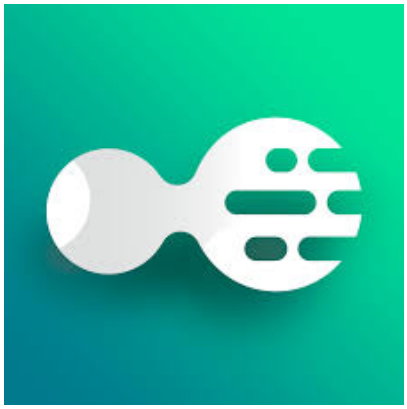

Celero

RELATÓRIO TÉCNICO DO
Desafio Celero - DevOps

Nome: **Alessandro Elias**
E-mail: ale.elias2011@gmail.com
fone: +55 41 99888 8962

Curitba
28 de abril de 2020

Sumário

Todos os arquivos mencionados neste documento podem ser encontrados em um dos repositórios públicos abaixo.

Repositório com o fork do *Blog-API-with-Django-Rest-Framework.git* (o repositório subsequente é submódulo para este):
https://github.com/alessandro11/Blog-API-with-Django-Rest-Framework.git

Repositório com artefatos para geração das imagens docker e implantação com ferramenta de automação. As imagens são: PostgreSQL, aplicação Django e Nginx webserver.
https://github.com/alessandro11/celero-devops.git

Para clonar Blog-API-with-Django-Rest-Framework.git e submódulos em uma única linha de comando execute:

$$git\ clone\ --recurse-submodules$$
$$https://github.com/alessandro11/Blog-API-with-Django-Rest-Framework.git$$

O endereço para acessar o blog online é:

blog-celero.duckdns.org
34.95.201.71
Veja como acessar por este domínio na seção 6.2.
blog.celero.com

1 Introdução

Neste relatório é apresentado uma solução para o problema proposto: implantação (deploy) de uma aplicação REST, desenvolvida com o framework Django. A implantação deve ser construída através de docker containers, visando mais de uma instância para redundância, bem como balancear a carga. Está deve ser preparada para implantação no ambiente do Google Cloud Platform (GCP).

Primeiramente as decisões tomadas para construir as imagens, foram segurança, estabilidade e custos (budget), pois armazenamento, processamento e tráfico de rede possuem custos. Este para ser minimizado, decidiu-se por construir as imagens docker baseadas no *Alpine* [1], o *Alpine* é leve e seguro, como pode ser observado em [2].

Foram construídas três imagens: uma com PostgreSQL (requisito), a segunda com Ubuntu, aplicação com framework Django. Infelizmente algumas dependências do Django não funcionam no Alpine, seria necessário fazer algumas desatualizações (downgrade), para possivelmente funcionar, ou utilizar uma imagem não oficial, o que poderia comprometer a segurança. E por último Nginx baseado no Alpine. Veja na seção 3. como é a interação entre os três containers.

1.1 GCP Produtos e Serviços

Os seguintes produtos e ou serviços foram utilizados: Compute Engine, Disks, Metadata, Cloud Build, Container Registry, Virtual Private Cloud, External IP, Firewall rules, Network Services, Cloud DNS e Cloud SDK.

Para utilizar estes recursos fora utilizado uma conta Free Trial no GCP. Devido ao primeiro contato com o GCP, em alguns pontos não foram adotadas as melhores práticas, contudo com alguma prática e estudo na documentação a adaptação será natural.

[1] https://alpinelinux.org
[2] https://nickjanetakis.com/blog/the-3-biggest-wins-when-using-alpine-as-a-base-docker-image

2 Destalhes na construção das imagens docker

Nesta seção é apresentado detalhes de implementação das imagens docker, bem como o comportamento na instanciação. Todas as VMs criadas na GCP foram criados através do Sistema Operacional Container-Optimized OS (COS), no qual já contempla docker e ferramentas necessárias para para instanciar um container no boot. É apresentado comando *gcloud* e trecho do *manifesto* para criar as VMs com respectivos containers. **Observe que alguns parâmetros irão variar, como projeto id, aqui é apresentado dois celerodevops e env-test-00001.**

