

Case Técnico - DevOps

Julho 2023

Gabriel Antonio Fogaça Vieira

Acesso externo a aplicação: <http://35.247.231.37.nip.io/>

Projeto GitHub: https://github.com/Gasbrieu/GKE_Nginx

Tecnologias utilizadas:

Docker - Utilizei o docker para gerar uma imagem python possibilitando rodar o programa no cluster Kubernetes,

Python - Através da lib Flask e Gunicorn foi criado uma aplicação web simples retornando "Hello World!",

Cloud Build - Utilizei arquivos yaml para configuração da imagem Docker através do Cloud Build para facilitar na atualização do projeto hospedado no GitHub,

Cloud Deploy (CI/CD) - Criei um pipeline de entrega contínua no Cloud Deploy utilizando dois clusters Kubernetes (Staging e Production),

Kubernetes - Criei dois cluster Kubernetes para utilizar no pipeline de CI/CD,

Nginx - Utilizei para gerenciar o tráfego no Cluster Kubernetes de VPC Privado;

Artifact Registry - Para armazenar a imagem Docker utilizada no projeto.

Inicialmente utilizaremos o git clone para clonar o repositório do projeto no Google Cloud Platform:

```
git clone https://github.com/Gasbrieu/GKE\_Nginx
```

No projeto configuramos uma pasta no Artifact Registry para armazenar a imagem Docker:

```
gcloud artifacts repositories \
  create docker-main-app \
  --repository-format=docker \
  --location=southamerica-east1 \
  --description="Docker repository"
```

Através desse repositório vamos configurar o Cloud Build com a criação da imagem através do Dockerfile do projeto:

```
! cloudbuild.yaml
1  steps:
2    - name: 'gcr.io/cloud-builders/docker'
3      args: ['build', '-t', 'southamerica-east1-docker.pkg.dev/devopssaocarlosmonks/docker-main-app/hello-world:$SHORT_SHA', '.']
4
5    - name: 'gcr.io/cloud-builders/docker'
6      args: ['push', 'southamerica-east1-docker.pkg.dev/devopssaocarlosmonks/docker-main-app/hello-world:$SHORT_SHA']
7
8    - name: 'gcr.io/google.com/cloudsdktool/cloud-sdk'
9      entrypoint: 'bash'
10     args:
11       - '-c'
12       - >
13         gcloud deploy releases create release-$BUILD_ID
14         --delivery-pipeline=hello-world-app
15         --region=southamerica-east1
16         --source=./
17         --images=southamerica-east1-docker.pkg.dev/devopssaocarlosmonks/docker-main-app/hello-world:$SHORT_SHA
18
```

Para este projeto, criei um Dockerfile que utiliza uma imagem Python3, fiz referência a sub-pasta com a aplicação e a copiei para o contêiner, logo após indiquei os comandos a serem executados como instalação das dependências e criação do servidor web :

```
Dockerfile
1 FROM python:3
2 WORKDIR /app
3
4 COPY ./app .
5
6 RUN pip install --no-cache-dir -r requirements.txt
7
8 CMD ["gunicorn", "-b", "0.0.0.0:8080", "main:app"]
```

Após a configuração do Dockerfile podemos executar o comando:

```
cd GKE_Nginx
gcloud builds submit
```

Com isso indicamos o acesso a pasta do projeto, e iniciamos o arquivo de configuração “cloudbuild.yaml”, fazendo o build do Dockerfile e salvando ele na pasta do Artifact Registry (Vale ressaltar que nesse primeiro momento possivelmente será necessário excluir as linhas 8-17 e nomear a versão manualmente).

