Relatório Preliminar

Pedro Carrega, nº49480 Vasco Ferreira, nº49470
Ye Yang, nº 49521

20 de Junho de 2020

Indíce

1 Motivação para Dataset e Serviços

O dataset Ecommerce foi escolhido pelo grupo devido ao grande número de eventos gerados e consequente informação produzida durante a utilização de uma loja de ecommerce. Informação esta que pode ser utilizada de diversas formas através de um grande número de variados serviços. Essa mesma informação poderá ser utilizada em vários contextos, sendo que escolhemos os seguintes 5 serviços que demonstram diferentes tipos de informação sobre o dataset:

- api/products/listCategories: Fornece todas as diferentes categorias presentes nos dados
- api/products/popularBrands: Fornece a contagem de eventos associados a cada marca
- api/products/salesByBrand: Lista o numero de vendas de cada marca
- api/products/salePrice: Calcula o valor médio de venda de uma determinada marca
- api/events/ratio: Apresenta a distribuição relativa de cada tipo de evento, havendo os possíveis valores: view, cart e purchase
- api/spark/svc1: Apresenta o rácio médio de tipos de evento para cada intervalo predefinido de sessão
- api/spark/svc2: Calcula a média de visualizações feitas até os clientes realizarem uma compra
- api/spark/svc3: Obtém a taxa de fidelidade média dos clientes, ou seja, a probabilidade de os clientes comprarem, adicionarem ao carrinho ou visualizarem da sua marca mais comum nessa mesma categoria

Os serviços foram escolhidos de forma a que consigam fazer diferentes tipos de operações sobre os dados, desde serviços mais específicos e por isso com menos carga na base de dados, a serviços mais abrangentes e consequente aumento de carga. Foram também escolhidos pois todos os serviços fornecem dados úteis para serem explorados no contexto de lojas de ecommerce.

