Desafio Engenharia de Dados

Rastreamento de entrega de pedidos - Zé Delivery

Candidato: Dheinny Vinnycius Marques

Problema proposto

O aplicativo Zé Entregador deseja implementar um novo requisito que irá rastrear onde o entregador se encontra durante a entrega de um pedido de bebidas. Para isso é preciso encontrar uma solução para receber um volume grande de coordenadas de cada entregador durante o período de entrega e direcionar para os serviços responsáveis por tratar os dados recebidos, disponibilizar o rastreamento do pedido e notificar o cliente da aproximação ou chegada do entregador.

As seguintes perguntas devem ser respondidas:

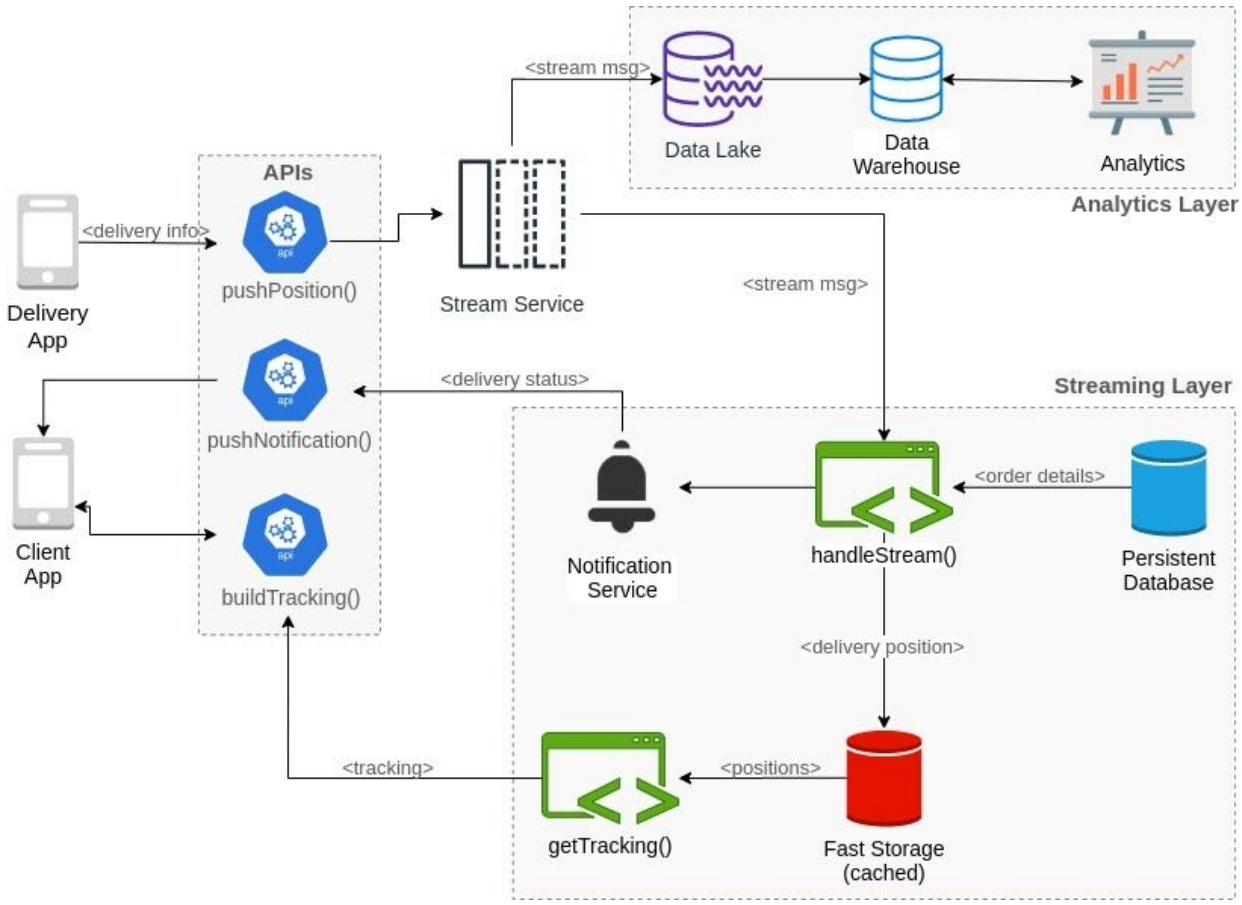
1. Como receber todos os dados de localização do aplicativo do entregador? Quais protocolos, serviços, componentes será usados para receber, armazenar e disponibilizar os dados para serem usados? Deve-se:
    1.1. Usar API?
    1.2. Usar componentes gerenciados?
    1.3. Usar filas, mecanismos pub-sub, serve
2. Enquanto recebe-se um dado de localização associado com informações do pedido, se for preciso ingerir esses dados extras:
    2.1. Como pode ser feita essa ingestão?
    2.2. Em qual camada essa ingestão será feita?
3. É preciso criar uma API para recuperar a última posição da localização do pedido, disponível:
    3.1. Como pode ser feito essa API?
    3.2. Como armazenar a última localização do pedido, enquanto se recebe os dados de localização?
4. Como implementar um sistema de notificação sobre o status do pedido, enquanto coleta-se os dados de localização?
5. Como armazenar os dados de forma a disponibilizá-los para o time de Data Analytics realizar pesquisas sobre eles?

