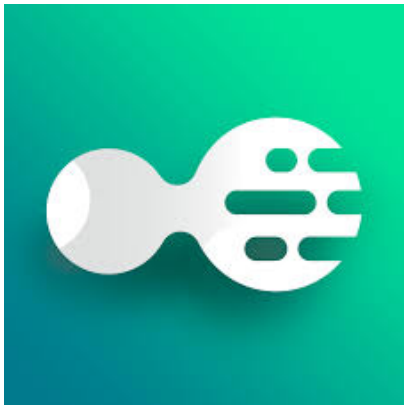

Celero

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 citados abaixo. Eles estão organizados da seguinte maneira:

Repositório com conteúdo do desafio:
O repositório abaixo possui o fork da aplicação django e o subsequente repositório como submodule.
https://github.com/alessandro11/Blog-API-with-Django-Rest-Framework.git

Neste repositório estão os atefatos para geração das imagens docker: postgres, aplicação com framework django e nginx.
https://github.com/alessandro11/celero-devops.git

Para clonar Blog-API-with-Django-Rest-Framework.git 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
35.247.200.44
Veja como acessar por este domínio na seção 6.2.
blog.celero.com

1 Introdução

Neste relatório é apresentado uma solução para o seguinte 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.

[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, bem como o comportamento na instanciação.

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 Container-Optimized OS (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 */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 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.

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 (VM) 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

[3] https://github.com/GoogleCloudPlatform/konlet
[4] Blog-API-with-Django-Rest-Framework

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.

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 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.

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.