2 Diagrama de Casos de Uso

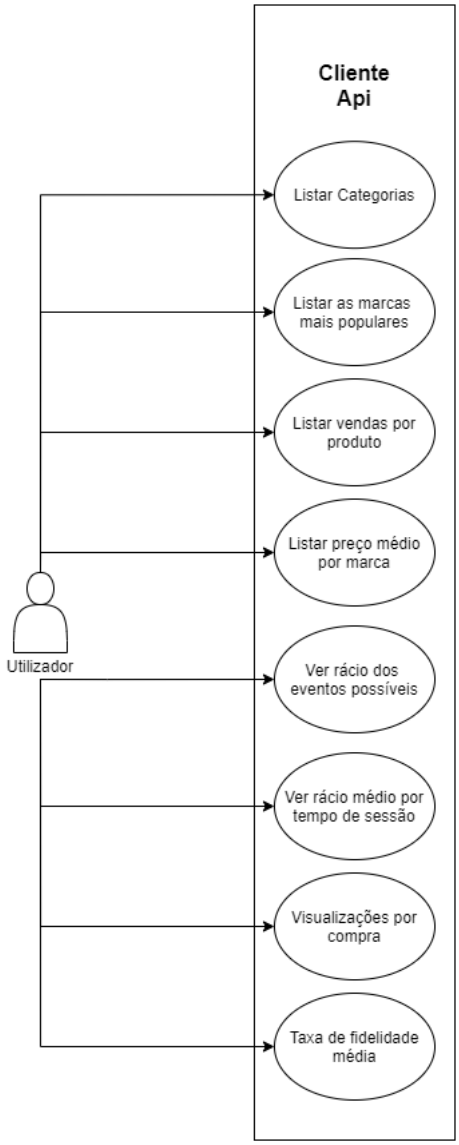

Figura 1: Diagrama de casos de uso

3 Requisitos

Requisitos Funcionais	**Descrição**
Listar Categorias Disponíveis	Serviço que fornece aos clientes todas as categorias
Visualizar Popularidade das Marcas	Serviço que fornece cada marca associada com a sua popularidade
Visualizar Número de Vendas Individuais	Fornece o numero total de vendas de cada marca
Visualizar Preço Médio de Venda	Fornece o preço médio dos produtos vendidos de uma determinada marca
Visualizar Rácio de Tipo de Eventos	Fornece a percentagem de cada tipo de evento
Rácio por período de sessão	Calcula a duração de cada sessão e qual o rácio de tipos de evento colocando a sessão com o respetivo rácio associado a um determinado intervalo de período de sessão, no fim calculando a média dos rácios para cada intervalo
Visualizações necessários para compra	Calcular o valor médio de visualizações necessárias até um cliente efetuar uma compra
Taxa Média de Preferência de Marca	Calcula a taxa de lealdade dos clientes para a sua marca mais comprada para uma determinada categoria, calculando para todos clientes e categoria. No fim devolvendo a taxa de lealdade média

Tabela 1: Requisitos Funcionais

Requisitos Não Funcionais	**Descrição**
Portabilidade	Implementação de servidor em NodeJS e cliente em HTML de modo a facilitar o processo de mudança de plataforma do serviço
Legibilidade	A separação clara entre as camadas de apresentação, lógica de negócio e acesso à base de dados irá tornar o fluxo do sistema mais legível para os desenvolvedores
Estabilidade	A distribuição do sistema por diversas máquinas virtuais, que poderão estar distribuídas por diferentes fornecedores cloud e em diferentes Data Centers permitem obter um sistema estável
Elasticidade	O sistema deverá ser capaz de se adaptar à carga de trabalho através do provisionamento e desprovisionamento dos recursos de forma autónoma. Idealmente, de forma que em qualquer ponto do tempo, o sistema apenas utilize o número de máquinas necessárias de forma a corresponder à carga atual
Escalabilidade	Capacidade do sistema lidar com o crescimento de carga de trabalho. Pode-se associar à capacidade de elasticidade do sistema
Confiabilidade	Visto o sistema estar implementado na núvem, conseguimos garantir alta confiabilidade nos sistema, pois se um servidor no datacenter falhar, conseguimos facilmente migrar a VM para outro servidor funcional

Tabela 2: Requisitos Não Funcionais

4 Arquitetura da aplicação

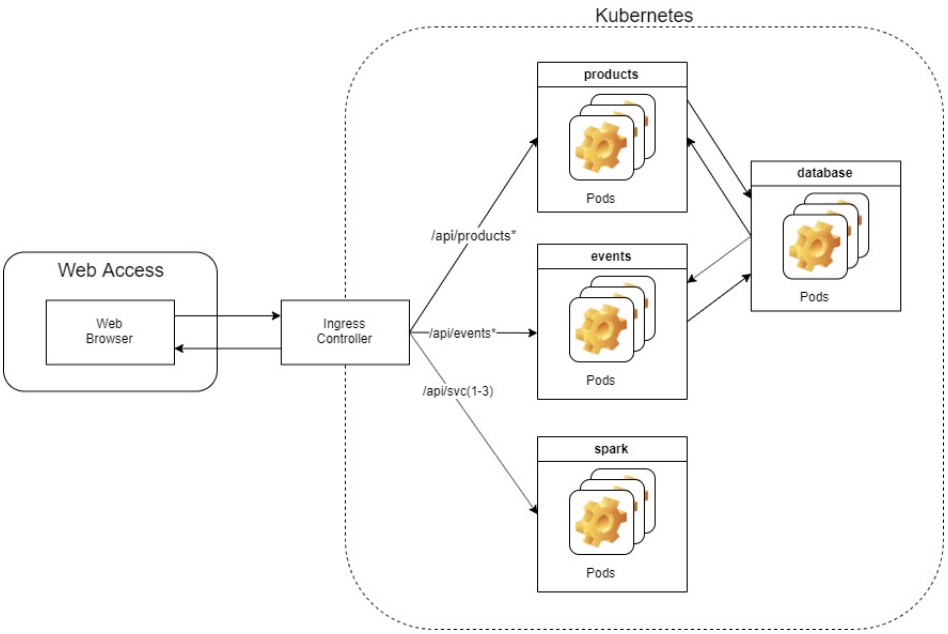

Figura 2: Arquitetura da aplicação

Após a definição da API na fase anterior, foi implementado de raiz um servidor em NodeJS que segue a API definida.

