Descrição da Solução do Projeto CNV2024TF

Computação na Nuvem

Licenciatura em Engenharia Informática e Computadores

49513 Diogo Rodrigues

49419 Daniel Carvalho

Docentes

José Simão Luis Assunção Fernanda Passos

May 29, 2024

Índice

Lista de Figuras

Capítulo 1

Introdução

A computação na nuvem tem revolucionado a forma como sistemas e aplicações são desenvolvidos, implementados e escalados, oferecendo recursos de alta disponibilidade e flexibilidade. Este relatório aborda o desenvolvimento de um sistema denominado CNV2024TF, projetado para a deteção de características em imagens e a tradução dessas características utilizando os serviços integrados da Google Cloud Platform (GCP). O CNV2024TF exemplifica a aplicação prática de conceitos avançados de computação na nuvem, incluindo elasticidade, processamento distribuído e comunicação assíncrona.

O objetivo central deste projeto é demonstrar como planear e implementar um sistema de submissão e execução de tarefas de computação na nuvem com requisitos específicos de elasticidade. Para isso, utiliza-se uma combinação de serviços da GCP como Cloud Storage, Firestore, Pub/Sub, Compute Engine e Cloud Functions. Cada serviço desempenha um papel crucial na infraestrutura do sistema, garantindo não só o armazenamento eficiente e seguro das imagens, mas também a gestão eficaz do fluxo de dados e a escalabilidade necessária para responder a variações na carga de trabalho.

Capítulo 2

Arquitetura

2.1 Visão Geral

A arquitetura do CNV2024TF integra diversos serviços da GCP para criar um sistema escalável e eficiente para a detecção e tradução de características em imagens. O uso de Cloud Storage, Firestore, Pub/Sub, Compute Engine e Cloud Functions garante um fluxo de trabalho robusto, desde a submissão e armazenamento de imagens até ao processamento e consulta de resultados, enquanto a orquestração da elasticidade assegura que os recursos sejam utilizados de maneira eficiente conforme a variação da carga.

Esta arquitetura divide o problema em 3 grandes módulos:

- **gRPC Server -** Responsável por receber as imagens enviadas pelos clientes, armazená-las no Cloud Storage e fornecer interfaces (comunicação com o Firestore) para consultar as características detectadas e suas traduções. Também é responsável por gerir o tópico e as subscrições do Pub/Sub e enviar mensagens sobre o pedido de submissão no tópico.
- **Labels App -** Encarregada de processar as imagens, detectando características e traduzindo essas características de inglês para qualquer outra língua (presente na lista de línguas apresentada) escolhida pelo cliente, e armazenando os resultados no Firestore.
- **Logging App -** Responsável por registrar, no Firestore, todas as mensagens publicadas no tópico *images-request*, garantindo que haja um histórico detalhado sobre as propriedades da imagem submetida (utilizando Cloud Functions).

2.2 Componentes gRPC

- O serviço Cloud Storage armazena as imagens a processar [2];
- O serviço Firestore guarda a informação relevante sobre processamento de um ficheiro, nomeadamente o identificador do pedido, data do processamento, as características detetadas nas imagens e traduções, ou outros que achar convenientes [4];
- O serviço Pub/Sub é usado para a troca desacoplada de mensagens entre os servidores de gRPC e as aplicações de processamento de imagens (Labels App) [5];
- O serviço Compute Engine aloja máquinas virtuais e instance groups onde se executam os servidores de gRPC e as aplicações (Labels App) de deteção de características de imagens [3];
- O serviço Cloud Functions é usado para implementar uma cloud function para lookup dos endereços do grupo de servidores gRPC (trigger HTTP) e para fazer o logging para o Firestore, quando é enviada uma mensagem para o tópico *images-request* (trigger Pub/Sub topic) [1].

