Documentação Case Globo - DevOps Sr

Autor: Carlos Guilherme Matos de Almeida da Silva

Objetivos

1. **Criar duas aplicações**
    1. Utilize linguagens diferentes.
    2. Cada aplicação deve ter duas rotas:
        1. Uma retornando um texto fixo.
        2. Outra retornando o horário atual do servidor.

2. **Adicionar uma camada de cache, as respostas das aplicações devem ser cacheadas por diferentes tempos de expiração.**
    1. A primeira aplicação deve ter um cache de 10 segundos.
    2. A segunda aplicação deve ter um cache de 1 minuto.

3. **Facilitar a execução**
    1. A infraestrutura deve ser fácil de iniciar e rodar com o menor número de comandos possível.

4. **Implementar observabilidade se possível**

5. **Desenhar e analisar a infraestrutura**
    1. Criar um diagrama representando a arquitetura.

6. **Identificar e sugerir pontos de melhoria.**

7. **Atualizações**
    1. No desenho, mostre como seria o fluxo de atualização de cada componente da infra e do código.
    2. Identificar e sugerir pontos de melhoria.

Etapa 1

Foram utilizadas as linguagens de programação Python e Go para a criação das aplicações. Com o objetivo de tornar as aplicações mais próximas de um cenário real, foi optado pelo uso dos framework Flask (Python) e Echo (Go) para que a implementação tenha um caráter mais robusto mesmo dada a simplicidade das API's.

Foi criado um repositório público no Github (https://github.com/cgmattos/case-devops-globo)sionar o código de ambas as aplicações e a infraestrutura relacionada a mesma.

Ambas as aplicaçẽso possuem dois endpoints, o /hello e o /time, onde o /hello responde um JSON com uma mensagem amigável e o /time responde um JSON com a data e hora do servidor hospedando a aplicação.

Foi criada a branch feat-part-1 para versionar o código antes da implementação da camada de cache.

Etapa 2

Para a etapa 2, foi implementada uma camada de cache, utilizando o cache store Redis como base. Foi criado um docker-compose.yaml para o deploy das aplicações localmente de forma que seja simples e rápido os testes locais.

Assim como na etapa 1, foi criada a branch feat-part-2 no repositório do projeto para versionar o código utilizado para essa etapa.

Em ambas as aplicações foi priorizado o uso de connections pools para simular cenários reais.

Os tempos de cache foram configurados para seguir um valor padrão caso, mas podem ser controlados pelas mesmas através de variáveis de ambientes. Na implementação local, é utilizado o arquivo .env para passar as variáveis de ambiente para os serviços criados pelo docker-compose, porém é preciso explicitar que as aplicações necessitam de variáveis de ambiente de fato, e não um arquivo .env.

Etapa 3

Para a etapa 3 foi optado por utilizar o Terraform em conjunto com o provedor de nuvem GCP. Para o deploy das aplicações é necessário criar um fork do repositório, cloná-lo para a máquina local e criar um projeto no cloud provider GCP. Uma vez criado o projeto, é interessante seguir os passos descritos no arquivo case-devops-globo/terraform/README.md para evitar complicações e utilizar uma conta de serviço para a criação dos recursos, invés de utilizar a conta owner do projeto.

Após a preparação da conta de serviços e do arquivo access-key.json, é necessário criar uma secret no repositório com o nome ENCODED_ACCESS_KEY_JSON possuindo o valor codificado do arquivo access-key.json em base64.

Após isso, basta inserir as variáveis de ambiente com o prefixo TF_VAR_<nome da variável> e o valor desejado para adaptar o código ao seu projeto.

É importante ressaltar que o código criado utiliza o provisioner local-exec para criar as imagens das aplicações localmente e durante o tempo de execução realizar o envio das imagens para o repositório de imagens do projeto (Artifac Registry), o que não é uma boa prática, tal estratégia foi tomada para diminuir a complexidade da implementação.

Foi escolhido o Cloud Run para realizar o deploy das aplicações, devido ao seu custo mais em conta em caso de poucos acessos e a sua fácil atualização em cenários de mudanças de código.

