Área Departamental de Engenharia de Eletrónica e Telecomunicações e de Computadores

Trabalho final

Autores: 47220 João Nunes
47222 Alexandre Luís
44873 Paulo Rosa

Relatório para a Unidade Curricular de Programação da Licenciatura em Engenharia Informática e de Computadores

Computação na Nuvem

Professor: Luís Assunção

6 – 6 – 2022

Resumo

Neste trabalho de Computação da Nuvem, pretende-se criar um sistema integrado que engloba uma multitude de serviços do GCP (Google Cloud Platform), nomeadamente: Cloud Storage, Firestore, Pub/Sub, Compute Engine e Cloud Functions e Vision API.

Mais concretamente, a aplicação cliente vai suportar a submissão de imagens para o nosso servidor gRCP (e usando outros serviços), e este vai retornar ao cliente os objetos detetados nessa imagem.

Índice

Lista de figuras

1. Introdução

É assumido que o enunciado foi consultado para se poder compreender este relatório. Em primeiro lugar, foi implementado o contrato *rpcService.proto* que define as operações e como os dados vão ser transmitidos entre o Cliente e o serviço *RPC*.

De seguida, foram implementadas ambas a aplicação cliente e servidor em paralelo, onde foi necessário configurar um tópico de subscrição de mensagens, um local de armazenamento de ficheiros em nuvem, e armazenamento *noSQL* de documentos em *Firebase*, através dos serviços do *Google Cloud Platform*.

Para o processamento de imagens, a aplicação *Worker* foi realizada usando a *API Vision*, tendo acesso à base de dados *Firestore* e ao *Cloud Storage* para o armazenamento dos dados dos pedidos processados.

Foram publicadas, duas funções na nuvem, uma usada autonomamente pelo cliente, para a obtenção de um *IP* de uma instância de aplicação serviço, e outra para a monitorização do número de instâncias de aplicações *Worker*, dependendo da carga de pedidos feitos.

As conexões entre as aplicações e serviços deste projeto são ilustrados na figura 1, proveniente do enunciado.

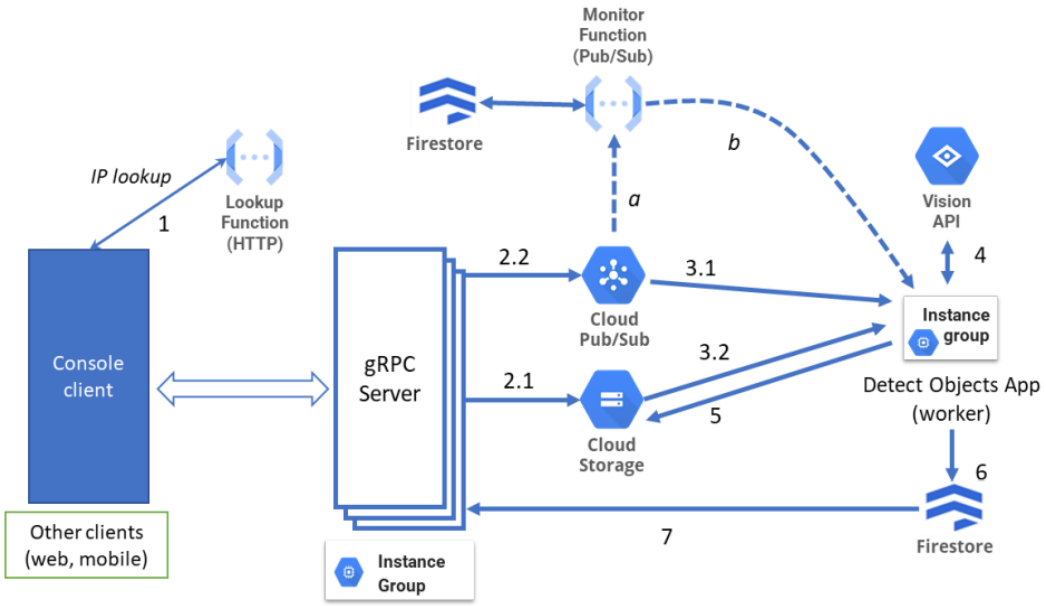

Figura 1: Componentes do *CN2122TF* e respetivas interações

2. Desenvolvimento

Abaixo listamos os programas e entidades principais no nosso trabalho, assim como as suas funções e detalhes.