4.1 Load Balancer

Foi implementado um Load Balancer utilizando o Ingress que distribui os pedidos HTTP recebidos dos clientes browser e que reencaminha para os respetivos serviços.

4.2 Servidores NodeJS

Os servidores irão receber os pedidos a partir do Load Balancer, e processá-los de acordo com a funcionalidade pretendida. A lógica de negócio é aqui tratada, realizando as queries necessárias para a base de dados. Ao receber os dados, processa-os de acordo com a funcionalidade, e encaminha o resultado para o Load Balancer.

4.3 Base de dados

A base de dados está contida num containder de MongoDB dentro do qual se encontra todo o conteúdo do ficheiro .csv da base de dados anteriormente

escolhida.

4.4 Serviços Spark

Os serviços spark também são executados no kubernetes, porém estes usam uma série de APIs e serviços diferentes fornecidos pelo GCP que irão ser apresentados mais à frente.

5 Arquitetura técnica

Para garantir os requisitos não funcionais mencionados a cima devemos ter em atenção os serviços da cloud escolhidos, pois proporcionam vários aspetos fundamentais dos requisitos.

A utilização dos serviços disponibilizados pelo GCP permite cumprir alguns dos requisitos não funcionais mencionados, nomeadamente a Escalabilidade e a Elasticidade. Já o Load Balancer implementado com o Ingress permite que este serviço da cloud trate automaticamente da separação dos diferentes tipos de pedidos HTTP recebidos para os respetivos serviços. A base de dados utilizada consiste numa imagem Docker baseada na imagem Mongo que como o nome sugere é uma imagem de MongoDB. Foi escolhida esta alternativa de forma a não estar dependente dos serviços de Bases de Dados de um fornecedor específico, sendo que com esta opção a imagem pode ser implementada numa instância em qualquer fornecedor cloud. Quanto à escalabilidade a imagem poderia ser replicada por diversas réplicas de forma a distribuir os pedidos feitos.

A Confiabilidade do sistema também é melhorada com a instalação em serviços na cloud, visto estas terem deteções automáticas de falhas de servidores, possibilitando a migração do sistema virtualizado para outro servidor funcional.

A Estabilidade é garantida nas várias ferramentas usadas na instalação do sistema. O Load Balancer como referido anteriormente faz a distribuição dos pedidos HTTP para os respetivos serviços, sendo que o Kubernetes atribui um DNS igual para todas as réplicas do serviço de maneira a que o Load Balancer apenas reencaminhe para o respetivo DNS, sendo o Kubernetes a distribuir a carga pelas respetivas réplicas do serviço. Isto permite que o sistema escale com facilidade já que aumentar o número de réplicas de um micro-serviço em Kubernetes é um processo rápido e sem necessitar de qualquer alteração no Load Balancer.

Já a Estabilidade é garantida devido a ter a Lógica de negócio distribuída por várias máquinas virtuais que cumprem a função de servidores.

A utilização do Docker permite-nos facilmente fazer deploy de um serviço, em qualquer máquina e em qualquer um fornecedor de cloud. Isto deve-se ao Dockerfile garantir a instalação de todas as dependências do serviço, variando conforme a linguagem de implementação do mesmo, cumprindo assim o requisito de Portabilidade do sistema.

6 Implementação

6.1 Migração de AWS para GCP

Tal como anteriormente referido o desenvolvimento dos serviços em bases portáveis tornou o processo de migração da nossa implementação prévia em AWS para GCP menos demorado, ajudando no cumprimento do deadline estipulado. As bases entre ambos os fornecedores são semelhantes, mudando apenas no caso da definição de projetos (algo que não existe no AWS) e buckets na criação de buckets de armazenamento.