Para criar os cluster segui o passo a passo presente na [documentação do Google](#) em relação ao Nginx e GKE para criação de regras para o firewall e IP, mas resumindo os comandos utilizados:

```
gcloud container clusters create-auto production
--zone=southamerica-east1 --enable-private-nodes
--master-ipv4-cidr=172.16.2.0/28
--enable-master-authorized-networks --master-authorized-networks
$CLOUDSHELL_IP/32
```

```
gcloud container clusters create-auto staging
--zone=southamerica-east1 --enable-private-nodes
--master-ipv4-cidr=172.16.2.0/28
--enable-master-authorized-networks --master-authorized-networks
$CLOUDSHELL_IP/32
```

Utilizei o Autopilot que é um serviço on-demand do Google Cloud para clusters Kubernetes.

Clusters do Kubernetes

criar

implantar

atualizar

visão geral

observabilidade

otimização de custos

Filtro

Insira o nome ou o valor da propriedade

<div><input type="checkbox"/></div>	Status	Nome <div>↑</div>	Local	Modo	Número de nós	Total de vCPUs	Memória total	Notificações	Marcadores
<div><input type="checkbox"/></div>	<div><div></div><div></div></div>	<div>production</div>	southamerica-east1	Autopilot		1,25	4,5 GB	—	<div><div></div><div></div></div>
<div><input type="checkbox"/></div>	<div><div></div><div></div></div>	<div>staging</div>	southamerica-east1	Autopilot		0,5	2 GB	—	<div><div></div><div></div></div>

Após a criação dos clusters, configuramos um manifesto Kubernetes para o deploy e serviço(vamos utilizar o “service” principalmente em staging, no cluster de produção a ideia é usar o Nginx como controlador de acessos) do nosso cluster:

```
k8s > ! deployment.yaml
1  ---
2  apiVersion: apps/v1
3  kind: Deployment
4  metadata:
5    name: hello-world-app
6  spec:
7    selector:
8      matchLabels:
9        app: hello-world-app
10   template:
11     metadata:
12       labels:
13         app: hello-world-app
14     spec:
15       containers:
16       - name: hello-world-app
17         image: hello-world-app
18         ports:
19         - containerPort: 8080
20
```

```
k8s > ! service.yaml
1  ---
2  apiVersion: v1
3  kind: Service
4  metadata:
5    name: hello-world-service
6  spec:
7    ports:
8    - port: 80
9      targetPort: 8080
10   type: LoadBalancer
11   selector:
12     app: hello-world-app
13
```

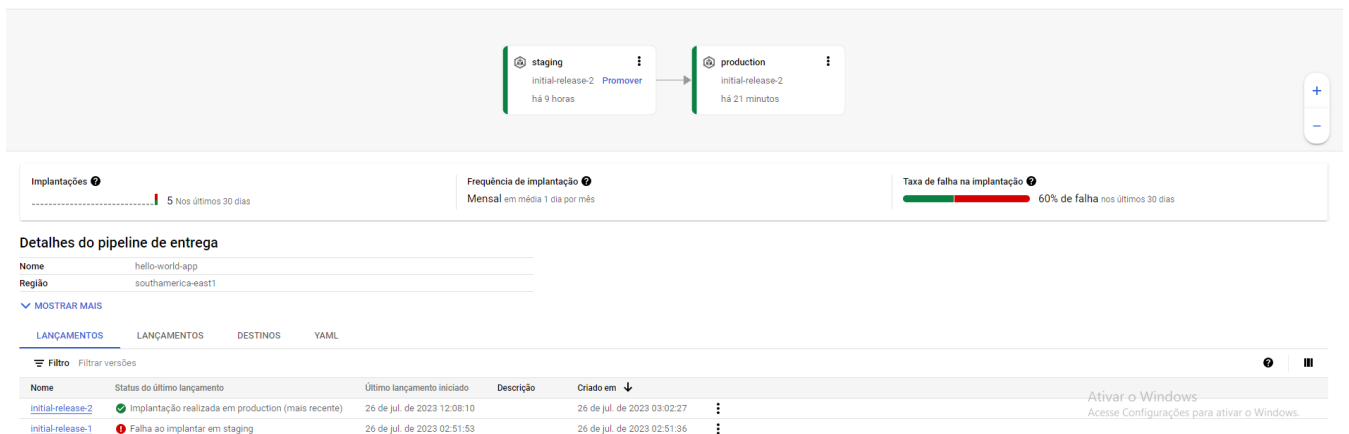
Em seguida também configuramos o arquivo de Cloud Deploy, especificando como funcionará o pipeline e o nome dos clusters utilizados:

```
! clouddeploy.yaml
1  ---
2  apiVersion: deploy.cloud.google.com/v1beta1
3  kind: DeliveryPipeline
4  metadata:
5    name: hello-world-app
6  description: Hello World Deployment Pipeline
7  serialPipeline:
8    stages:
9    - targetId: staging
10   - targetId: production
11
12  ---
13  apiVersion: deploy.cloud.google.com/v1beta1
14  kind: Target
15  metadata:
16    name: staging
17  description: Staging Environment
18  gke:
19    cluster: projects/devopssaocarlosmonks/locations/southamerica-east1/clusters/staging
20
21  ---
22  apiVersion: deploy.cloud.google.com/v1beta1
23  kind: Target
24  metadata:
25    name: production
26  description: Production Environment
27  gke:
28    cluster: projects/devopssaocarlosmonks/locations/southamerica-east1/clusters/production
29
30
```

