

**Projeto Final**

**Computação na Nuvem**

2020 / 2021 verão

**Licenciatura em Engenharia Informática e de Computadores**

**Autores:**

Diogo Monteiro, nº 44828

Pedro Silva, nº 44835

Sandro Marques, nº 46029

**Índice**

[1. Introdução 2](#_Toc75111041)

[2. Contrato 3](#_Toc75111042)

[3. Aplicação consola 4](#_Toc75111043)

[3.1. Função Cloud: GetIps 4](#_Toc75111044)

[4. Servidor gRPC 5](#_Toc75111047)

[4.1. Vision API 8](#_Toc75111053)

[4.2. Função cloud: translateAPI 8](#_Toc75111054)

[4.3. Firestore 9](#_Toc75111055)

[4.4. Cloud Storage 9](#_Toc75111056)

[4.5. Cloud Publish/Subcribe 10](#_Toc75111058)

[5. Conclusão 12](#_Toc75111060)

[6. Referências bibliográficas 13](#_Toc75111061)

### 1. Introdução

O presente trabalho foi elaborado no âmbito da unidade curricular Computação na Nuvem, do curso da Licenciatura de Engenharia Informática no Instituto Superior de Engenharia de Lisboa.

Este relatório tem como objetivo principal ilustrar as diferentes etapas de planeamento e realização de um sistema de submissão e execução de tarefas de computação na nuvem. O sistema desenvolvido terá que detetar caraterísticas de imagens, traduzi-las de inglês para português e, igualmente, possuir elasticidade suficiente para poder aumentar ou diminuir a capacidade de processamento das imagens.

As máquinas virtuais utilizadas foram preparadas para correr em ambiente Linux e foram instaladas com a versão 11 do JDK.

Para gerenciamento e execução dos serviços da *cloud*, foi utilizado uma *service account* com permissões de *Compute Admin, Cloud Datastore Owner, Storage Admin* e *Pub/Sub Admin*.

### 2. Contrato

Para estabelecer uma ligação entre o cliente e os serviços, definiu-se na linguagem *protobuf* um contrato ***grpcContract***. Foram atribuídos dois serviços, ***ReceiveFileBlocks*** e ***SendFileBlocks***, sendo o primeiro um **caso 2** uma vez que o cliente envia um pedido e recebe várias *streams* como resposta. Já no segundo serviço usou-se o **caso 3** visto que o cliente irá enviar uma sequência de mensagens numa *stream* (blocos do ficheiro) e apenas irá receber uma única resposta por parte do servidor. Estes serviços foram definidos com o objetivo de o cliente poder fazer *upload* e *download* de ficheiros, respetivamente.

Foram também definidos quatro tipos:

***RequestFiles*** quecontém 4 campos e tem como finalidade definir os parâmetros de um pedido para selecionar um conjunto de ficheiros alojados no *Storage*:

1. Data inicial;
2. Data-limite;
3. Caraterística contida nos ficheiros;
4. Se pretende fazer download da lista de ficheiros retornados.

***ResultReceived*** contém 3 campos sobre o conteúdo do ficheiro a ser descarregado.

1. Número de blocos do ficheiro;
2. Nome do ficheiro;
3. *Array* de bytes que contem o conteúdo de um ficheiro.

***Contents*** que representa o todas as informações necessárias sobre um ficheiro e contém 4 campos:

1. *Array* de *bytes* que contém o conteúdo de um ficheiro;
2. Nome do ficheiro;
3. Enumerado que representa os possíveis estados de uma *flag;*
4. *Flag* que guarda um estado.

***Result*** contém 2 campos que representa a *label* contida na respetiva imagem:

1. Nome do ficheiro;
2. *Label.*

Por fim, o cliente e o servidor fazem uso do contrato.

### 3. Aplicação consola

A aplicação cliente contém uma interface simples, estando disponível as seguintes funcionalidades:

1. Upload de uma imagem, deteção e tradução das suas caraterístiscas (0);
2. Listagem dos ficheiros dentro de um período de tempo parametrizado pelo utilizador (1);
3. Download dos ficheiros dentro de um período de tempo parametrizado pelo utilizador (2);
4. Saída da aplicação (‘*exit’*).

Note-se que, na seleção do comando do 0, será possível definir flags de **OVERWRITE** ou **KEEP\_BOTH** para casos onde exista já uma imagem repetida na *cloud storage*.