2.1 PostgreSQL

Devido a natureza de volatilidade de um container, foi criado um disco de persistência dos dados. Foi utilizado o parâmetro `--disk name=storage-postgresql,...` (do GCloud SDK) para identificar o disco dentro da VM em execução. Este gera um disco identificado por id, no qual é populado em */dev/disk/by-id/google-storage-postgresql*. Este disco é formatado no primeiro boot. O script é configurado no Metadado startup-script, no qual verifica se há um sistema de arquivo no disco, caso não exista, este é formatado. Observe que há o *bootcmd:* (executado pelo serviço *cloud-init*), no qual é executado bem no princípio do boot, porém não é compatível com imagens de VMs para COS. Esta regra é mencionada devido à condição de corrida (race condition), pois se o container subir antes do disco ser montado, os dados do banco podem ir parar no ponto de montagem dentro do container (volátil). Porém podemos constatar que este race condition não ocorre. As regras de dependência são resolvidas pelo systemd, conforme informação retirada da instância da VM postgres na Figura 1 e documentação [3].

```
[Unit]
Description=Containers on GCE Setup
Wants=network-online.target google-startup-scripts.service gcr-online.target docker.socket
After=network-online.target google-startup-scripts.service gcr-online.target docker.socket

postgres /lib/systemd/system #
```

Figura 1: Conteúdo parcial do arquivo /lib/systemd/system/konlet-startup.service

Este disco é montado pelo mesmo script como um volume no container. O disco de persistência é montado no host em */mnt/disks/storage-postgresql* e no container em */var/lib/postgresql/data*.

Na imagem com o PostgreSQL foi explorado as variáveis de ambiente, *POSTGRES_PASSWORD_-_FILE* e a customizada *BLOG_PASSWORD*. A primeira é gerado um arquivo em */var/lig/postgresql -/.postgres_passwd* com uma senha aleatória, obtida através do comando `mkpasswd -m sha-512 "mypassword"` como pode ser observado no Dockerfile. Observe que não usamos "mypassword" como senha, mas os dezesseis primeiros caracteres do hash gerado. Este mecanismo possui duas vantagens: quem gerou o código de implantação não saberá a senha do usuário *postgres* (caso não possua acesso e publicação da imagem), segundo, a senha possivelmente possuirá caracteres especiais, letras maiúsculas e minusculas, números e pontuação, portanto uma senha forte. A última variável é utilizado para criar usuário *blog* (estático) e senha. Este script é disparado pela entrada de execução (entrypoint), veja o script em 01-initdb.sh. Caso não seja passado esta variável de ambiente, uma senha (`123mudar`) padrão será configurada. Esta imagem final ficou com ≈150MB, metade de uma imagem com Debian9 ≈350MB. Isto implica em menor custo na hora de transferir/migrar as imagens.

Para criar esta VM via gcloud SDK, execute o comando abaixo:
(substitua gcr.io/celerodevops/postgres-12.2-alpine-3.11:blog-0.0.5 com sua imagem, caso tenha modificado disco e rede, faça as alterações de acordo com cado recurso alocado)

```
gcloud compute instances create-with-container postgres \
  --container-image gcr.io/celerodevops/postgres-12.2-alpine-3.11:blog-0.0.5 \
  --container-env 'POSTGRES_PASSWORD_FILE=/var/lib/postgresql/.postgres_pass,\
    BLOG_PASSWORD=bIhJ6ekK$FxCKJTn0Wg47z' \
  --container-mount-host-path 'host-path=/mnt/disks/pgdata/data,mount-path=\
```

[3] https://github.com/GoogleCloudPlatform/konlet

```
    /var/lib/postgresql/data,mode=rw' \
  --container-restart-policy on-failure \
  --tags allow-postgres \
  --description 'Instance that handle PostgreSQL from persistent disk' \
  --machine-type f1-micro \
  --network-interface 'network=blog-network,subnet=blog-network-southamerica-east1,\
    private-network-ip=blog-postgresql-internal-ip' \
  --disk name=pgdata,auto-delete=no,device-name=pgdata,mode=rw \
  --metadata '^:^startup-script=#!/bin/bash
        if ! blkid /dev/disk/by-id/google-pgdata &>/dev/null; then
        mkfs.ext4 -L pgdata -m 0 -E \
        lazy_itable_init=0,lazy_journal_init=0,discard /dev/disk/by-id/google-pgdata;
        fi
        fsck.ext4 -tvy /dev/disk/by-id/google-pgdata;
        mkdir -p /mnt/disks/pgdata \
        && mount -t ext4 -o discard,defaults \
        /dev/disk/by-id/google-pgdata /mnt/disks/pgdata \
        && [ ! -d "/mnt/disks/pgdata/data" ] && mkdir
        "/mnt/disks/pgdata/data"'
```

Note que este comando possui algumas dependências, como disco e rede que devem ser criados previamente. Veja em gcloud-sdk.sh a ordem dos comandos.