Para responder às perguntas acima, será apresentado na sequências uma arquitetura de sistema que se propõe a resolver os problemas encontrados.

Arquitetura

Arquitetura Geral

Para melhor entendimento, primeiro foi desenhado uma solução abstrata, com foco nos componentes envolvidos na solução, sem especificar nenhum serviço a ser utilizado:



O modelo proposto baseia-se em uma API de serviços para a comunicação com os aplicativos do entregador e do cliente; um serviço de stream que ficará responsável para receber o fluxo de dados com as localizações dos entregadores durante suas entregas; uma camada Analítica e camada de Stream, que lidará com os dados que chegam em tempo real.

APIs

A princípio 3 serviços serão disponibilizados na camada de API:

1. pushPosition(): Usado para receber a localização periódica do entregador. Ao receber a mensagem com a localização, o serviço direciona a mensagem para o serviço de Stream.
2. pushNotification(): Responsável por enviar notificações de status de entrega para o cliente.
3. buildTracking(): Enviará para o aplicativo do cliente os dados de rastreamento do pedido. Poderá enviar todas as localizações daquele pedido, bem como apenas a última localização.

Serviço de Stream

Um serviço de Stream será utilizado para receber o fluxo de dados com a localização dos entregadores e o orquestrará dentro da aplicação.

Ele será responsável por 2 tarefas principais:

1. Armazenar os dados no data lake da companhia.
2. Disparar a função de tratamento dos dados de localização

Camada Analítica

Essa camada é onde os dados serão armazenados num componente de persistência de dados e ficará disponível para consultas analíticas bem como para outros possíveis usos que venham a ser necessários.
Um data warehouse será um componente que realizará as consultas exigidas pelas aplicações analíticas, no data lake.

Camada de Streaming

Nessa camada ocorrerá o processamento do fluxo de dados das localizações dos entregadores.

Primeiramente o serviço de Stream acionará a função handleStream(), passando a mensagem com os dados de localização.

Aos receber essa mensagem a função primeiro armazena esses dados em um componente de armazenamento de acesso rápido(fast storage), para que ao ser consultado, a latência de resposta seja baixa, uma vez que se trata de buscar a informação em tempo real para ser exibido no aplicativo do cliente.

Após gravar as localizações, a função faz uma consulta em um banco de dados que armazena os detalhes do pedido, para obter a informação de destino da entrega. De posse dessa informação, a função realiza uma comparação com a posição atual do entregador para decidir se manda uma notificação de status de entrega pro cliente, de acordo com a política de notificação definida. Quando uma notificação deve ser realizada a função faz o uso de um serviço de notificações que ficará responsável por disparar esse envio.