Contrato

Define interfaces de comunicação entre o Cliente e o Servidor.

- ***SubmitImageFile(...)*** - Cliente envia blocos de bytes da imagem escolhida através de um objeto em stream, juntamente com o nome da mesma (*ImageRequestReply*), para que o servidor tenha informação para o envio da imagem para o cloud storage, e tópico de subscrição. Após a submissão, o servidor retorna o ID do pedido ao cliente. (*SubmitRequestReply*)
- ***GetResultObjects(...)*** - Cliente envia ao servidor o ID recebido numa submissão anterior (SubmitRequestReply) e recebe em stream os objetos vindos da procura feita pelo servidor ao Firestore (Nome e Certeza de procura). A escolha, da resposta do servidor ser feita em stream, deve-se à possibilidade da ocorrência de falhas com o servidor ao enviar a resposta. Desta forma, se o mesmo ocorrer, o cliente já deve ter recebido os objetos enviados através das chamadas do método *onNext()* da *StreamObserver*, até à ocorrência da falha.
- ***GetResultImage(...)*** - Cliente envia o ID recebido numa submissão anterior (SubmitRequestReply) e recebe em stream blocos de bytes da imagem processada e anotada (*ImageRequestReply*).
- ***SearchFilesBetween(...)*** - Cliente envia a data de início e data final para a procura, certeza de procura, e o nome do objeto, e recebe a lista de nomes de imagens encontradas nessa procura.

FireStore

Para este projeto foram usadas duas coleções para o armazenamento de dados:

- **PubSub-Requests** - Onde a função monitor, ao receber a mensagem através da subscrição, à qual está subscrita, envia as informações iniciais do pedido para a coleção:
  - ID do pedido (*request_id*)
  - Nome do bucket onde está guardada a imagem submetida (*bucket_name*)
  - Nome do *blob* que designa a imagem submetida (*blob_name*)
  - Timestamp usado para os cálculos realizados pela função monitor

E também onde as aplicações *Worker* irão completar com as seguintes informações:

- Data de processamento (*process_date*)
- Nome do *blob* onde se encontra a imagem processada (*processed_blob_name*)

- ***DetectedObjects*** - Onde são armazenados os objetos encontrados em cada imagem. Cada documento inserido descreve um objeto com as seguintes informações:
  - *obj_name*
  - *object_certainty_grade*
  - *request_id*

Cliente

Programa de consola que se conecta, aleatoriamente, a um dos servidores gRCP instanciados num grupo de instâncias da plataforma GCP.

Podem ser executados 5 comandos:

1. Submissão de uma imagem;
2. Obter a lista de objetos encontrados na imagem submetida;
3. Obter a imagem processada e anotada;
4. Procurar pelas imagens do armazenamento em nuvem, que foram processadas num certo intervalo de tempo e que contenham um objeto com um nome dado e que tenham tido um determinado valor mínimo de certeza;
5. Terminar a aplicação.

O programa deve receber como argumento o nome do grupo de instância a qual o cliente se conecta ou opcionalmente o IP e a porta do serviço.

Através do nome do grupo de instâncias, a aplicação faz uma chamada *HTTP GET* para a função da nuvem *nLookup*, que retorna os *IPs* do número de instâncias corrente. É escolhido um deles aleatoriamente, e é inicializada a conexão com a instância de servidor.