Está imagem também foi gerada através do manifesto (somente um trecho é apresentado, para ver na integra veja na seção 6.1.

```
...
  - key: gce-container-declaration
    value: |
      apiVersion: v1
      kind: Pod
      metadata:
        name: blog-postgres-instance
      spec:
        containers:
        - env:
          - name: BLOG_PASSWORD
            value: bIhJ6ekK$FxCKJTnOWg47z
          - name: POSTGRES_PASSWORD_FILE
            value: /var/lib/postgresql/.postgres_pass
          image: gcr.io/env-test-00001/postgres-12.2-alpine-3.11:blog-0.0.3
          name: blog-postgres-instance
          volumeMounts:
          - mountPath: /var/lib/postgresql/data
            name: pgdata
            readOnly: false
        restartPolicy: Always
        volumes:
        - hostPath:
            path: /mnt/disks/pgdata/data
          name: pgdata
  - key: startup-script
...
```

Em resumo os problemas de dependência entre os componentes (recursos GCP) são resolvidos através do Deployment Manager, no qual é defino as inter dependências entre os componentes.

2.2 Aplicação Blog-API-with-Django-Rest-Framework

Na imagem baseada no Ubuntu, foi criado o usuário *app* (/home/app), sem privilégios no sistema (visando segurança), em caso de falha na aplicação e um atacante chegar até a máquina (Container) há mais uma barreira para impedir a execução de comandos que requerem privilégios. Dentro deste diretório foi criado um ambiente virtual python para a aplicação e utilizado *pip* para instalar todas as dependências. Na geração desta imagem é clonado do fork do repositório Blog-API-with-Django-Rest-Framework [4]. Antes de fazer as mudanças no arquivo *settings.py* foi feito um dump dos dados do sqlite, e estão armazenados no db.json, para posteriormente popularmos o Postgres.

No arquivo settings.py foi alterado a engine do baco de dados para o postgres, e as configurações: NAME, USER, PASSWORD, HOST para conexão com o banco de dados foi deixado um placeholder, no qual é alterado pelo docker-entrypoint.sh no momento da instanciação da imagem, estes parâmetros é esperado via env da linha de comando do docker. As migrações ocorrem a cada instanciação. Isto pode gerar uma pergunta. E, em caso de alteração no banco e existir uma migração? sim uma instância poderá estar diferente da outra. Isto é desejável? Depende. Caso a migração não cause problemas de retro compatibilidade, nenhuma alteração seria necessária, basta ter uma nova instância, é uma vantagem, porém caso não seja retro compatível esta abordagem causará problemas de compatibilidade no banco.

Para fazer a interface com o servidor web foi instalado o Web Server Gateway Interface (WSGI) *Gunicorn*[5], por padrão será instanciado um work para cada core da VM. O bind para o proxy reverso é feito na porta 8080.

No caso da imagem baseada no Alpine, é a versão do python, 3.8 que gera alguns problemas.

Para criar esta VM via gcloud SDK, execute o comando abaixo:
(substitua gcr.io/celerodevops/ubuntu-18.04.4-lts:blog-0.0.4 com sua imagem, caso tenha modificado disco e rede, faça as alterações de acordo com cado recurso alocado)

```
gcloud compute instances create-with-container blog2 \
       --container-image gcr.io/celerodevops/ubuntu-18.04.4-lts:blog-0.0.4 \
       --container-env 'DB_PASSWD=bIhJ6ekK$FxCKJTnOWg47z,DB_SERVER=10.128.0.200' \
       --container-restart-policy on-failure \
       --tags blog-worker \
       --machine-tgype n1-standard-1 \
       --description 'Worker for application blog' \
       --network-interface 'subnet=webserver,private-network-ip=blog2'
```