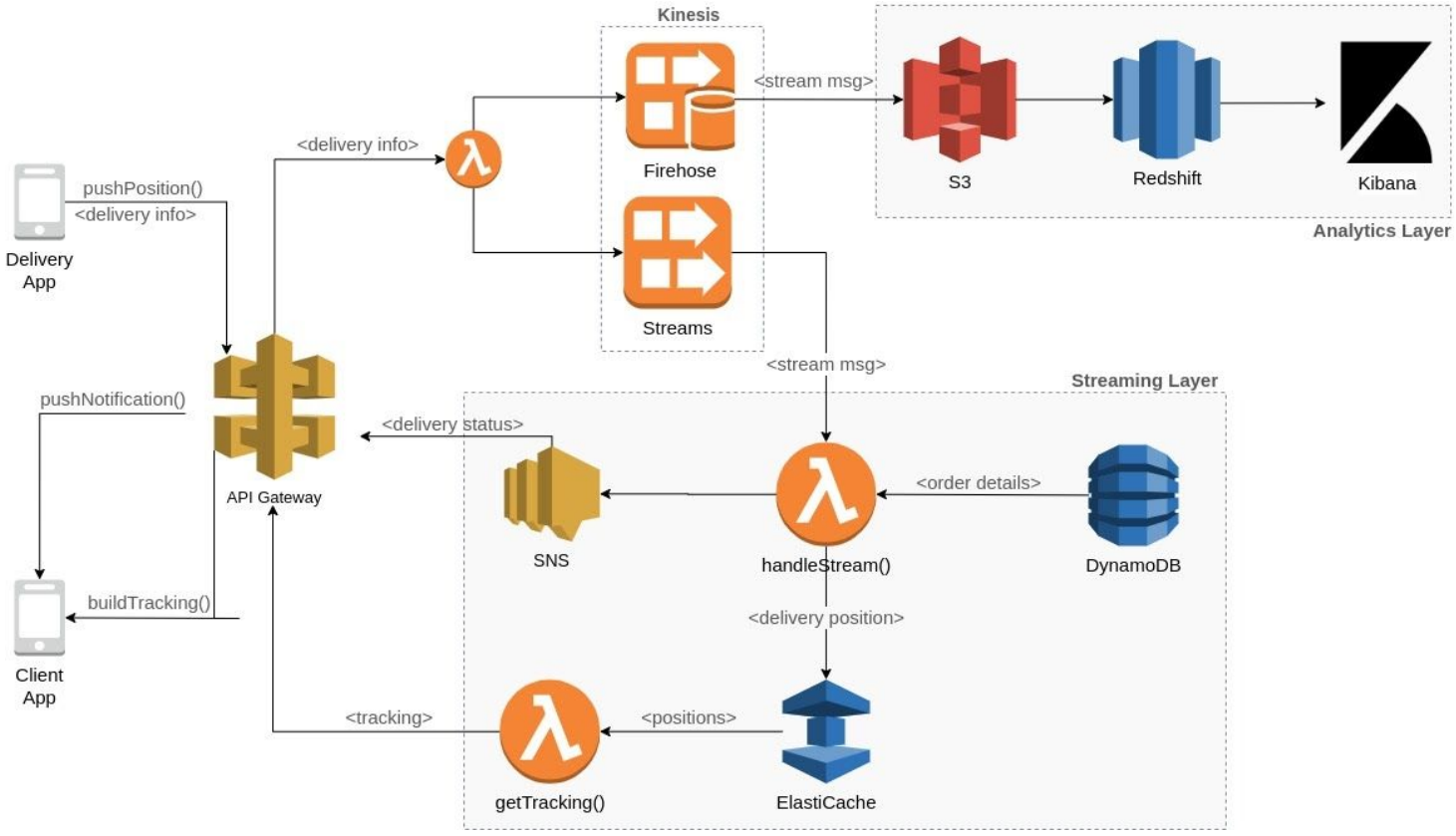
Com as localizações gravadas no fast storage, a função getTracking() pode consultá-las e enviar para o cliente através da API buildTracking(), descrita anteriormente no tópico de APIs

Portanto, essa é a arquitetura geral da solução proposta.

Alguns detalhes a serem considerados:

1. O componente de fast storage foi pensado para ser usado como armazenamento temporário e de rápido acesso para um cenário onde os volumes de dados sejam extremamente altos, e a latência para recuperá-los de um bando de dados esteja atrapalhando o desempenho e a experiência do uso da aplicação.
2. Em um cenário menos adverso, onde a latência para recuperação das localizações não esteja comprometendo o desempenho do sistema e onde se queira armazenar os dados de rastreamento de forma mais persistentes para acesso das aplicações do sistema, o mesmo banco utilizado para armazenar os detalhes do pedido podem ser usado para armazenar as localizações do trajeto da entrega, abrindo mão do componente de fast storage.

Arquitetura baseada em Serviços AWS



A arquitetura apresentada acima é baseada na arquitetura geral especificando os serviços da AWS apropriados para cada componente.

APIs

Na camada de APIs, o API GATEWAY será o serviço responsável por atender requisições e fazer a comunicação entre o sistema e os aplicativos.