2.3 Comunicação entre componentes e fluxo de operações

2.3.1 Comunicação entre componentes

A comunicação entre os diversos componentes do sistema CNV2024TF é realizada de forma a garantir a eficiência e a escalabilidade. Os principais pontos de comunicação são:

- **Aplicação Cliente e gRPC Server:** A comunicação é feita via gRPC, onde a aplicação cliente envia imagens para o gRPC Server, que por sua vez as armazena no Cloud Storage e envia mensagens para o Pub/Sub.
- **gRPC Server e Cloud Storage:** As imagens enviadas pelos clientes são armazenadas diretamente no Cloud Storage, utilizando a API do Google Cloud Storage. Esta operação é realizada em blocos de 1MB para otimizar a transferência.
- **gRPC Server e Pub/Sub:** Após armazenar a imagem, o gRPC Server envia uma mensagem para um tópico Pub/Sub contendo o identificador do pedido, o nome do bucket e do blob. Esta mensagem é essencial para o processamento posterior da imagem.
- **Pub/Sub e Logging App:** O Logging App, que é uma Cloud Function, está subscrito ao tópico Pub/Sub. Esta subscrição é do tipo push, fazendo com que o Pub/Sub envie as mensagens diretamente para o endpoint HTTP da Cloud Function, garantindo que todas as solicitações de processamento sejam registradas no Firestore. Mesmo sendo uma Cloud Function, esta subscrição continua a respeitar o padrão fan-out.
- **Pub/Sub e Labels App:** A Labels App está subscrita ao mesmo tópico Pub/Sub, mas utilizando o padrão work-queue. Cada worker da Labels App recebe mensagens contendo referências globais (URI `gs://`) das imagens a serem processadas.
- **Labels App e Vision API/Translation API:** A Labels App utiliza a Vision API para detecção de características nas imagens e a Translation API para traduzir essas características para a língua especificada pelo utilizador.
- **Labels App e Firestore:** Após o processamento, as informações relevantes sobre as características detectadas e suas traduções são armazenadas no Firestore.
- **Aplicação Cliente e gRPC Server (Consulta):** A aplicação cliente pode, a qualquer momento, usar o identificador do pedido para consultar o gRPC Server sobre as imagens submetidas. O gRPC Server recupera as informações do Firestore e retorna-as ao cliente.

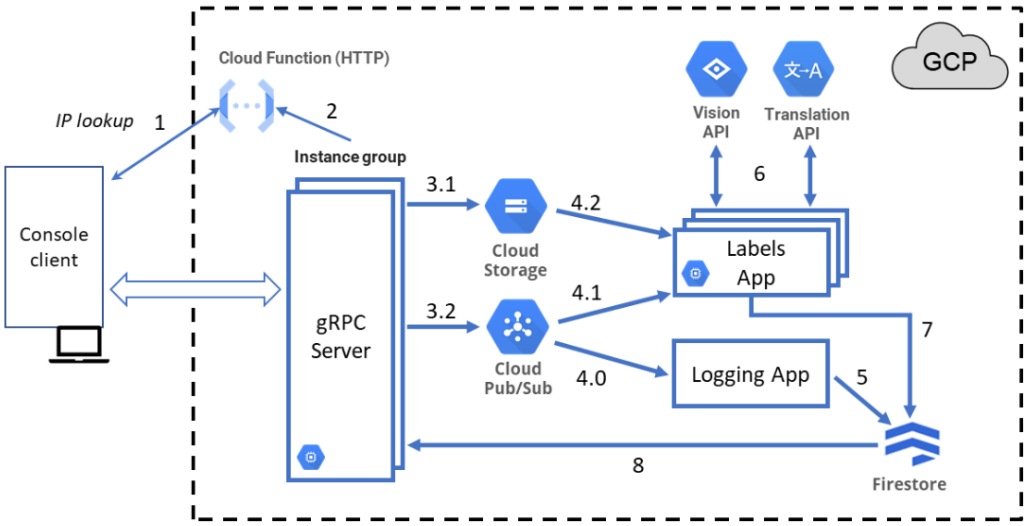

Figure 2.1: Diagrama do fluxo de operações

2.3.2 Fluxo de operações

Tendo em conta os números de sequência de ações, apresentados na Figura 2.1, a lista seguinte resume cada uma das funcionalidades:

- O serviço **Lookup Function**, usado pela aplicação cliente (1) para obtenção dos endereços IP dos servidores gRPC, deve ser desenvolvido como uma Cloud Function que obtém (2) os endereços IP das VM que fazem parte do instance group. A aplicação cliente permite ao utilizador escolher um IP da lista retornada pela função. Em caso de falha de ligação ao servidor gRPC através do IP escolhido, o utilizador tenta outro IP ou repete o processo de lookup para atualizar a lista dos IP até estabelecer uma ligação.