Está imagem também foi gerada através do manifesto (somente um trecho é apresentado, para ver na integra veja na seção 6.1.

```
  - key: gce-container-declaration
    value: |
      apiVersion: v1
      kind: Pod
      metadata:
        name: blog-worker1-instance
      spec:
        containers:
        - env:
          - name: DB_PASSWD
            value: bIhJ6ekK$FxCKJTnOWg47z
          - name: DB_SERVER
            value: 10.128.0.128
          image: gcr.io/env-test-00001/ubuntu-18.04.4-lts:blog-0.0.4
          name: blog-worker1-instance
        restartPolicy: Always
  kind: compute#metadata
name: blog-worker1-instance
```

[4] Blog-API-with-Django-Rest-Framework
[5] https://gunicorn.org

```
networkInterfaces:
- accessConfigs:
  - kind: compute#accessConfig
    name: external-nat
    natIP: 35.247.202.6
    networkTier: PREMIUM
    type: ONE_TO_ONE_NAT
  fingerprint: GWqLyOn-1co=
  kind: compute#networkInterface
  name: nic0
  network:
https://www.googleapis.com/compute/v1/projects/env-test-00001/ \
               global/networks/blog-network
  networkIP: 10.128.0.131
  subnetwork:
https://www.googleapis.com/compute/v1/projects/env-test-00001/ \
              regions/southamerica-east1/subnetworks/blog-network-southamerica-east1
scheduling:
  automaticRestart: true
  onHostMaintenance: MIGRATE
  preemptible: false
```

2.3 Nginx Servidor Web

O servidor web foi baseado no Alpine, no qual ficou com o tamanho de ≈20MB. Nesta imagem apenas foi removido o arquivo de configuração *default.conf* e copiado o blog.conf para */etc/nginx/conf.d/*. Neste arquivo foi deixado um placeholder para adicionar os servidores proxy para a aplicação e o nome do servidor. O docker-entrypoint.sh é responsável por fazer o parse das variáveis de ambiente e configurar o nome do servidor e proxies. A porta 8080 esta estática, mas é possível passar também este parâmetro e deixar esta imagem tão genérica que poderá servir para diferentes propósitos.

Foi adicionado certificado através do Let's Encrypt. Não há um mecanismo para a geração automática destes certificados, o que foi feito é gerado e copiado para a imagem docker. Certamente não é a melhor abordagem para produção, todavia este mecanismo está fora do escopo deste desafio e foi adicionado como recurso suplementar.

Ainda como recurso suplementar é utilizado como DNS o dnsduck.org, no qual atualiza o IP gobal do servidor web no momento da instanciação do container, como este IP foi reservado, ele não muda.

Para criar esta VM via gcloud SDK, execute o comando abaixo:
(substitua gcr.io/celerodevops/ngix-alpine-3.11:blog-0.0.7 com sua imagem, caso tenha modificado disco e rede, faça as alterações de acordo com cado recurso alocado)

```
gcloud compute instances create-with-container nginx \
       --container-image gcr.io/celerodevops/ngix-alpine-3.11:blog-0.0.7 \
       --container-env '^:^SERVER_NAME=blog.celero.com:SERVERS=10.128.0.10,10.128.0.11' \
       --container-restart-policy on-failure \
       --tags http-server \
       --machine-type n1-standard-1 \
       --description 'Web server' \
       --network-interface 'address=nginx-global,network=vpc-celero,subnet=webserver,private-netw
```

2.3.1 Balanceamento de carga

Na seção do arquivo *blog.conf* mostrado abaixo, será inserido os endereços das máquinas que servirão a aplicação através do proxy reverso, servido pelo Gunicorn. Por padrão o Nginx serve a cada máquina nesta lista com a política de round-robin. Em caso de alguma máquina não responder a próxima da lista assume a carga, e a distribuição continua para as demais. Em caso de exaustão e nenhuma máquina responder, "502 Bad Gateway" será retornado para a requisição do cliente. Abaixo um trecho do blog.conf, configuração do nginx os workers da aplicação serão assinalados abaixo do comentário "# SERVERS".

```
upstream blog {
    #
    # Load balancing is round-robin by default
    #
    # DO NOT REMOVE THE COMMENT BELOW, IT IS A
    # PLACE HOLDER TO APPEND DYNAMICALLY
    #
    # SERVERS
    server 10.128.0.131:8080;
    server 10.128.0.132:8080;
    .
    .
    .
}
```