Serviço de Stream

O serviço de stream utilizado nessa arquitetura será o Kinesis.

A vantagem de usar o Kinesis é a capacidade que ele tem de consumir, armazenar em buffer, e processar dados de streaming em tempo real.

Ele também é totalmente gerenciado, sem a necessidade de gerenciamento de qualquer infraestrutura. E ele é escalável, se adaptando às necessidades do momento da aplicação.

Para entregar os dados puros ao data lake, será usado o serviço Kinesis Firehose e para direcionar o fluxo de mensagens para a função que tratará as informações, será feito o uso do Kinesis Streams.

Camada Analítica

Na camada analítica, o data lake será composto pelo S3, serviço de armazenamento da AWS. Ele armazenará permanentemente todos os dados de localização das entregas.

E através do Redshift + Kibana o time de Análise de Dados poderá ter acessos a essas informações e utilizá-las para realizar todas as análises necessária.

Camada de Streaming

Indo para a camada de streaming, temos como serviço principal o lambda handleStream(). Através do mapeamento de fontes de eventos do serviço Lambda, é possível que fazer com q o serviço se associe ao Kinesis Streams para ficar o observando a espera de novas mensagens. Ao chegar uma nova mensagem o serviço lambda chama o handleStream() irá tratá-la adequadamente.

Como banco de dados persistente, foi proposto o uso do DynamoDB, que é um banco escalável e de baixa latência. O principal uso dele nessa arquitetura será armazenar os detalhes do pedido, como id e local de entrega, entre outros. Porém, como veremos abaixo, também é possível usá-lo como substituto do ElastiCache como local para armazenar as localizações dos entregadores durante a entrega.

Continuando com o handleStream() ao ser acionado, ele desempacotará a mensagem recebida pelo Kinesis Streams, e armazenará a localização atual no ElastiCache, que é um banco de armazenamento temporário em cache, extremamente rápido, que visa diminuir a latência de consulta do sistema. Essa solução foi baseada no cenário mais extremo, que pode acontecer, uma vez que o espera-se um aumento exponencial no volume de dados de

localização, porém o uso do ElastiCache deve analisado com cautela, pois é um serviço sensivelmente mais caro que do que outros serviços de armazenamento persistente, como o DynamoDB, por exemplo. Portanto, se o volume de dados não for suficiente para comprometer a latência do sistema, pode ser mais apropriado utilizar o próprio DynamoDB para armazenar os dados de localização de entrega.

Outros pontos podem ser considerados aqui, como a necessidade de permanência desses dados no banco, ou a necessidade de rodar um serviço que de tempos em tempos apague localizações mais antigas, uma vez que esses dados já estão persistidos no data lake, no S3. Essa decisão depende do modelo de negócio definido.

Voltando ao fluxo do handleStream(), após armazenar os dados de localização, essa função recupera a informação dos detalhes da entrega, para obter a localização do endereço de entrega, e realiza uma comparação com a posição atual do entregador, para decidir se uma notificação com o status da entrega deve ser emitida para o cliente.

Para o sistema de Notificações, será usado o SNS, que pode enviar mensagens para tópicos, onde todos aplicativos que estiverem inscritos nele recebem a notificação, ou diretamente ao aplicativo do cliente, através de um endpoint móvel.

Enfim, com as localizações armazenadas no local apropriado, a API que monta o percurso da entrega pode consultá-las e usá-las para informar ao cliente onde seu pedido se encontra.

POC - Camada de Streaming

Para demonstrar parte dessa arquitetura na prática, foi desenvolvida uma POC (Proof of Concept) da camada de Streaming no ambiente da AWS, usando o console web da plataforma.

Para essa demonstração não foram criadas as APIs.

As mensagens estão sendo enviadas para o Kinesis Streams diretamente pela sua API. Após o envio das mensagens o sistema é todo orquestrado automaticamente, sem precisar de qualquer interferência.

Nela também foi usado o DynamoDB, como armazenamento das localizações, no lugar do ElastiCache, pelos motivos descritos anteriormente.

E para envio de solicitação foi criado um tópico no SNS.

Para ter acesso ao ambiente, foi criado um usuário na minha conta AWS. Para se logar use dos dados abaixo:

Navegue até o console da AWS pelo seguinte endereço:
http://console.aws.amazon.com/

Account ID: 155945245308
IAM user name: poc_viewer
Password: senha123

Esse usuário tem acesso para visualizar os seguintes serviços?