- Após a submissão de uma imagem, a mesma é guardada no **Cloud Storage** (3.1) e é retornado ao cliente gRPC um identificador único para posteriormente ser possível realizar as interrogações. De seguida, é enviado para um tópico **Pub/Sub** (3.2) uma mensagem que contém o identificador do pedido, o nome do bucket e do blob, onde ficou a imagem. A mensagem será posteriormente processada pela aplicação de deteção de labels.

- Com o objetivo de registar (logging) no **Firestore** todas as solicitações de processamento, existe uma subscrição no tópico para onde são enviados os pedidos, a qual é subscrita apenas pela aplicação **Logging App** (fan-out pattern). Esta subscrição Pub/Sub é do tipo push, pois o **Logging App** é uma Cloud Function. A Logging App recebe assim as mesmas informações enviadas para as aplicações de processamento (4.0). No Firestore são guardadas (5) estas informações numa coleção de nome Logs dedicada para o efeito.

- Associado ao tópico referido anteriormente existe uma outra subscrição partilhada por vários workers (work-queue pattern). Um worker (aplicação **Labels App**) de análise de imagem recebe, em cada mensagem, o nome do bucket e do blob da imagem a processar (4.1) que permite obter uma referência global (URI `gs://`) do Cloud Storage (4.2), interagindo depois com o serviço **Vision API** (6) para identificação de labels e com o serviço de tradução (6) para traduzir cada uma das labels detetadas de inglês para português ou outra língua especificada pelo utilizador.

- Após o processamento da imagem, são guardadas no **Firestore** (7) as informações relevantes do pedido e do resultado da análise.

- A aplicação cliente, a qualquer momento, usando o identificador do pedido, pede ao servidor gRPC informações sobre as imagens submetidas. Para retornar essa informação, o servidor gRPC consulta o Firestore (8).

Nota: No envio da imagem, não guardamos estado no servidor e enviamos diretamente as mensagens em blocos de 1MB para o blob. O utilizador, ao submeter uma imagem, especifica a língua para tradução a partir de uma lista apresentada. Assim, apenas a tradução para a língua desejada é armazenada no Firestore, economizando recursos.

Capítulo 3

Contratos protobuf e pressupostos

Este capítulo tem como objetivo explicar o que é o *Protocol Buffers* e como foi utilizado no projeto.

3.1 Contratos Protobuf

Protocol Buffers (ou simplesmente Protobuf) é um método flexível e eficiente de serialização de dados estruturados, desenvolvido pelo Google. Protobuf é amplamente utilizado para definir a interface e as mensagens trocadas entre serviços, principalmente em sistemas distribuídos. No contexto deste projeto, Protobuf foi utilizado para definir os contratos entre os serviços gRPC, assegurando uma comunicação eficiente e estruturada entre os componentes do sistema CNV2024TF.

Os contratos Protobuf utilizados no projeto estão descritos nos ficheiros `SGcontract.proto` e `SFcontract.proto`. Abaixo, apresentam-se exemplos de mensagens e métodos gRPC utilizados:

```
// Exemplo de mensagem no SFcontract.proto
message ImageRequest {
    string image_name = 1;
    bytes image_data = 2;
}

// Exemplo de serviço gRPC no SFcontract.proto
service ImageService {
    rpc SubmitImage (stream ImageRequest) returns (ImageResponse);
    rpc GetLabels (LabelRequest) returns (LabelResponse);
}

// Exemplo de mensagem no SGcontract.proto
message ScaleRequest {
    int32 num_instances = 1;
}

// Exemplo de serviço gRPC no SGcontract.proto
service ScalingService {
    rpc ScaleInstances (ScaleRequest) returns (ScaleResponse);
}
```

3.2 Pressupostos

Para a implementação do sistema, foram assumidos os seguintes pressupostos:

- O serviço gRPC deve estar disponível para receber conexões de múltiplas instâncias de clientes simultaneamente, garantindo alta disponibilidade e escalabilidade do sistema.
- A comunicação entre os componentes deve ser eficiente e de baixa latência, utilizando Protobuf para a serialização de mensagens.
- A persistência de dados no Firestore deve ser robusta e permitir consultas rápidas para recuperar as características e traduções das imagens processadas.
- O Pub/Sub deve assegurar a entrega de mensagens de forma confiável e ordenada, permitindo a coordenação entre o serviço de submissão de imagens e as aplicações de processamento.
- As instâncias do servidor gRPC e das aplicações de processamento devem ser geridas automaticamente pelo sistema de elasticidade, garantindo que os recursos sejam ajustados conforme a carga de trabalho.

Esses pressupostos foram estabelecidos para garantir que o sistema CNV2024TF seja escalável, eficiente e capaz de lidar com variações na carga de trabalho, proporcionando uma experiência de usuário estável e confiável.

Capítulo 4

Estrutura de Dados

Neste capitulo será representado e explicadas as estruturas de dados utilizadas ao longo do decorrer do projeto. Estas estruturas de dados são utilizadas para armazenar e manipular os dados necessários para a execução do projeto. Foram utilizadas algumas estruturas de dados nomeadamente no envio de mensagens para o *Labels App* e *Logger App* e posteriormente na submissão dos dados obtidos através das APIs externas (*Google Cloud Vision API* e *Google Cloud Translation API*) para o *FireStore*. Para a colocação das imagens no *Google Cloud Storage* foi utilizado um *bucket* que é um contentor de objetos que permite armazenar e servir conteúdo como imagens, vídeos, e ficheiros de áudio. A cada bucket são associados objetos que são os ficheiros que são armazenados no *blob*. Cada objeto tem um nome único que é o seu identificador. Um bucket pode conter vários blobs.

4.1 Estrutura de Dados Cloud Storage

Para a colocação das imagens no *Google Cloud Storage* foi utilizado um *bucket* que é um contentor de objetos que permite armazenar e servir conteúdo como imagens, vídeos, e ficheiros de áudio. A cada bucket são associados objetos que são os ficheiros que são armazenados no *blob*. Cada objeto tem um nome único que é o seu identificador. Um bucket pode conter vários blobs. Este contentor é criado pela primeira vez que é feito um pedido de envio de uma imagem onde foi criado um *bucket* com o nome *images-bucket-project-cn18* e onde são armazenadas todas as imagens que são enviadas para o *Labels App*. Estas imagens são armazenadas em *blobs* que são os ficheiros que são armazenados no *bucket* e que são identificados por um nome único que é o seu identificador. Um detalhe de implementação que será descrito mais a frente é que o nome do *blob* é descrito pela seguinte estrutura:

```
Bucket: "images-bucket-project-cn18"

    Blob: "<Nome da imagem> - <UUID>"
    Blob: "<Nome da imagem> - <UUID>"
    Blob: "<Nome da imagem> - <UUID>"
                (...)
```

Listing 4.1: Estrutura de Dados Cloud Storage

4.2 Estrutura de Dados para Envio de Mensagens

Como referido anteriormente, foi utilizado um sistema de publicação/subscrição para o envio de mensagens para o *Labels App*. Este serviço é fornecido pela *Google Cloud Platform* e é denominado de *Pub/Sub*. As mensagens são organizadas em **tópicos** e **assinaturas**. Um tópico é uma fila de mensagens que as aplicações podem publicar. Cada mensagem publicada num tópico é enviada para cada subscritor do tópico. No nosso caso temos um tópico chamado *image-requests* que é criado pela primeira vez que é feito um pedido de envio de uma imagem para o *Labels App* e que usamos para enviar as mensagens contendo o identificador do pedido, o nome do *blob* onde a imagem se encontra e o nome do seu respetivo *bucket*. Contém também um campo *language* onde é especificado o idioma para o qual se pretende traduzir a descrição da imagem que como foi referido anteriormente é dado a escolher ao utilizador qual a linguagem que ele pretende que a descrição da imagem seja traduzida.

```
{
    "requestId": "string",
    "bucket": "string",
    "blob": "string",
    "language": "string"
}
```

Listing 4.2: Estrutura de Dados para Envio de Mensagens

4.3 Estrutura de Dados para Submissão de Dados para o Firestore

Apoós a submissão da menssagem para o *Labels App*, é a vez deste obter a descrição da imagem com ajuda ao acesso da *Google Cloud Vision API* e posteriormente traduzir a descrição obtida para o idioma escolhido pelo utilizador com a ajuda da *Google Cloud Translation API*.