Está imagem também foi gerada através do manifesto (somente um trecho é apresentado, para ver na integra veja na seção 6.1.

```
  - key: gce-container-declaration
    value: |
      apiVersion: v1
      kind: Pod
      metadata:
        name: blog-webserver-instance
      spec:
        containers:
        - env:
          - name: SERVER_EXTERNAL_IP
            value: 35.198.42.15
          - name: SERVER_NAME_80
            value: 35.198.42.15
          - name: SERVER_NAME_443
            value: blog-celero.duckdns.org
          - name: SERVERS
            value: 10.128.0.131,10.128.0.132,10.128.0.133
          image: gcr.io/env-test-00001/ngix-alpine-3.11:blog-v0.0.11
          name: blog-webserver-instance
        restartPolicy: Always
  kind: compute#metadata
name: blog-webserver-instance
networkInterfaces:
- accessConfigs:
  - kind: compute#accessConfig
    name: external-nat
    natIP: 35.198.42.15
    networkTier: PREMIUM
    type: ONE_TO_ONE_NAT
  fingerprint: s6ZlWroaYTQ=
  kind: compute#networkInterface
  name: nic0
  network: https://www.googleapis.com/compute/v1/projects/env-test-00001/\
           global/networks/blog-network
  networkIP: 10.128.0.2
  subnetwork: https://www.googleapis.com/compute/v1/projects/env-test-00001/\
              regions/southamerica-east1/subnetworks/blog-network-southamerica-east1
```

3 Solução na nuvem

A solução adotado é implantar em alguma regional definida na linha de comando de gcloud ou Deployment Manager. Deste modo é possível instanciar cada recurso em uma arquitetura mais simples. Porém é possível definir em diversas regionais e construir uma infra estrutura através de VPC peering, VPN. Nesta abordagem, recursos de rede e disco devem permanecer na mesma regional

Os recursos de rede foram alocados IPs internos estáticos, dentro de uma Virtual Private Cloud (VPC), desta forma não importa em qual projeto e ou região será feito a implantação, os IPs internos sempre serão os mesmos. Vantagem deste modelo, é segurança e controle. O IP global para o webserver também é estático, uma vez criado o recurso só mudará se for recriado. Os IPs alocados na subrede 10.128.0.0/16 são:

- 10.128.0.2 webserver nginx.
- 10.128.0.129 banco de dados postgres.
- 10.128.0.131 worker1 proxy reverso gunicorn.
- 10.128.0.132 worker2 proxy reverso gunicorn.
- 10.128.0.133 worker3 proxy reverso gunicorn.

Um disco de persistência foi criado para o banco de dados. A solução de arquitetura é apresentado na Figura 2.