6.2 Base de dados

A base de dados utilizadas pela aplicação básica (serviços REST) não sofreu alterações na implementação desta fase. Porém o acesso aos dados da base de dados tornou-se diferente no caso dos serviços Spark.

Dado as limitações do PySpark em conectar e ler através de um MongoDB docker container, optamos pela leitura direta do ficheiro **.csv** dentro de um bucket privado. Para tal foi necessário através do *script* de inicialização a criação de um novo bucket num projeto do utilizador, dentro do qual é importado o ficheiro da base de dados, acedido através de um bucket publicamente disponível. Assim todas as leituras e escritas dos serviços Spark serão feitas de e para este novo bucket criado. Isto implica a necessidade de certas permissões para que um dado serviço num container consiga aceder ao bucket. Estas permissões são geradas no correr do *script* de inicialização, que cria uma nova conta admin e exporta o ficheiro dos credenciais para ser utilizado nos vários serviços.

6.3 Web Server

6.3.1 Serviços REST

O Web Server foi inicialmente implementado em NodeJS com um router Koa. Porém a documentação relativa ao router era escassa em comparação a outras alternativas, razão pela qual mudamos para um router Express que se encontrava com melhor documentação.

As queries para a base de dados na DynamoDB foram implementadas em NodeJS, onde foi necessário tratar da paginação. A documentação relativa à paginação das queries encontrava-se disponível sendo a implementação relativamente simples, excluindo as tentativas de melhorar o desempenho.

Mudando a implementação da base de dados para uma de MongoDB, deparamo-nos com pequenos problemas de implementação das queries, mas que foram eventualmente resolvidos, observando-se um aumento de desempenho a várias ordens de magnitude em comparação com a implementação na DynamoDB (queries que demoravam 10-15 minutos na Dynamo não excediam os 20 segundos no mongo).

6.3.2 Serviços Spark

Para os serviços Spark, foi feita a leitura direta ao ficheiro que contém os dados do dataset como anteriormente referido. Esta implementação foi adotada pois assumimos que os serviços Spark sendo mais computacionalmente intensivo, e pela natureza dos serviços em si (estudo de dados) não irão ser executados regularmente ao contrário dos serviços REST. Assim podemos limitar as velocidades de acesso à base de dados mas mantendo o poder computacional.

Cada serviço Spark utiliza os APIs e bibliotecas NodeJS fornecidas pelo GCP, sendo necessário os credenciais corretos para qualquer mudança sobre um projeto GCP. De seguida encontra-se um diagrama da execução de um serviço spark:

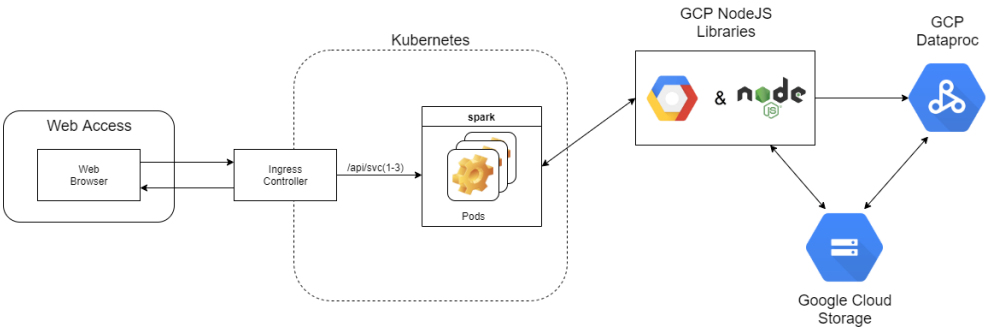

Figura 3: Execução dos serviços Spark

Os serviços no backend sempre que invocados utilizam a biblioteca de NodeJS do GCP para a criação de um cluster de Dataproc, ao qual irá ser submetido o *job* correspondente a cada um dos serviços. Estes *jobs* acedem diretamente ao ficheiro **.csv** dentro do *storage bucket* criado pelo *script* a partir do qual fazem as leituras e o processamento necessário.