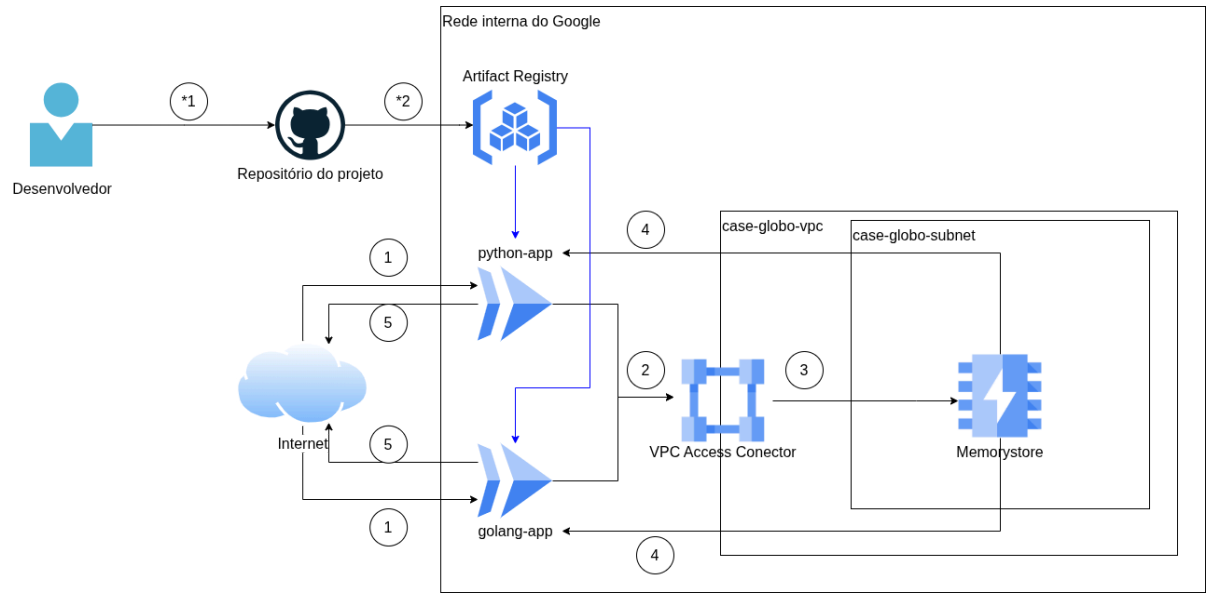
Etapa 4

Por estar sendo utilizado o ambiente da GCP e o serviço do Cloud Run, nativamente já são exportadas diversas métricas para o Cloud Monitoring, serviço de monitoramento de eventos e dados da GCP, portanto não foi necessário a implementação da solução.

Entretanto, é possível realizar melhorias criando alarmes para downtimes, número de requests e outros.

Etapa 5

Foi criado o diagrama abaixo para visualizar a solução adotada.



Fluxo da solução:

1. Usuário acessa a aplicação através do endpoint público gerado pelo Cloud Run obtido via output do terraform
2. Aplicação utiliza o VPC Access Conector para acessar a rede interna do projeto através do backbone interno da Google
3. É realizada uma consulta no Memorystore (Store de cache nativa da GCP) para obtenção da chave do cache da requisição realizada
4. É retornado o valor encontrado na estrutura de dados ou um valor em branco caso não haja a chave desejada
5. Por fim, dependendo do valor retornado pelo cache, a aplicação gera uma nova resposta, atualiza o cache e retorna ao usuário o valor ou simplesmente retorna ao usuário o valor encontrado no cache.

Fluxo de atualização de código:

1. É realizado um push de código para a branch main do repositório

2. Github Actions é acionado dependendo dos arquivos modificados. Caso alguma das pastas de código ou do terraform sejam atualizadas, é acionado um workflow que realiza um apply do código do terraform, que possui um local-exec responsável por criar as novas imagens das aplicações.
3. As imagens criadas são enviadas para o Artifact Hub e através do apply o Cloud Run é atualizado para utilizar a nova tag
4.

Como citado anteriormente, como a opção utilizada para hospedar as aplicações foi o Cloud Run, uma solução serverless, é necessário a utilização do VPC Access Conector de forma que os acessos ao Memorystore possam ser executados apenas a partir de um range específico interno da GCP, tornando o acesso ao Memorystore mais controlado e evitando tráfego de dados das aplicações pela internet pública.

Etapa 6

1. Troca de abordagem de repositório
   Em um cenário ideal, é sugerido que sejam utilizados repositórios separados para a infraestrutura e para o código, desta forma permitindo mais flexibilidade para mudanças e controle de acesso mais granular
2. Utilização de solução para gerenciamento de secrets para as variáveis das aplicações.
   Dado o escopo do case, as variáveis de ambiente das aplicações estão sendo inseridas diretamente no terraform, o que não é uma boa prática. Em um cenário ideal podem ser utilizados produtos como o Secret Manager da GCP, HashiCorp Vault e outros produtos do tipo
3. Uso de direcionamento de tráfego para diferente versões do Cloud Run
   No cenário atual, toda atualização de código gera uma nova release no Cloud Run e altera 100% do tráfego de entrada para a nova release, idealmente essa mudança de tráfego deve ser realizada de forma cadenciada, evitando desastres caso a nova versão possua algum bug.
4. Criação de alarmes para visualização de eventos de erro, scaling, uso de recursos e outros
5. Utilização do Cloud Logging para visualização mais precisa dos logs das aplicações