Desafio

Visão Geral

O desafio proposto consiste na construção de uma solução para a consumo, processamento e disponibilização de dados existentes em bases externas.

É possível imaginar ao menos duas soluções para o desenvolvimento da solução: 1) a criação de uma aplicação web sem servidor, e 2) o desenvolvimento utilizando serviços em containers e orquestrador.

Uma solução sem servidor pode ser construída utilizando as soluções AWS Lambda e Amazon API Gateway, por exemplo. Neste cenário, uma API é disponibilizada com uma série de endpoints, que são conectados a funções que implementam as chamadas para as bases externas.

As principais vantagens deste método são a escalabilidade e o baixo custo. Em cenários de forte demanda de acesso, a própria AWS se encarrega de escalar mais containers com essas funções em seu backend. O custo de processamento do AWS Lambda não é muito alto, e além disso, funciona com cobrança sob demanda. Ou seja, no caso de baixa ou demanda inexistente, nenhuma cobrança de processamento é realizada.

A desvantagem se dá quando a estrutura está intrinsecamente acoplada a AWS. No caso de ocorrer uma alteração de estrutura de cloud, pouca coisa poderá ser aproveitada da implementação.

Já a segunda solução pode ser construída utilizando containers Docker e Cluster Kubernetes. Nesse cenário, é necessário escrever os serviços que irão acessar as bases externas e realizar o deploy em um cluster Kubernetes, onde são configurados os escaladores.

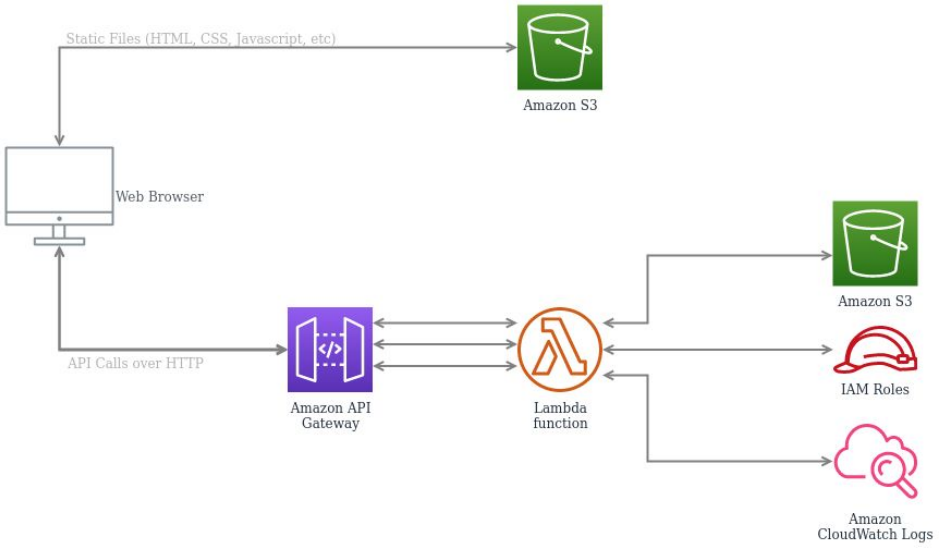
O Kubernetes pode ser definido como Cloud Agnostic, ou seja, pode ser executado em diferentes estruturas de cloud (utilizando AWS EKS ou GKE, por exemplo), sem a necessidade de adequação da solução.

Porém, os clusters Kubernetes demandam alto processamento, pois ficam constantemente em operação dentro da cloud, fazendo com que o custo de implementação aumente consideravelmente.

Dito isso, e analisando o contexto do problema acredito que, mesmo com a maior complexidade em um possível cenário de migração do serviço, a solução sem servidor é a que melhor atende imediatamente o problema proposto. Principalmente visto seu custo e baixa complexidade quando comparada a uma solução com Kubernetes. Portanto, o desenvolvimento proposto será de desenvolvimento web ser servidor.

Arquitetura da solução

A arquitetura da solução pode ser definida como uma arquitetura de nano-serviços, onde é utilizada as ferramentas AWS Lambda, o Amazon API Gateway e o Amazon Simple Storage Service. O S3 fornece os recursos para a hospedagem de um site estático que será carregado no navegador do usuário. O Javascript do navegador irá enviar e receber dados de uma API pública criada utilizando o Amazon API Gateway e o AWS Lambda. Para simplificar o problema, os dados a serem consultados pelas funções da AWS Lambda foram disponibilizados em um segundo Bucket no S3 em formato JSON.



Hospedagem de site estático

Um Bucket foi criado no S3 com permissão de acessos públicos e selecionando a opção de Hospedagem de site estático.

O frontend da aplicação foi desenvolvido em ReactJS, e o código pode ser encontrado no Github. Após entrar com o CPF do usuário e clicar em “buscar”, a aplicação realiza as chamadas para a API que foi criada com o Amazon API Gateway.