Após a finalização da computação, os dados são escritos para o mesmo *storage bucket* que são finalmente lidos pelo serviço NodeJS utilizando uma API de acesso ao *storage* da Google Cloud.

6.4 Limitações do GCP

Durante a implementação do sistema, deparamo-nos com uma limitação nos recursos dos projetos do GCP, os quais que não estão em nosso controlo. Os recursos dados por defeito eram demasiado limitantes e os serviços de suporte da Google não permitiam o aumento dos limites devido a falta de dados de utilização visto as contas utilizadas serem recentes.

Para possibilitar o lançamento e o teste dos serviços tivemos que conglomerar os 3 serviços sparks num só o que limita a escalabilidade do sistema. Esta decisão teve de ser tomada pois não havia outra alternativa (mesmo com instâncias de tamanho mínimo dentro de um cluster) que permitisse o lançamento dos serviços REST juntamente com os serviços Spark (visto que cada serviço Spark cria um cluster **Dataproc** chegando rapidamente aos limites de IPs *in-use* e número de instâncias).

7 Consolidação do projeto

7.1 Terraform

Para a entrega da fase de consolidação, decidimos tornar o processo de execução e *tear-down* mais automatizada. Para tal incorporamos no *script* de deployment a utilização do Terraform.

O Terraform possibilitou-nos a criação e destruição automática dos recursos criados no projeto GCP de forma a que o processo seja mais automatizado e facilitado. Comandos como **terraform init** e **terraform destroy** permitem a criação e destruição de recursos por nós definidos dando as credenciais de GCP certas. Tal é possível devido ao suporte de serviços Google por parte do Terraform que permite aceder a vários recursos da GCP e a criação dos mesmo.

8 Execução do *script*

8.1 Requerimentos

Para a execução do *script* assume-se que a máquina local tem as seguintes ferramentas instaladas:

- Docker
- gcloud CLI
- NodeJS
- (Opcional para testar localmente) PySpark
- Terraform

8.2 Pré-preparação

Antes da execução do *script* de deployment é necessário a criação de um projeto na conta pessoal do GCP e ativar o billing.

Dado que os dados de pagamento são confidenciais, não é possível automatizar a criação de um projeto com a validação do billing.

Após criado o projeto, guardar o **project-id** atribuído.

Após criado o projeto no GCP, navegar para o ficheiro **deploy-gcp.sh** e abrir o ficheiro. Editar apenas as seguintes variáveis de ambiente definidas no ficheiros: **PROJECT_NAME** (com o id anteriormente atribuido), **BUCKET _NAME** (um nome para um *storage bucket* único) e **CLUSTER_NAME**.

8.3 Deployment do sistema

Para executar o script basta inserir no terminal: **./deploy-gcp** e seguir os passos de autenticação que irão ser pedidos.

O script irá tratar da escrita dos ficheiros de inicialização de recursos usando o Terraform, a criação e exportação dos credenciais necessários para acesso aos serviços da GCP, a escrita, montagem e deployment em containers do código fonte da camada de negócio e finalmente a criação dos pods que irão conter todos os serviços e o load balancer no cluster de Kubernetes criado.

8.4 Acesso ao serviço

Após a finalização do script de deployment, haverá um tempo de espera de cerca de **5 minutos** até que o load balancer criado acabe de realizar todos os health checks necessários. O estado dos health checks pode ser verificado com o seguinte comando:

- **kubectl describe ingress**

Após a terminção dos health checks (mensagem OK) é possível aceder aos serviços utilizando o IP gerado. O IP pode ser retirado utilizando o comando anterior, ou alternativamente através do seguinte comando:

- **kubectl get ingress**

Para aceder um serviço basta colocar o endereço IP num navegador web ou no Postman com pedido GET e completando com os URLs definidos no nosso API (ex.: <ip-add>/api/spark/svc1).