Após a obtenção da descrição da imagem (com o grau de precisão) e uma vez feita a tradução de cada campo detetado pela API de visão, é submetido um documento para a coleção *images* do Firestore com a seguinte estrutura:

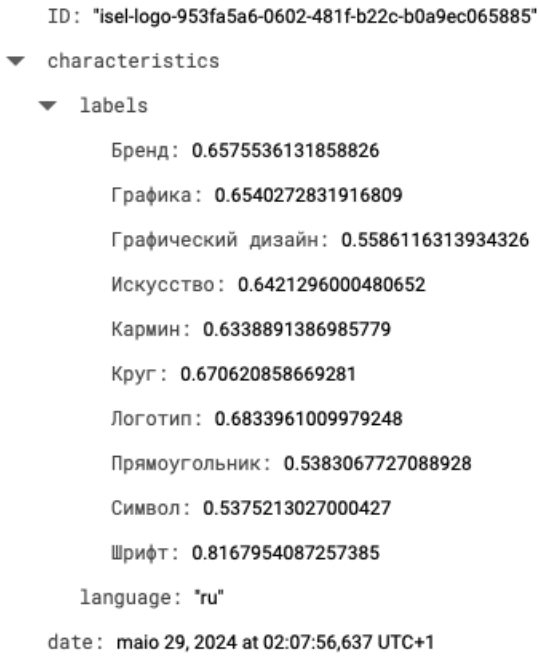

Figure 4.1: Estrutura de Dados do Firestore em Russo

Como podemos ver na figura 4.3, retirada diretamente do *Google Cloud Firestore*, a estrutura de dados do *Firestore* é composta por um documento com o nome do *blob* onde a imagem se encontra e que contém um campo **ID** que é o identificador do pedido, um campo *date* que é a data em que o pedido foi feito e um campo *characteristics*. Este campo é um *map* que contém os campos detetados pela API de visão (*labels*) que, por sua vez, é um mapa com o nome do campo e o seu respetivo valor de precisão. O campo das caracteristicas da imagem contém o campo *language* que é o idioma para o qual todas as descrições foram traduzidas.

Durante o envio da menssagem para a fila de menssagens do tópico *image-requests* é notório na aplicação que existe uma *Cloud Function* que é ativada sempre que uma nova mensagem é enviada para o tópico. Ao ser ativada esta função é responsável por registar o pedido no *Firestore* criando um documento cujo nome é um identificador único na coleção *Logs* e que contém a seguinte estrutura:

Figure 4.2: Estrutura de Dados do Firestore para Logs

Como podemos ver na figura 4.3, retirada diretamente do documento de Logs do *Google Cloud Firestore*, a estrutura de dados é composta por quatro campos: *message-attributes*, *message-id*, *message-data* e *pub-time*. O campo *message-attributes* contém os atributos da mensagem que foi enviada para o tópico *image-requests*. O campo *message-id* contém o identificador único da mensagem que foi enviada para o tópico *image-requests*. O campo *message-data* contém os dados da mensagem e *pub-time* contém a data e hora em que a mensagem foi enviada para o tópico *image-requests*.

Capítulo 5

Implementação

Neste capítulo é descrito alguns detalhes de implementação que o grupo achou relevante realçar para melhor compreensão do projeto.

5.1 IP Lookup

Quando o cliente se conecta é lhe mostrado um conjunto de opções para escolher, opções estas que contituem os IPs das máquinas virtuais onde os servidores estão alocados. Isto é feito apenas no lançamento da aplicação e não é feito de forma dinâmica, ou seja, se um servidor for adicionado ou removido, o cliente terá de reiniciar a aplicação para ver as alterações.