Após a realização do build para a criação dos arquivos estáticos, o deploy para a AWS foi realizado. O site para acessar a aplicação pelo navegador é:

http://creditbureau.cantu.com.s3-website-sa-east-1.amazonaws.com/

Implementação do Backend

O backend foi implementado utilizando o AWS Lambda e o Bucket criado no S3 para a armazenagem de dados. Para que as solicitações de consulta de CPF feitas no navegador sejam executadas, o Javascript irá invocar os serviços que estão sendo executados na AWS.

Armazenamento de dados

Uma série de alternativas de armazenamento de dados poderia ser adotada para resolver este problema. De acordo com as descrições da bases de dados do desafio, a Base A poderia ser implementada utilizando o Amazon RDS com um banco Postgres, por exemplo.

Já a base B poderia ser implementada utilizando um banco nosequel, como o DynamoDB, já que o retorno em JSONs também colabora para a extração de dados de Machine Learning.

Por último a base C, por ter como requisito o acesso rápido, poderia ser implementado um sistema de cache, utilizando por exemplo o Amazon ElastiCache ou um Redis.

Com a finalidade de tornar o problema de armazenamento de dados externos mais simples, e direcionar o sistema apenas com as consumações, foram definidas as três bases como sendo arquivos JSONs que foram armazenados em um Bucket não público no S3.

1. Base A: arquivo user-data.json
2. Base B: arquivo credit-score.json
3. Base C: arquivo cpf-event.json

Em comum todos os arquivos têm como chave os valores de CPFs dos usuários, definido desta forma para que o reaproveitamento de código das funções Lambda seja possível. Assim, quando uma das funções do Lambda consumirem a base de dados, será passada como chave o CPF vindo dos parâmetros da requisição.

Gerenciamento de acessos e recursos

Para cada função Lambda implementada, foi criada uma função do IAM (Identity and Access Management). Essas funções concedem permissão à função do Lambda de gravar logs no CloudWatch Logs e para acessar os arquivos do S3 em modo de leitura.

Processamento de solicitações

O AWS Lambda vai executar um código sempre que for chamado pela Amazon API Gateway, e como resposta irá retornar os dados oriundos da base externa (se existirem dados a serem retornados).

Foram criadas três funções, uma para cada base de dados, e todas foram escritas utilizando Node.Js 12.x. Para cada uma, foi associada uma regra de acesso IAM, que foi previamente criada.

1. Base A: função userData;
2. Base B: função creditScore;
3. Base C: função cpfEvent;

O código de cada função pode ser encontrado também no Github, e é o mesmo para todas as funções, com exceção apenas do nome do arquivo que é verificado no Bucket no S3 de armazenamento de dados.

As funções recebem como parâmetro um evento vindo da API, e após a coleta do CPF do usuário oriundo dos parâmetros da requisição, efetua a consulta a base de dados no S3, passando como parâmetro da função o próprio CPF.

Caso o CPF seja encontrado como chave no arquivo JSON, a função do Lambda irá retornar um body com as informações oriundas da base de dados. Caso contrário, retornará um erro 404 Not Found.

Construção da API

Para expor uma função do Lambda que já foi criada como uma API RESTful acessível pela rede pública, será utilizado o Amazon API Gateway.

O site estático hospedado no S3 é configurado para interagir com a API, onde serão enviadas requisições HTTP de método GET para obter os dados das bases externas. Para isso, foram criados três recursos, um para cada base de dados, cada qual com dois métodos: um GET e um OPTIONS.

O método GET retorna para o site estático um body com os dados da base externa (quando se aplica). O método OPTIONS teve de ser implementado em função do protocolo CORS, para que o navegador possa exibir os dados dos usuários. No cabeçalho da requisição OPTIONS são retornados, entre outros, os campos:

1. Access-Control-Allow-Headers,
2. Access-Control-Allow-Origin,
3. Access-Control-Allow-Methods",

4. X-Requested-With.

Complementos à solução

1. Amazon Cognito

Em um sistema que trabalha com dados críticos de usuários, um alto nível de segurança da informação é imprescindível. O que gostaria de ter implementado, e entendo que é importante em uma solução como está é um gerenciamento de usuários utilizando o Amazon Cognito.

Ao acessar o navegador, o primeiro passo seria o registro como novo usuário. Com isso, o Cognito enviaria um email de confirmação para esse novo usuário, para só depois realizar o login e permitir o acesso às informações disponíveis.

2. Frontend mais detalhado

Considerando o tempo hábil para resolução do desafio, optei por envidar maiores esforços na parte de desenvolvimento da solução de backend e infraestrutura, e com isso, a disposição visual dos elementos, bem como a estrutura de código do frontend tem espaço para melhora.