8.5 Desconstrução do deployment

Para apagar os recursos criados basta seguir os seguintes passos:

- Navegar para a diretoria do terraform criada pelo script e correr no terminal: **terraform destroy**, esperar pelo fim da execução

- Navegar para a consola do gcp e apagar o projeto inicialmente criado
- Eliminar as imagens docker locais executando **docker images** para obter os IDs de todas as imagens e **docker image rm <ID1 ID2 ... IDN>** para remover todas as imagens.

9 Cenários de Validação

- Listar Categorias:
  1. Inserir no navegador o URL anteriormente mencionado acrescentado /**api**/**products**/**listCategories**
  2. Após alguns segundos irá aparecer no ecrã o resultado.
- Listar a popularidade das marcas:
  1. Inserir no navegador o URL anteriormente mencionado acrescentado /**api**/**products**/**popularBrands**
  2. Após alguns segundos irá aparecer no ecrã o resultado.
- Listar vendas de cada marca:
  1. Inserir no navegador o URL anteriormente mencionado acrescentado /**api**/**products**/**salePrice**
  2. Após alguns segundos irá aparecer no ecrã o resultado.
- Listar o preço médio de uma marca:
  1. Inserir o url mencionado posteriormente adicionado "/<Marca>", aonde <Marca> representa a marca que o utilizador quer analisar.
  2. Após alguns segundos irá aparecer no ecrã o resultado.
- Ver o ratio dos eventos possíveis:
  1. Inserir no navegador o URL anteriormente mencionado acrescentado /**api**/**events**/**ratio**
  2. Após alguns segundos irá aparecer no ecrã o resultado.
- Portabilidade:
  1. Escolher 2 diferentes fornecedores de serviço cloud
  2. Implementar os serviços criados, nas plataformas diferentes utilizando os docker containers fornecidos.
  3. Verficar que ambos os serviços se encontram funcionais

Os docker containers tornam os sistema muito portável dado que é uma ferramenta comum entre os vários fornecedores de serviços cloud. Daí o deployment dos serviços é facilitada caso seja necessário a mudança de fornecedor, sendo apenas necessário tratar dos aspetos técnicos em termos de balanceamento de carga e quantidade de instâncias ativas nos serviços utilizando as ferramentas de cada fornecedor.

- Elasticidade:

  1. Verfique o numero de instâncias virtuais ativas
  2. Assumindo uma baixa carga de sistema, sobrecarregue o serviço com pedidos
  3. Visualize o numero de maquinas virtuais ativas aumentar de forma a poder satisfazer os diferentes pedidos sem afetar a qualidade do serviço

10 Discussão e Conclusões

Durante o desenvolvimento do projeto foram encontrados bastantes obstáculos, seja na implementação dos serviços, na escolhe e consequente aplicação da base de dados ou na automatização dos *scripts* de implementação dos serviços desenvolvidos. Todos estes problemas resultaram em muitas horas de trabalho extra por parte do grupo, mas serviram também de aprendizagem, seja em como trabalhar com fornecedores de serviços na nuvem, ou como desenvolver *scripts* de automatização de implementação. Apesar de todos os problemas, o grupo foi capaz de os solucionar de forma eficiente como mencionados na secção da Arquitetura técnica.

O DynamoDB aparenta ser uma boa solução para fazer hosting de bases de dados onde a procura seja limitada a um conjunto de valores limitados. Isto torna-o pouco útil para a nossa implementação, porém, continua a ser uma boa ferramenta para os casos mencionados se a base de dados for de dimensões elevadas, sendo difícil fazer o hosting local da mesma.

Os serviços da cloud no geral, se utilizadas de forma correta, aparentam ser poderosas em sentido computacional, tendo uma grande variedade de ferramentas que possibilita a implementação de aplicações, serviços ou outra qualquer funcionalidade.

Na nossa experiência pessoal, a documentação da AWS foi mais escassa em comparação ao GCP. Dado a escolha entre ambas recomendaríamos fortemente a utilização do GCP para futuras implementações.