### 3.1. Função Cloud: GetIps

Para obter os endereços IP dos servidores *gRPC*, criou-se uma *Cloud Function getIps* desenvolvida num projeto *Maven* localmente. O método denominou-se de *ShowIps* e é responsável por listar todas as máquinas virtuais no link <https://europe-west2-g08-t3n-v2021.cloudfunctions.net/getIPs> que se encontram disponíveis para o cliente poder estabelecer ligação.

Através da ferramenta *gcloud* foi feito o *deployment* do ficheiro *JAR* construído a partir deste projeto utilizando o comando da fig.1.

Uma imagem com texto

Descrição gerada automaticamente

# Fig.1- Comando de *deploy* da função *getIps*

De modo a evitar a repetição de código, achou-se por bem ter um local único para armazenar dados de relevância sobre a função criada. Para tal, criou-se no *FireStore* na coleção *Information*, um documento *InstanceGroupServers* contendo metadata pertinente para que, no futuro, caso fosse necessário haver alterações de código, precisar-se apenas de serem feitas modificações nesse documento ao invés de alterar variáveis globais de múltiplos projetos (fig.2).



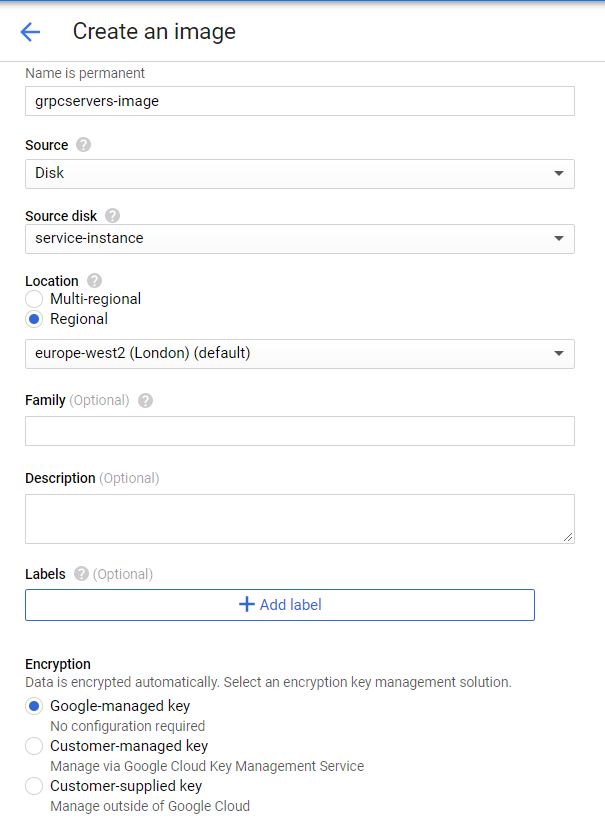
# Fig.2 Metadados do documento InstanceGroupServers

Por fim, na aplicação cliente quando esta se encontra em execução, procura-se estabelecer inicialmente a conexão com uma máquina virtual que se encontre disponível, sendo esta escolhida aleatoriamente.

### 4. Servidor gRPC

Os servidores RPC (*Remote Procedure Call*) da *Google* permitem fazer interações transparentes entre clientes e servidores com grande flexibilidade e baixa latência. Tira partido do protocolo HTTP/2 na camada de transporte uma vez que permite, na mesma ligação, a multiplexagem concorrente de vários pedidos HTTP, intercalando pedidos e respostas em *streams*.

De modo a tirar vantagem dos serviços do Google Cloud Platform, especificamente a escalabilidade e elasticidade das máquinas virtuais, criou-se uma imagem (**fig.3**) a partir do disco associado à máquina virtual onde se instalou e configurou o software que se pretende executar em todas as instâncias do grupo denominado de *grpcservers*.



# Fig.3 Configurações da imagem

Seguidamente, criou-se um *template* sendo este composto pela configuração do tipo de instância (número de CPUs, memória, rede, segurança) e pela imagem anteriormente criada que será usada nas instâncias criadas a partir deste *template*. As configurações do mesmopodem ser observadas na figura seguinte.

**Uma imagem com texto

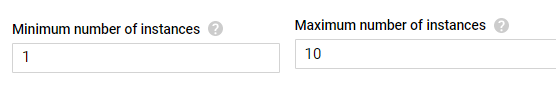
Descrição gerada automaticamente**

**Uma imagem com texto

Descrição gerada automaticamente**

# Fig.4 Configurações da *template*

Com a imagem e o *template* gerados, deu-se então a criação do grupo de instâncias *grpcservers (***fig.6***).* Como o grupo foi criado a partir de um *template,* trata-se então de um grupo de instâncias *managed,* sendo assim possível aumentar/diminuir o número de instâncias definidas entre um intervalo mínimo e máximo (**fig.5**), de modo a balancear a carga do sistema tirando partido da elasticidade.