Para a submissão da imagem, a aplicação cliente lê blocos de 4*Kbytes* da imagem escolhida, e usando um *StreamObserver* envia os blocos para o servidor concluindo com a receção do ID desse pedido.

Ao receber os blocos da imagem anotada esta é escrita em ficheiro.

Todas as operações usam um *stub* não bloqueante.

Server

Programa que é replicado em cada instância do grupo de instância *instance-group-servers*. Este guarda a imagem recebida do cliente dentro uma pasta chamada "images/" num *Bucket* (cn-2022-g06) do *Cloud Storage* e constrói um pedido para o processamento da imagem que vai ser enviado para um tópico de *Pub/Sub* que

chamámos de *detectionworkers* (e que contém informações acerca do pedido, tais como o nome do *blob* da imagem)

O servidor recebe a imagem por stream de blocos, que são armazenados temporariamente em memória, e no fim, escritos num ficheiro que é enviado para o *bucket* do *Cloud Storage*.
Ainda é enviada uma mensagem para o tópico *detectionworkers* usando um *schema* criado no GCP, e em formato JSON.
Formato do schema é o seguinte:

```
syntax = "proto2";

message DetectionWork {
  string request_id = 1;
  string bucket_name = 2;
  string blob_name = 3;
}
```

O request_id é composto pelo nome do bucket utilizado e o nome da imagem a ser processada (ex: bucket1-imagem.jpg)

Para obter uma imagem processada, o servidor procura por um documento (na coleção *Pub-Sub-Requests*) que tenha um *request_id* igual ao fornecido pelo cliente, e este terá uma referência para a localização do *blob* no campo *processed_blob_name*, que obterá a imagem para ser enviada ao cliente.

Na procura entre duas datas, de imagens que tenham um certo objeto, com uma certeza acima daquela pedida, o formato da data no *Firestore* é uma *string* simples (dd/mm/yyyy), de forma a evitar *querys* com *timestamps* ao *Firestore*, devido às diferenças entre fuso horários.
Nesta procura, é pesquisado na coleção dos objetos encontrados, por aqueles que tenham o mesmo nome de objeto e grau de certeza maior ou igual, ao que o cliente enviou. De seguida é procurado na coleção dos pedidos, aqueles que tenham o mesmo ID que os documentos encontrados na pesquisa anterior, e finalmente, é verificado neste últimos documentos encontrados se as suas datas de processamento estão dentro do intervalo de tempo pedido.

Worker

Programa que é replicado em cada instância do instance group Detect Objects App (e que nós denominamos de instance-group-worker). Estas instâncias é que vão usar a Vision API para processar as imagens do utilizador. A imagem resultante vai ser colocada dentro da pasta "annotated-images/" do Bucket (cn-2022-g06) do Cloud Storage e os seus meta-dados vão ser armazenados na coleção do Firestore que se chama DetectedObjects

A aplicação subscreve ao tópico pubsub e usa a classe *MessageReceiveHandler* para receber trabalho através de uma subscrição em modo pull.

A classe *MessageReceiveHandler* recebe uma mensagem enviada pelo servidor, decodifica-a para *JSON* e envia um pedido para a Vision API para processar a imagem submetida pelo cliente, guardando essa imagem primeiro em memória antes de ser enviada à API. A API devolve a imagem anotada e esta é enviada para o cloud storage, e os objetos são criados no firestore com o esquema mencionado na seção acima.

Look Up Function

Função HTTP *triggered* que retorna a lista de IPs de servidores gRCP cujo cliente se pode conectar. O cliente chama esta função por HTTP e deve enviar como *query* parameter o nome do grupo de instância, nomeado como "instanceGroupName"
Estes IP's são retornados na forma de uma String, em que cada IP é separado por um espaço. Se não forem encontradas instâncias com o nome especificado é retornado "empty"