Para obter a lista de servidores disponiveis foi utilizado uma *Cloud Function* que é ativada através de um pedido HTTP e que devolve a lista de IPs disponiveis no projeto. No cliente existe uma função que realiza um pedido HTTP á *Cloud Function* e dependendo da resposta, mostra ao utilizador a lista de servidores disponiveis. Caso a resposta seja vazia, o cliente mostra uma mensagem sugestiva ao utilizador.

5.2 Conexão com o Servidor

Após a escolha do Ip e a conexão com o servidor estabelecida, o cliente recebe uma lista de opções com determinadas funcionalidades. Tendo o utilizador escolhido a opção 1, opção esta que permite inserir um ficheiro, o cliente envia o ficheiro para o servidor. É então a partir daqui que todo o desenrolar do projeto acontece.

Através do contrato *Protobuf*, do *gRPC* explicado no capítulo 3, o cliente envia um determinado ficheiro para o servidor que por sua vez o processa e devolve a resposta ao cliente. Este ficheiro não é enviado diretamente, mas sim dividido em blocos de 1MB e enviado bloco a bloco. Isto é feito para garantir que o ficheiro não é corrompido durante a transmissão. Isto é possível pois o gRPC permite a transmissão de dados em *streaming* e que é uma das vantagens desta tecnologia. O cliente usa um *stub* não bloqueante para enviar os blocos do ficheiro e o servidor usa um *stream* para receber os mesmos.

Após a resposta do servidor, o cliente recebe o identificador do pedido e com este pode consultar as informações que foram guardadas no *Firestore* através da seleção de uma nova opção no menu.

5.3 Processamento do Ficheiro

O servidor ao receber o ficheiro do cliente, guarda-o num *blob* do *Google Cloud Storage*. Para tal o servidor em cada chamada do cliente vai enviar os bytes do ficheiro para o *blob*. Isto é feito em cada chamada do *OnNext*, ou seja, o servidor não guarda qualquer tipo de estado do ficheiro, apenas trata da sua transmissão diretamente para o serviço da Google. Isto possibilita uma maior escalabilidade do servidor e permite que este possa processar vários ficheiros em simultâneo.

Como explicado no capitulo 4 depois da imagem estar guardada no *Cloud Storage* é retornado ao cliente o identificador do pedido e onde foi guardada a imagem (nome do blob). Por motivos de simplificação e para não complicar o projeto, o identificador do pedido é o nome do blob.

'Em simultâneo' o servidor cria um tópico (image-requests) no *Pub/Sub* (apenas na primeira vez) e envia uma mensagem para o mesmo. Após a criação do tópico o servidor cria uma subscrição (labels-app) que fica á espera de mensagens.

5.3.1 Labels App

Este módulo constitui um serviço de subscrição do *Pub/Sub* obedecendo ao padrão *work-queue*. Neste padrão, são criados múltiplos subscritores (*workers*) que ficam à espera de mensagens de um determinado tópico. Quando uma mensagem é recebida, um dos *workers* é escolhido para processar a mensagem. Este processamento consiste em aceder à API de visão da Google e obter as características da imagem.

A Figura 5.1 [6] ilustra o funcionamento do padrão *work-queue* com múltiplos *workers* processando mensagens do tópico.

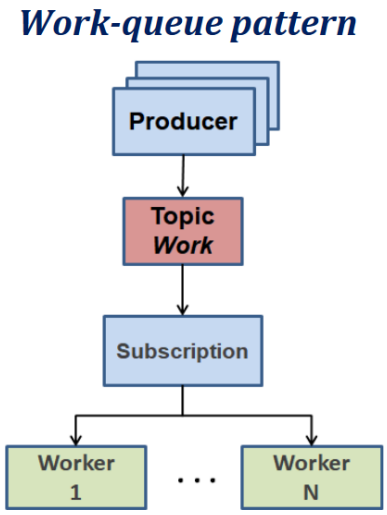

Figure 5.1: Padrão *work-queue* com múltiplos *workers*

Após a obtenção das características, utiliza-se o parâmetro *language* da mensagem do *Pub/Sub* para fazer a tradução dessas características para a língua escolhida pelo cliente. Esta tradução é realizada através da API de tradução da Google. A obtenção das caracteristicas é feita através do acesso ao *Cloud Storage* através do uso do prefixo 'gs://<bucket-name>/<blob-name>'.