****

**Uma imagem com texto

Descrição gerada automaticamente**

# Fig.6 Configurações do *grpcservers*

# Fig.5 Número mínimo e máximo de instâncias

Por fim, definiu-se um script para que quando a instância fosse ligada ou reiniciada, automaticamente executar o *JAR* associado ao servidor (**fig.7**). Note-se que o *JAR* foi colocado na diretoria /var do *Linux*, tendo sido utilizado o *sudo* para termos permissões para tal.

Uma imagem com texto

Descrição gerada automaticamente

# Fig.7 Script de arranque do servidor

### 4.1. Vision API

A API visão faz deteção de características sobre as imagens, retornando para cada uma delas, uma pontuação de confiança entre **0** e **1**.

Para implementar esta parte do projeto, decidimos começar por criar uma subscrição a um tópico que aguarda que um cliente faça *upload* de uma imagem. Posteriormente, essa imagem será enviada para a API visão que irá retornar uma lista de ***PairLabels*** que contém uma associação de um *score* para cada *label*. De seguida, tanto estes *PairLabels* como a metadata da imagem (*date, length, contentType, etc)* serão guardados na base de dados ***FireStore****.*

Por fim, é publicada uma mensagem para o tópico ***TOPIC\_TRANSLATE*** que irá despoletar um *trigger* na *Cloud Function* para executar o método *getTranslation*.

### 4.2. Função cloud: [translateAPI](https://console.cloud.google.com/functions/details/europe-west2/translateAPI?project=g08-t3n-v2021)

Para submeter pedidos à API de tradução, foi criado uma função *cloud* *GetTranslation* com a finalidade de traduzir texto de inglês para português. Tal como a função *cloud* GetIps, esta função foi criada num projeto *Maven* localmente. A função contém um *trigger pub/sub* sendo este despoletado sempre que há uma publicação no tópico **TOPIC-TRANSLATE.** Quando isto acontece,as caraterísticas (*labels*) são obtidas do respetivo

documento da coleção VisionResultsno Firestore, sendo colocadas numa lista de objetos *PairLabel*. Este objeto representa uma *label* contendo o nome da *label* e o seu *score*.

Seguidamente, iteramos sobre esta lista e, tirando partido da API de tradução, é feita a alteração de inglês para português. Após isso, os valores das caraterísticas traduzidas são inseridos na coleção *TranslateResults.*

### 

### 4.3. Firestore

Nesta base de dados foram criadas várias coleções, cada uma delas com vários documentos com o intuito de guardar todas a informações importantes e fulcrais para o bom funcionamento do nosso projeto.

Em primeiro lugar, temos a coleção *ComputeStorage* onde cada documento representa um *upload* de uma imagem por parte de um cliente. Cada documento tem o nome da respetiva imagem assim como vários campos que representam o Buckete Blobonde as respetivas imagens se encontram alojadas no *Storage*.

De seguida temos a coleção *Information* que contém informações globais de todo o projeto. Decidimos usar esta estratégia de forma a evitar a repetição excessiva de código ao longo do nosso projeto. Em caso de alteração de alguma informação, apenas teria que ser feita num só sítio (*FireStore*) em vez de inúmeras classes.

Temos ainda outra coleção chamada *VisionResults*, onde irão estar alojados todas as *labels* e os respetivos *scores* de confiançaassociados a cada imagem (documento), informação essa retornada pela API visão.

Por fim temos a coleção *TranslateResults*, que terá uma grande semelhança com a *VisionResults*, porém as *labels* encontram-se traduzidas pela API tradução.