Definindo o pipeline, devemos executar o seguinte comando:

```
gcloud deploy releases create initial-release-1
--delivery-pipeline=hello-world-app
--region=southamerica-east1 --source=.
--images=hello-world-app=southamerica-east1-docker.pkg.dev/devopss
aocarlosmonks/docker-main-app/hello-world:v1
```

Vale ressaltar que o “v1” presente no campo “–images” corresponde a versão criada para a imagem no arquivo “cloudbuild.yaml”. Após a execução do comando acima vamos ter algo parecido com esse pipeline de entrega:



Após testar a aplicação em “staging” utilizamos o botão promover no pipeline para subir as alterações realizadas ao cluster “production” onde vamos configurar o acesso externo através do Nginx.

Acessando o cluster:

```
gcloud container clusters get-credentials production --zone
southamerica-east1
```

Iniciando o pod nginx-ingress:

```
helm install nginx-ingress ingress-nginx/ingress-nginx
```

Criando uma variável de ambiente que vamos utilizar no arquivo ingress-resource.yaml:

```
export NGINX_INGRESS_IP=$(kubectl get service
nginx-ingress-ingress-nginx-controller -ojson | jq -r
'.status.loadBalancer.ingress[].ip')
```

Criação do arquivo ingress-resource.yaml:

```
cat <<EOF > ingress-resource.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-resource
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
  - host: "$NGINX_INGRESS_IP.nip.io"
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: hello-world-service
            port:
              number: 8080
EOF
```

Aplicando as configurações definidas no arquivo:

```
kubectl apply -f ingress-resource.yaml
```

Para acessar a aplicação basta pegar o “host” presente no arquivo “ingress-resource.yaml”, segue um print de um acesso externo a aplicação rodando no cluster de VPC Privada criado nesse projeto:



Segue os prints solicitados na documentação do case, vale ressaltar que utilizei a plataforma do GCP para facilitar a visualização, mas os pods podem ser listados com os comandos “kubectl get svc” e “kubectl get pod”:

Serviços e entrada

ATUALIZAR

+ CRIAR ENTRADA

EXCLUIR

Cluster

Namespace

REDEFINIR

SAVE

SERVIÇOS

ENTRADA

Os serviços são conjuntos de pods com um endpoint de rede que podem ser usados para descobertas e balanceamento de carga. As entradas são conjuntos de regras para rotear o tráfego HTTP(S) externo aos serviços.