Monitor Function

Função Pub/Sub *triggered* que gere a interação entre as mensagens do Pub/Sub e o número de instâncias do instance group "Workers". Esta função guarda no Firestore os 3 campos que vêm numa mensagem, que são: request_id, bucket_name e blob_name. E nesta função, adicionamos um campo extra que chamámos de timestamp que representa um instante temporal (em segundos), que é usado em chamadas consecutivas da função para calcular a quantidade de mensagens nos últimos 60 segundos. Isto é, sempre que uma mensagem é recebida, é percorrida a coleção Pub-Sub-Requests e contamos o número de documentos (mensagens) que têm um timestamp cuja diferença com o instante atual seja igual ou inferior a 60s. E a partir disto, é usada a fórmula referida no enunciado para gerir o número de instâncias de Workers. É de notar que: visto que as Cloud Functions não tem garantia de guardar o seu estado (instância) nem de serem executadas por ordem, é usado o Firestore como intermediário para guardar e consultar os timestamps corretamente, assegurando idempotência.

2.2 *Deployment*

Foram preparadas duas instâncias de VM, uma para o *server* e outra para o *worker* (tal como uma *service account* com as permissões necessárias).

Ambas as configurações para as instâncias são de uma máquina *e2-micro* e . Cada instância recebeu o seu *startup script* correspondente (*server/startup.sh* e *worker/startup.sh*). Foi necessário criar uma regra de firewall *ingress* para o servidor, que neste caso usa a porta 8000. Para cada instância foi instalado o *openjdk-11-jdk* como único requisito. No fim, foi transferido o ficheiro com a chave privada do *service account* e os *fatJars* necessários. No caso do *server*, é necessário enviar também a dependência do contrato de gRPC (CN2122TF_Contract).

De seguida, foram criadas as imagens com base nas suas instâncias correspondentes.

A partir destas imagens são criados os *templates*, são usadas as mesmas configurações usadas para instâncias de VM.

Por fim, são criados os dois *instance groups*. O *instance group* do *server* tem no mínimo 1 instância até (no máximo) 3 instâncias. Ambos os *instance groups* têm o *auto-scalling* desativado uma vez que, o *server* é aumentado o número de instâncias de forma manual; enquanto que o número de instâncias do *instance group* de *workers* é controlado pela *Monitor Function*.

Para o cliente encontra-se um *README.md* na pasta *CN2122TF_Client* contendo as instruções de *build* e *execution/running*.

2.1 Testagem

De forma a testar as nossas soluções, foram realizados testes manuais, análise de logs e a consulta de dados armazenados durante a execução de programas e funções.

3. Conclusão

Neste trabalho, aplicámos e consolidámos conhecimentos lecionados nas aulas.

Fomos capazes de criar múltiplos programas que usam várias ferramentas do GCP e que respondem aos pedidos de um cliente, como foi explicado acima.

Umas das autocríticas ao nosso trabalho é sempre possível tornar o código mais robusto relativamente ao *checking* de erros na comunicação entre as variadas entidades, objetos e serviços do GCP e o input de utilizador.

Algo que deveria ser melhorado, e que tem caráter mais grave, é o facto de que os identificadores dos pedidos podem não ser únicos. Se for processada uma imagem com o mesmo nome, e for usado o mesmo bucket para o armazenamento de imagens, poderá haver IDs repetidos.

No cliente, deveria ser melhor pensado, a necessidade de todas as operações serem realizadas em stubs não bloqueantes, pois talvez usar outros bloqueantes pudesse melhorar a robustez da aplicação.

Por fim, este trabalho mostrou-nos quão complexo e diferente é programar na nuvem. E também nos elucidou ao facto de que as ferramentas do GCP tem muita capacidade, flexibilidade e fiabilidade para desenvolver programas na nuvem.

4. Responsabilidades

João Nunes – Contract, Client e Server

Alexandre Luís – Cliente, Server, Cloud storage, Pub Sub

Paulo Rosa – Cloud functions e Relatório

5. Bibliografia e recursos

Slides e código fornecidos da cadeira e documentação do Google