### 4.4. Cloud Storage

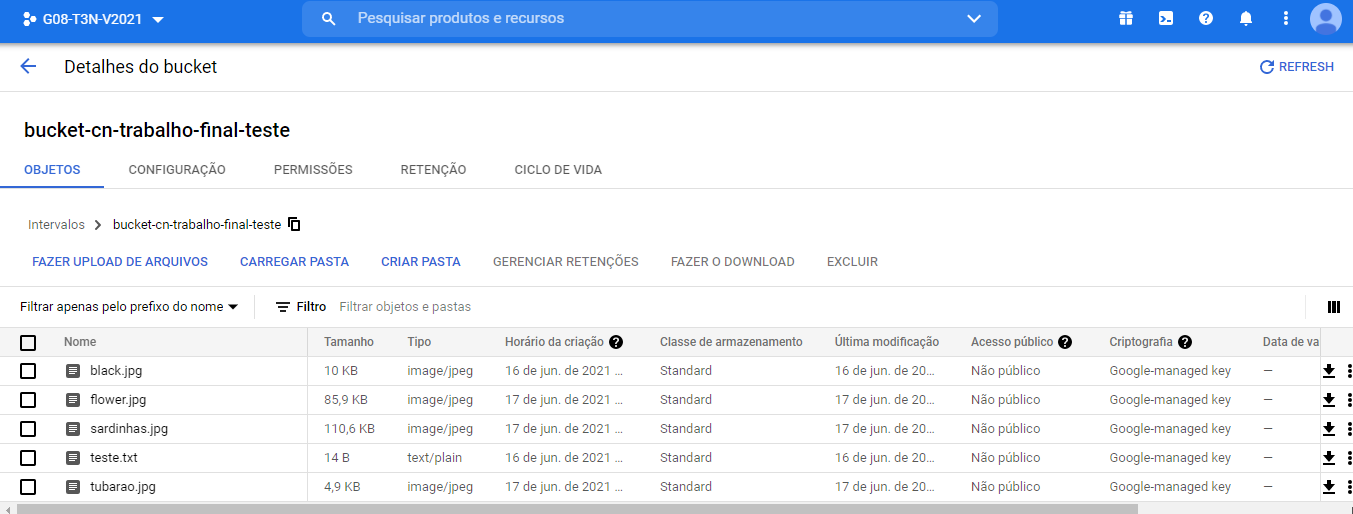
O *Cloud Storage* da *Google* é um dos módulos mais importantes do nosso projeto. É através deste que o servidor consegue alojar de forma fiável e segura todos as imagens que são carregadas pelos clientes.

O cliente e o servidor comunicam através de *streams*, enviando blocos de bytes de tamanho pré-definido. Desta forma, é possível verificar se houve alguma falha na comunicação, caso o número de blocos recebidos seja diferente ao número de blocos enviado.

Cada imagem fica alojada num blobdiferente no bucket através de objetos imutáveis de forma a poderem ser acedidos mais tarde, onde o seu nome corresponde ao nome da imagem.

Uma vez que cada *Blob* tem que ter um nome único, o Cliente pode informar o servidor, através de *flags*, se pretende que haja um *Overwrite* ou *Keep Both* caso haja uma repetição de nomes.

O cliente pode ainda escolher fazer uma listagem de todas as imagens (*Blobs)* alojados no Storage, e se pretender, fazer download das mesmas.



# Fig.7 Exemplo de blobs do bucket “bucket-cn-trabalho-final-teste”

### 4.5. Cloud Publish/Subcribe

Na realização deste Projeto, foram criados dois tópicos de subscrição: **TOPIC-TRANSLATE** e **TOPIC-VISION**. O módulo **VisionCtrl** subscreve ao tópico Vision e uma vez notificado de uma mensagem por parte da API visão, é feito um acknowledge e as *labels* são inseridas na FireStore.

De seguida, é publicada uma mensagem no tópico Translate, que irá desencadear o trigger, que por sua vez também tem uma subscrição no tópico anteriormente referido. Desta forma este modulo sabe que as labels retornadas pela API visão já se encontram na

FireStore. Este *trigger* irá comunicar com a API tradução e inserir na base de dados as traduções das *labels* anteriormente recebidas.

Uma imagem com mesa

Descrição gerada automaticamente

# Fig.8 Tópicos de Pub/Sub

### 5. Conclusão

Com este projeto tivemos a oportunidade de colocar em prática as competências adquiridas no decorrer de todas as aulas laboratoriais. Algumas destas competências incluem a utilização de diferentes serviços de computação na nuvem tais como o FireStore, PubSub, CloudFunctions, etc, assim como compreender a necessidade de coordenar e sincronizar a tomada de decisões em ambientes distribuídos.

Ao implementarmos este projeto, conseguimos também consolidar as competências adquiridas nas unidades curriculares de semestres anteriores.

Por forma a realizar este projeto, utilizamos o IDE IntelliJ para editar, compilar, correr e testar as classes Java e Protobuff. Usamos também o programa Bitvise para conseguirmos nos conectar às máquinas virtuais da GCP.

A falta de familiaridade com as dependências em Maven dificultou e atrasou o progresso inicial do projeto.

### 6. Referências bibliográficas

<https://cloud.google.com/sdk/gcloud/reference/functions/deploy>

<https://cloud.google.com/functions/docs/calling/pubsub#functions-calling-direct-pubsub-http-java>

<https://github.com/GoogleCloudPlatform/java-docs-samples/tree/master>