Filtro

É objeto do sistema : Falso

Filtrar serviços e entradas

<input type="checkbox"/>	Nome ↑	Status	Tipo	Pontos de extremidade	Pods	Namespace	Clusters
<input type="checkbox"/>	hello-world-app	OK	IP do cluster	10.4.0.233	1/1	default	production
<input type="checkbox"/>	hello-world-service	OK	Balanceador de carga externo	35.198.14.247:80	1/1	default	production
<input type="checkbox"/>	hello-world-service	OK	Balanceador de carga externo	34.95.247.51:80	1/1	default	staging
<input type="checkbox"/>	nginx-ingress-ingress-nginx-controller	OK	Balanceador de carga externo	35.247.231.37:80 35.247.231.37:443	1/1	default	production
<input type="checkbox"/>	nginx-ingress-ingress-nginx-controller-admission	OK	IP do cluster	10.4.3.20	1/1	default	production

Cargas de trabalho

ATUALIZAR

+ IMPLANTAR

EXCLUIR

Cluster

Namespace

REDEFINIR

SAVE

VISÃO GERAL

OBSERVABILIDADE

OTIMIZAÇÃO DE CUSTOS

As cargas de trabalho são unidades de computação implantáveis que podem ser criadas e gerenciadas em um cluster.

Filtro

É objeto do sistema : Falso

Filtrar cargas de trabalho

<input type="checkbox"/>	Nome ↑	Status	Tipo	Pods	Namespace	Cluster
<input type="checkbox"/>	hello-world-app	OK	Deployment	1/1	default	production
<input type="checkbox"/>	hello-world-app	OK	Deployment	1/1	default	staging
<input type="checkbox"/>	nginx-ingress-ingress-nginx-controller	OK	Deployment	1/1	default	production

Como último passo podemos criar um gatilho para atualizar o projeto sempre que um commit no github é enviado, para realizar essa etapa precisamos acessar a aba de gatilhos no Cloud Build > Conectar Repositório:

The screenshot shows the Google Cloud console interface. On the left, the 'Gatilhos' (Triggers) page is visible, showing a table with columns: Nome, Descrição, Repositório, Evento, Configuração da versão, Status, and EXIBIR. A single trigger named 'pipeline-hello-world' is listed, with a description of '-', repository 'Gasbriau/GKE_Nginx', event 'Enviar por push para o branch', configuration 'cloudbuild.yaml', and status 'Ativadas'. On the right, the 'Conectar repositório' (Connect repository) dialog is open, showing a list of providers for managing code sources. The first step is 'Selecionar provedor de gerenciamento de código-fonte' (Select code source management provider), with options for GitHub, GitHub Enterprise, Bitbucket Server, Bitbucket Data Center, and Bitbucket Cloud (marked as BETA). The second step is 'Autenticar' (Authenticate), followed by 'Selecionar repositório' (Select repository) and 'Criar um gatilho (opcional)' (Create a trigger (optional)).

Nome	Descrição	Repositório	Evento	Configuração da versão	Status	EXIBIR
pipeline-hello-world	-	Gasbriau/GKE_Nginx	Enviar por push para o branch	cloudbuild.yaml	Ativadas	

Conectar repositório

1 Selecionar provedor de gerenciamento de código-fonte

Região * southamerica-west1 (Santiago)

☒ GitHub (app GitHub do Cloud Build)
Criar código-fonte como resposta para solicitar envio e enviar.

☐ GitHub Enterprise
Criar código-fonte hospedado no local como resposta para solicitar envio e enviar.

☐ Bitbucket Server
Criar código-fonte hospedado no local como resposta para solicitações de envio e pushes.

☐ Bitbucket Data Center
Criar código-fonte hospedado no local como resposta para solicitações de envio e pushes.

☐ Bitbucket Cloud (espelhado) **BETA**
Criar código-fonte como resposta para envios, espelhados pelo Cloud Source Repositories.

▼ MOSTRAR MAIS

Para prosseguir, você precisará autorizar o aplicativo GitHub do Google Cloud Build a acessar sua conta do GitHub. É possível revogar o acesso ao GitHub a qualquer momento.

CONTINUAR

2 Autenticar

3 Selecionar repositório

4 Criar um gatilho (opcional)

Realizando a autenticação e selecionando o repositório correto, por se tratar de um arquivo de configuração “cloudbuild.yaml” o gatilho vem como default já configurado.