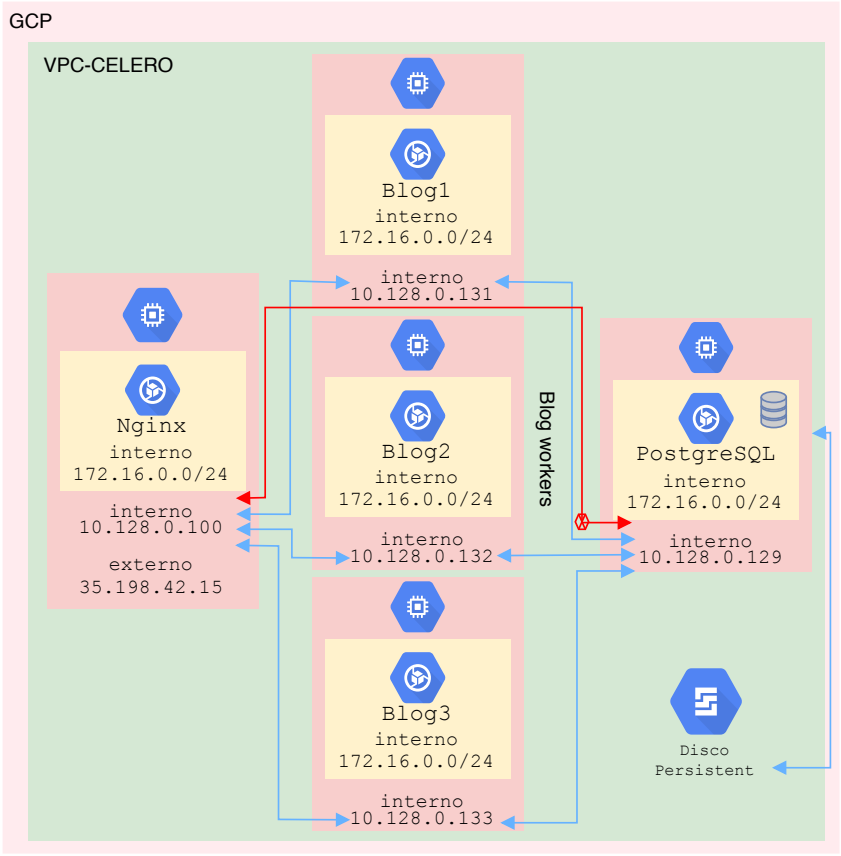

Figura 2: Arquitetura da implantação.

A Figura 2 apresenta as VMs dentro da rede vpc-celero (verde claro). A caixa rosa clara representa a VM, amarelo claro são os containers. Conexões com o banco de dados somente os workers podem fazer. Conexões com os workers, somente o web server pode estabelecer. A seta vermelha indica qual conexão não pode ser estabelecida.

4 Segurança e controle de acesso

A segurança para acessar as VMs é feita através do firewall. As sequintes regras foram aplicadas:

```
- name: 'allow-ssh'
  allowed:
    - IPProtocol: tcp
      ports:
        - '22'
  direction: INGRESS
  priority: 65534
  description: 'Allow ssh'

- name: 'allow-db'
  allowed:
    - IPProtocol: tcp
      ports:
        - '5432'
  sourceRanges:
    - 10.128.0.128/26
  direction: INGRESS
  priority: 65534
  description: 'Allow db connection to workers range'

- name: 'allow-workers'
  allowed:
    - IPProtocol: tcp
      ports:
        - '8080'
  sourceRanges:
    - $(ref.blog-ips.addresses['blog-webserver-internal-ip'].address)/32
  direction: INGRESS
  priority: 65534
  description: 'Allow connection to webserver only'

- name: 'allow-webserver'
  allowed:
    - IPProtocol: tcp
      ports:
        - '80'
        - '443'
  direction: INGRESS
  priority: 10
  description: 'Allow http(s)'

- name: 'allow-icmp-webserver'
  allowed:
    - IPProtocol: icmp
  targetTags:
    - allow-icmp2webserver
  direction: INGRESS
  priority: 65533
  description: 'Allow icmp'
```

Observe que a regra allow-db 10.128.0.128/26 é definido para a seguinte faixa:

```
Address:   10.128.0.128          00001010.10000000.00000000.10 000000
```

```
Netmask:   255.255.255.192 = 26 11111111.11111111.11111111.11 000000
Wildcard:  0.0.0.63             00000000.00000000.00000000.00 111111
=>
Network:   10.128.0.128/26      00001010.10000000.00000000.10 000000
HostMin:   10.128.0.129         00001010.10000000.00000000.10 000001
HostMax:   10.128.0.190         00001010.10000000.00000000.10 111110
Broadcast: 10.128.0.191         00001010.10000000.00000000.10 111111
Hosts/Net: 62                    Class A, Private Internet
```

está regra não permite que hosts como o webserver acesse o banco de dados, logo caso seja necessário escalar horizontalmente o webserver basta colar na faixa 2 ao 128. Workers basta colocar na faixa de 129 ao 190 62 hosts que automaticamente haverá comunicações entre workers e banco de dados.

O controle de acesso as VMs é feita através de chaves ssh adiciona ao projeto. O banco está protegido por senha. Uma melhor solução de gerência de acesso é utilizar Cloud Identity and Access Management (Cloud IAM), Não implementado, este ficou como trabalho futuro.

5 Como reproduzir a aplicação

No repositório o arquivo play-app.sh executa os passos para reproduzir a aplicação em containers localmente. Para fazer upload das imagens é necessário habilitar Google Container Registry API. Também é necessário configurar credenciais do GCP (configure-docker).

Docker e core-utilities devem estar instalados na máquina que for executar este script.

Como ele é executado?