Um detalhe utilizado para garantir tradução de qualquer caracteristica é a utilização da tradução para cada palavra obtida e não a tradução da lista de palavras. Isto é feito para garantir que todas as palavras são traduzidas sem problemas.

Depois de obter as características e de traduzir as mesmas, é criada uma coleção no *Firestore* denominada *images*. Nesta coleção, são criados documentos para cada imagem processada, contendo as características da imagem, a tradução dessas características e o identificador do pedido.

A **Labels App** adota o padrão *work-queue* para distribuir a carga de trabalho entre múltiplos *workers*, garantindo que as mensagens são processadas de forma eficiente e robusta.

5.3.2 Logging App

A **Logging App** é uma componente crucial da arquitetura do CNV2024TF, responsável por registar todas as mensagens publicadas no tópico *images-request*. Implementada como uma Cloud Function, a Logging App é acionada pela publicação de mensagens neste tópico, garantindo um histórico detalhado de todas as solicitações de processamento de imagens.

Sendo uma Cloud Function, a Logging App utiliza um modelo de subscrição Pub/Sub do tipo *push*. Neste modelo, o Pub/Sub envia diretamente as mensagens para o endpoint HTTP da Cloud Function, assegurando que todas as mensagens publicadas no tópico *images-request* são processadas e registadas no Firestore sem atraso.

O padrão *fan-out* é um modelo de comunicação em sistemas distribuídos onde uma única mensagem publicada num tópico é entregue a múltiplas subscrições. No contexto do CNV2024TF, isto significa que várias instâncias da Logging App podem ser criadas, cada uma com a sua própria subscrição ao tópico *images-request*. Por exemplo, se criarmos uma nova Cloud Function chamada *logging-app-1*, será automaticamente criada uma nova subscrição no tópico. Esta nova instância receberá as mesmas mensagens enviadas para as outras instâncias, assegurando redundância e escalabilidade.

Na Figura 5.2 [6], ilustra-se o funcionamento do padrão *fan-out* com múltiplas subscrições em Cloud Functions.

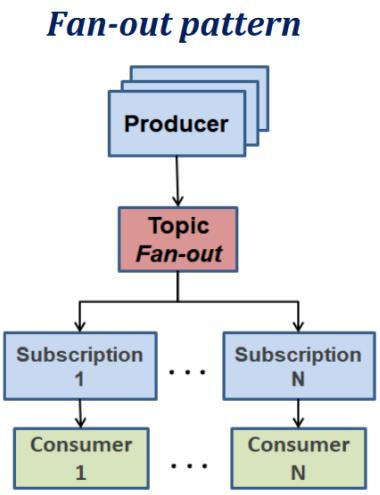

Figure 5.2: Padrão fan-out com múltiplas subscrições em Cloud Functions

A Logging App é implementada seguindo estes passos principais:

1. **Criação da Cloud Function**:
   - Configurar a função para ser acionada pela publicação de mensagens no tópico *images-request*.
   - Utilizar uma subscrição do tipo *push*, onde o Pub/Sub envia diretamente as mensagens para o endpoint HTTP da função.
2. **Processamento da Mensagem**:
   - A função recebe a mensagem, que inclui informações como o identificador do pedido, o nome do bucket e do blob.
   - Extração e validação das informações contidas na mensagem.
3. **Registo no Firestore**:
   - Após a validação, a função regista as informações da mensagem numa coleção específica no Firestore chamada *Logs*.
   - Armazenamento de detalhes como o identificador do pedido, data de submissão, e outras informações como os atributos da mensagem.
4. **Escalabilidade e Redundância**:
   - A implementação do padrão *fan-out* permite que múltiplas instâncias da Logging App processem as mesmas mensagens, aumentando a resiliência e capacidade de processamento do sistema.

