-%E3%81%AE%E4%BD%9C%E6%88%90%E3%82%92%E8%87%AA%E5%8B%95%E5%8C%96%E3%81%99%E3%82%8B