2.3.1 Balanceamento de carga

Na seção do arquivo *blog.conf* mostrado abaixo, será inserido os endereços das máquinas que servirão a aplicação. 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
}
```

[5] https://gunicorn.org

3 Solução na nuvem

Foi adotado uma implantação simples, devido ao primeiro contato com GCP. Cuja solução é implantar na regional *southamerica-east1*, zona *southamerica-east1-a*. Utilizando o GCloud SDK, foi criado uma Virtual Private Cloud (VPC), com o nome *vpc-celero*, com Boarding Gateway Protocol (BGP) regional como protocolo de roteamento. A rede foi segmentada através da subrede nomeada de *webserver*, faixa CIDR interno: 10.128.0.0/16 e secundária 172.16.0.0/20. O ip interno 10.128.0.200 foi reservado para o servidor postgres, 10.128.0.100 / externo 35.247.200.44 para o web server e 10.128.0.10, 10.128.0.11, 10.128.0.12 para os workers (instâncias da imagem da aplicação blog).

Através GCE fora instanciadas VMs com o Container Optimized OS, no qual automaticamente sobe o container designado no parâmetro do gcloud. Neste modelo é uma VM para um container. No qual a implantação é apresentado na Figura 2.

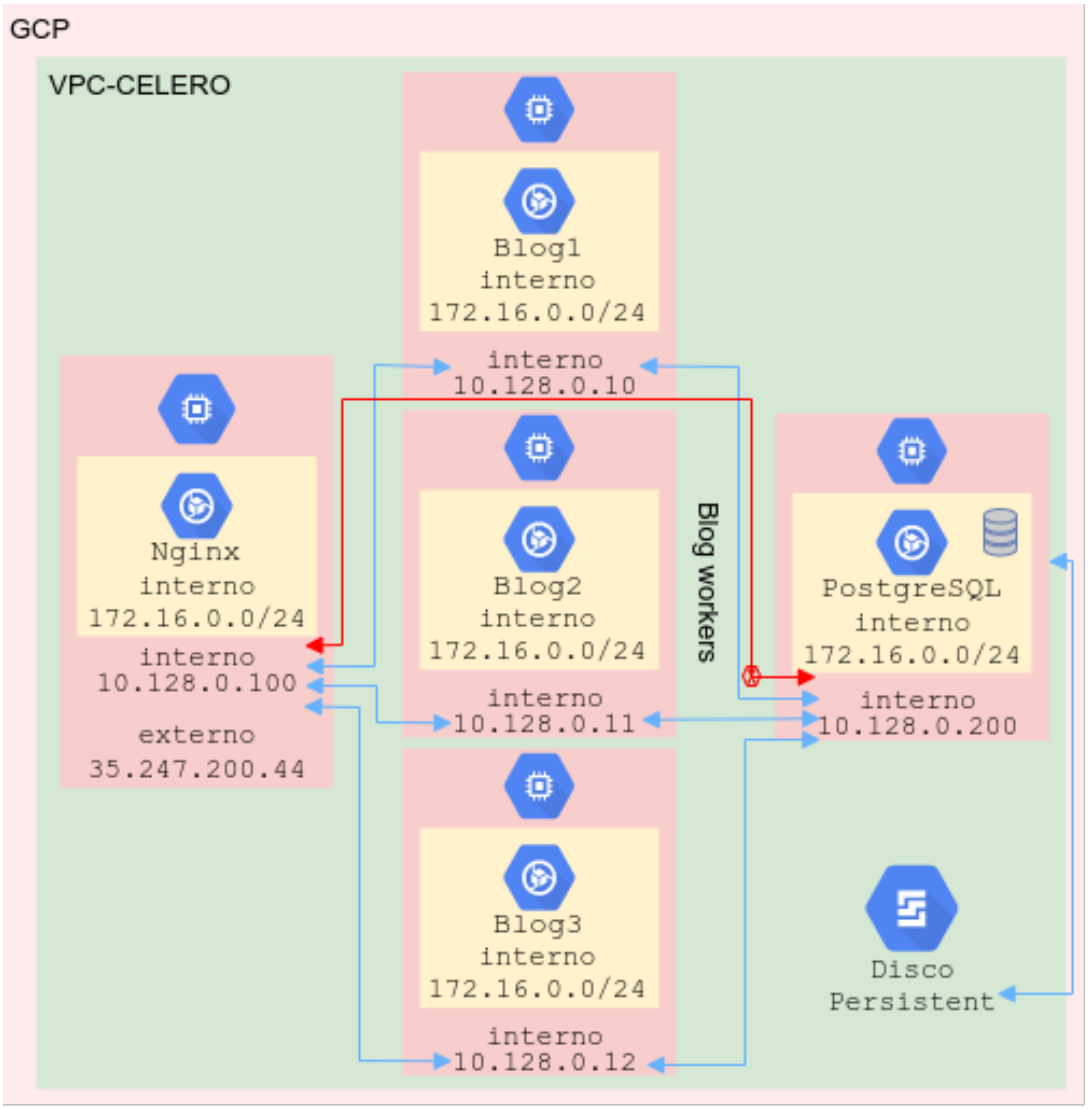

Figura 2: Arquitetura da implantação.

A Figura 2 mostra em que as VMs estão 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. Como é mostrado na seção anterior na Figura 2.

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. **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|-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
    [-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/celerodevops/postgres-12.2-alpine-3.11:blog-0.0.10
        - gcr.io/celerodevops/ubuntu-18.04.4-lts:blog-0.0.10
        - gcr.io/celerodevops/ngix-alpine-3.11:blog-0.0.10

    Edit those varriables:
    IMG_POSTGRES, TAG_POSTGRES, IMG_APP, TAG_APP, IMG_NGINX, TAG_NGINX"
Continue to build and run (y|N)?
```

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

Uma vez gerado as imagens

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, porém **não** está funcional.

> Para organizar e deixá-lo cem porcento funcional, é necessário mais um ou dois dias de trabalho. Como citado anteriormente este é meu primeiro contato com o GCP. Certamente em muitos pontos não utilizei as boas práticas da plataforma, porém mais algum tempo em contato com a ferramenta, certamente a afinidade crescerá e a eficiência também.

A ideia aqui seria executar:

$$gclouddeployment - managerdeploymentscreateblog - -configblog.yaml$$

e toda "mágica aconteceria".
A maneira que foi criado o projeto foi manual, para primeiramente absorver conhecimento transversal, e então escrever em um orquestrador. A insistência em utilizar o Deployment Manager é por estar tudo integrado e não necessitar de uma ferramenta externa, com tudo há outras ferramentas como Ansible, cheff etc, no qual tenho alguma familariedade.

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 (Adress) 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.**

6.3 MIT Licença

TODO.