Assumindo que o repositório já foi clonado. Entre no diretório onde foi clonado e execute a seguinte linha de comando:

$$./play - app.sh$$

Você verá a seguinte saída:

```
Usage: ./play-app.sh [-a|-b|-p|-r|-h]
    Build and run docker images do deploy the application
    Blog-API-with-Django-Rest-Framework.
    The following images will be build and or run:
    - PostgreSQL
    - Blog-API-with-Django-Rest-Framework.
    - Nginx

    [-a] - build and run images
    [-b] - just build images
    [-p] - push the images built
    [-r] - just run images
    [-h] - this help

    If no parameters has been assigned, -a is implied.
    If you wish to change the name and tag of those images:
        - gcr.io/env-test-0002/postgres-12.2-alpine-3.11:blog-0.0.10
        - gcr.io/env-test-0002/ubuntu-18.04.4-lts:blog-0.0.10
        - gcr.io/env-test-0002/ngix-alpine-3.11:blog-0.0.10

    Edit those varriables:
    PROJECT_ID, IMG_POSTGRES, TAG_POSTGRES, IMG_APP, TAG_APP, IMG_NGINX, TAG_NGINX
```

Observe que é possível apenas gerar as images, somente executá-las e ou ambos em um único comando.

Construa e execute push das images, elas devem estar em gcr.io/ para utilizar o Deployment Manager (veja seção 6.1).

```
./play-app.sh -bp
```

6 Bônus

Esta seção visa contemplar alguns desafios propostos adicionais, como: fazer deploy automático de forma a organizar e reconstruir o ambiente de forma prática. E algum recurso adicional utilizado se for o caso.

6.1 GCP Deployment Manager orquestrador

No repositório o diretório contém arquivos de configurações para executar o deploy de forma orquestrada. Para utilizar o orquestrador *gcloud deployment-manager* do GCP supomos que esteja instalado e funcional. Também é necessário estar habilitado a API deploymentmanager.googleapis.com.
Antes que No arquivo blog.yaml há o primeiro recurso config no qual deve-se setar alguns parâmetros: Configure o nome e tag das imagens que foi gerado, o play-app.sh script gera as imagens. Se desejar mude a senha do usuário (blog) do banco de dados.

```
postgres:
  image: 'postgres-12.2-alpine-3.11:blog-0.0.3'
  db_blog_password: 'bIhJ6ekK$FxCKJTnOWg47z'
worker:
  image: 'ubuntu-18.04.4-lts:blog-0.0.4'
webserver:
  image: 'ngix-alpine-3.11:blog-v0.0.11'
```

Antes de fazer o deploy, verifique se o projeto corrente é o desejado.

```
gcloud config get-value project
```

o deploy será feito no projeto corrente.
Execute o comando abaixo a partir do diretório deployment-manager/:

```
gcloud deployment-manager deployments create blog -config blog.yaml -preview
```

você verá a saída como está:

```
The fingerprint of the deployment is ro5MVPY7ykTJLSwrWEfdOg==
Waiting for create [operation-1589766087881-5a5e242f1b4e5-95a49093-9fb9220e]...done.
Create operation operation-1589766087881-5a5e242f1b4e5-95a49093-9fb9220e completed successfully.
NAME                              TYPE                   STATE       ERRORS  INTENT
allow-db                          compute.v1.firewall    IN_PREVIEW  []      CREATE_OR_ACQUIRE
allow-icmp-webserver              compute.v1.firewall    IN_PREVIEW  []      CREATE_OR_ACQUIRE
allow-ssh                         compute.v1.firewall    IN_PREVIEW  []      CREATE_OR_ACQUIRE
allow-webserver                   compute.v1.firewall    IN_PREVIEW  []      CREATE_OR_ACQUIRE
allow-workers                     compute.v1.firewall    IN_PREVIEW  []      CREATE_OR_ACQUIRE
blog-network                      compute.v1.network     IN_PREVIEW  []      CREATE_OR_ACQUIRE
blog-network-southamerica-east1   compute.v1.subnetwork  IN_PREVIEW  []      CREATE_OR_ACQUIRE
blog-postgres-instance            compute.v1.instance    IN_PREVIEW  []      CREATE_OR_ACQUIRE
blog-postgresql-internal-ip       compute.v1.address     IN_PREVIEW  []      CREATE_OR_ACQUIRE
blog-webserver-external-ip        compute.v1.address     IN_PREVIEW  []      CREATE_OR_ACQUIRE
blog-webserver-instance           compute.v1.instance    IN_PREVIEW  []      CREATE_OR_ACQUIRE
blog-webserver-internal-ip        compute.v1.address     IN_PREVIEW  []      CREATE_OR_ACQUIRE
blog-worker1-instance             compute.v1.instance    IN_PREVIEW  []      CREATE_OR_ACQUIRE
blog-worker1-internal-ip          compute.v1.address     IN_PREVIEW  []      CREATE_OR_ACQUIRE
blog-worker2-instance             compute.v1.instance    IN_PREVIEW  []      CREATE_OR_ACQUIRE
blog-worker2-internal-ip          compute.v1.address     IN_PREVIEW  []      CREATE_OR_ACQUIRE
blog-worker3-internal-ip          compute.v1.address     IN_PREVIEW  []      CREATE_OR_ACQUIRE
postgres-persistent-disk          compute.v1.disk        IN_PREVIEW  []      CREATE_OR_ACQUIRE
```