Capítulo 6

Pontos de Falha

Neste capítulo serão descritos alguns dos pontos de falha do projeto devido a vários fatores como falta de tempo, falta de conhecimento, etc...

6.1 IP Lookup

No capítulo 5 foi descrito o processo de obtenção dos IPs dos servidores disponiveis para o cliente. Este processo é feito através de uma *Cloud Function* que devolve a lista de IPs disponiveis no projeto. No entanto, este processo não é feito de forma dinâmica, ou seja, se um servidor for adicionado ou removido, o cliente terá de reiniciar a aplicação para ver as alterações. Isto é um ponto de falha da nossa aplicação pois o cliente não tem a possibilidade de ver em tempo real os servidores disponiveis.

6.2 Conexão com o Servidor

Após a escolha do Ip e a conexão com o servidor estabelecida, o cliente recebe uma lista de opções com determinadas funcionalidades. No entanto, após a escolha de um IP, o cliente tem a possibilidade de diminuir e aumentar a elastecidade do sistema. Contudo, caso as intancias do servidor sejam reduzidas para 0, o cliente ficará sem acesso ao servidor e não poderá realizar qualquer tipo de operação. Isto é um ponto de falha da nossa aplicação. Terá de criar manualmente uma nova instancia do servidor na consola GCP para poder aceder novamente ao mesmo.

6.3 Tratamento de Erros

Devido à falta de tempo, não foi possível implementar um tratamento de erros eficaz. Isto é um ponto de falha da nossa aplicação pois o cliente em alguns casos não tem a informação necessária para perceber o que correu mal. Por exemplo, caso haja um erro na diminuição de instancias do servidor, o cliente não tem a informação necessária para perceber o que correu mal e como resolver o problema.

Capítulo 7

Objetivos

Neste capítulo serão descritos os objetivos futuros do projeto, bem como as melhorias que podem ser feitas para tornar a aplicação mais eficiente e eficaz.

7.1 IP Lookup

Com a obtenção de IPs dos servidores disponiveis para o cliente feita através de uma *Cloud Function*, o cliente não tem a possibilidade de ver em tempo real os servidores disponiveis. Uma melhoria futura seria a implementação de um sistema de atualização dinâmica dos IPs disponiveis, de forma a que o cliente possa ver em tempo real os servidores disponiveis.

7.2 Conexão com o Servidor

Após a escolha do Ip e a conexão com o servidor estabelecida, o cliente recebe uma lista de opções com determinadas funcionalidades. Uma possível melhoria seria uma maneira automatizada de no arranque da aplicação, caso não haja instancias do servidor disponiveis, criar uma nova instancia do servidor.

7.3 Tratamento de Erros

Com um pouco mais de tempo seria possível implementar um tratamento de erros eficaz. Tanto o tratamento de erros como uma estrutura aplicacional melhorada. Não considerando um ponto de falha mas sim uma melhoria, seria uma mais valia por motivos de extensibilidade e manutenção futura do projeto, a construção de uma estrutura aplicacional mais robusta e escalável para futuras implementações. Para tal seria necessário uma análise mais aprofundada do código e uma reestruturação do mesmo. Como por exemplo, a implementação de contrato de interfaces, injeção de dependências, etc... Isto levava a um menor acopolamento entre as classes e uma maior facilidade de manutenção e extensibilidade do projeto.

Bibliography

[1] G. Cloud. Cloud functions documentation, 2024. Acessado em 29 de maio de 2024.

[2] G. Cloud. Cloud storage documentation, 2024. Acessado em 29 de maio de 2024.

[3] G. Cloud. Compute engine documentation, 2024. Acessado em 29 de maio de 2024.

[4] G. Cloud. Firestore documentation, 2024. Acessado em 29 de maio de 2024.

[5] G. Cloud. Pub/sub documentation, 2024. Acessado em 29 de maio de 2024.

[6] J. Simão, L. Assunção, and F. Passos. *Serviço Google Pub/Sub*. Instituto Superior de Engenharia de Lisboa (ISEL), 2023. Computação na nuvem, ISEL – LEIRT / LEIC / LEIM.