se o comando executar com sucesso, execute:

```
gcloud deployment-manager deployments update blog
```

para visualizar o status das VMs execute:

```
gclound compute instances list
```

se você vê a saída como a seguinte:

```
The fingerprint of the deployment is tdR0tPkZu5wLnOzX2MGc-A==
Waiting for update [operation-1589766426901-5a5e25726bd6e-8c20f1bb-7dc55f11]...done.
Update operation operation-1589766426901-5a5e25726bd6e-8c20f1bb-7dc55f11 completed successfully.
NAME                              TYPE                   STATE      ERRORS  INTENT
allow-db                          compute.v1.firewall    COMPLETED  []
allow-icmp-webserver              compute.v1.firewall    COMPLETED  []
allow-ssh                         compute.v1.firewall    COMPLETED  []
allow-webserver                   compute.v1.firewall    COMPLETED  []
allow-workers                     compute.v1.firewall    COMPLETED  []
blog-network                      compute.v1.network     COMPLETED  []
blog-network-southamerica-east1   compute.v1.subnetwork  COMPLETED  []
blog-postgres-instance            compute.v1.instance    COMPLETED  []
blog-postgresql-internal-ip       compute.v1.address     COMPLETED  []
blog-webserver-external-ip        compute.v1.address     COMPLETED  []
blog-webserver-instance           compute.v1.instance    COMPLETED  []
blog-webserver-internal-ip        compute.v1.address     COMPLETED  []
blog-worker1-instance             compute.v1.instance    COMPLETED  []
blog-worker1-internal-ip          compute.v1.address     COMPLETED  []
blog-worker2-instance             compute.v1.instance    COMPLETED  []
blog-worker2-internal-ip          compute.v1.address     COMPLETED  []
blog-worker3-internal-ip          compute.v1.address     COMPLETED  []
postgres-persistent-disk          compute.v1.disk        COMPLETED  []
```

Foi feito a implantação com sucesso, acesse: blog-celero.duckdns.org
34.95.201.71
Veja como acessar por este domínio na seção 6.2.
blog.celero.com
Para desfazer o deploy basta executar:

```
gcloud deployment-manager deployments delete blog
```

6.2 Recursos suplementares

Segurança é primordial quando é implantado um ambiente na web, bem como possuir um domínio (DNS) para acesso. Para segurança nas transações foi implementado certificado através da Certificate Authority (CA) Let's Encrypt
Para gerar o certificado é necessário um domínio. No momento da escrita deste relatório não há registro de um domínio que pudesse ser utilizado. Então dois mecanismos foram adotados, um com certificado e outro sem certificado.

- Sem certificado no Cloud DNS.
- Com certificado no duckdns.org.

O primeiro há uma entrada (Address) A (Name Server) (NS) no subdomínio *celero.com* (blog.celero.com), mas este registro foi feito no GCP Cloud DNS. Como não há como delegar a zona, pois este domínio é fictício, este não é propagado, logo não é visível na internet. Para utilizar este domínio deixe como primeira opção em seu */etc/resolv.conf* o seguine NS:

nameserver 216.239.32.109

O segundo (https://blog-celero.duckdns.org/) é gratuito e possui a funcionalidade de uma API para atualizar o endereço de IP em ambientes que o web server possui lease no DHCP, não é este o caso, foi reservado um IP global fixo para a VM do nginx neste desafio. Este domínio não há necessidade de alterar o */etc/resolv.conf*, pois o servidor de DNS (.org) delega zona para (duckdns.org).
Obs.: **Observe que é possível acessar o domínio duckdns seguro e não seguro, isto foi deixado de propósito, para funcionar o domínio blog.celero.com.**