- Kinesis
- Lambda
- DynamoDB
- SNS
- CloudWatch Logs

No console do Lambda estará listada a função lambda que manipula a mensagem recebida, salvando as localizações no lugar adequado e enviando notificações quando necessário. Entrando nela é possível ver o código da função (que também está no repositório git com a resolução desse teste).

Para executar o fluxo de dados também foi criado um usuário para executar linhas de comando através do AWS CLI (https://docs.aws.amazon.com/pt_br/cli/latest/userguide/install-cliv2.html).

Para enviar mensagens para o Kinesis é necessário configurar as seguintes chaves nas credenciais da aws através do comando

```
$ aws configure
```

Uma vez configurada as chaves de acesso, entre com os comandos abaixo para simular o envio de 5 mensagens de localização ao Kinesis Streams e iniciar o fluxo:

```
$ aws kinesis put-record --stream-name tracking-challenge
--partition-key 1 --data
"eyJvcmRlcl9pZCI6IDEsICJsb2NhdGlvbiI6IHsiY29vcmRfeCI6IDIsICJjb29yZF95
IjogM319"

$ aws kinesis put-record --stream-name tracking-challenge
--partition-key 1 --data
"eyJvcmRlcl9pZCI6IDEsICJsb2NhdGlvbiI6IHsiY29vcmRfeCI6IDMsICJjb29yZF95
IjogNH19"

$ aws kinesis put-record --stream-name tracking-challenge
--partition-key 1 --data
"eyJvcmRlcl9pZCI6IDEsICJsb2NhdGlvbiI6IHsiY29vcmRfeCI6IDYsICJjb29yZF95
IjogNX19"

$ aws kinesis put-record --stream-name tracking-challenge
--partition-key 1 --data
"eyJvcmRlcl9pZCI6IDEsICJsb2NhdGlvbiI6IHsiY29vcmRfeCI6IDgsICJjb29yZF95
IjogN319"

$ aws kinesis put-record --stream-name tracking-challenge
--partition-key 1 --data
"eyJvcmRlcl9pZCI6IDEsICJsb2NhdGlvbiI6IHsiY29vcmRfeCI6IDksICJjb29yZF95
IjogOH19"

$ aws kinesis put-record --stream-name tracking-challenge
--partition-key 1 --data
"eyJvcmRlcl9pZCI6IDEsICJsb2NhdGlvbiI6IHsiY29vcmRfeCI6IDEwLCAiY29vcmRf
eSI6IDd9fQ=="
```

O resultado da operação pode observado no CloudWatch logs. E os registros dos dados podem ser vistos na tabela **order_track** no console do DynamoDB.

Respostas às Perguntas do Desafio

Voltando às perguntas citadas na proposta de desafios, a maioria delas já foi respondidas indiretamente durante a explicação da arquitetura proposto.

Contudo, segue as respostas diretas para o desafio:

1. Como receber todos os dados de localização do aplicativo do entregador? Quais protocolos, serviços, componentes será usados para receber, armazenar e disponibilizar os dados para serem usados? Deve-se:
   1.1. Usar API?
   Para ter uma solução desacoplada da arquitetura, o ideal é usar uma API para receber as requisições com as localizações do aplicativo do entregador. Porém, o serviço de stream do Kinesis permite enviar mensagens diretamente para ele. Portanto, dependendo da necessidade pode se abrir mão do desacoplamento e fazer o aplicativo mandar as mensagens direto para o Kinesis, diminuindo a latência de envio.

   1.2. Usar componentes gerenciados?
   Sim, será utilizado o Kinesis que é um componente gerenciado de Stream associado ao serviço de Lambda.

   1.3. Usar filas, mecanismos pub-sub, serveless
   Não será usado serviço de filas e sim de stream e serverless com o uso do Kinesis e Lambda.

2. Enquanto recebe-se um dado de localização associado com informações do pedido, se for preciso ingerir esses dados extras:
   2.1. Como pode ser feita essa ingestão?
   A função lambda que será acionada sempre que uma mensagem com dados de localização for adicionado ao Kinesis poderá lidar com a ingestão de dados adicionais que vierem junto com a mensagem. Nesse ponto poderá ser tratado o dado e armazenado no lugar necessário.

   2.2. Em qual camada essa ingestão será feita?
   Na camada de Streaming da arquitetura proposta.

3. É preciso criar uma API para recuperar a última posição da localização do pedido, disponível:
   3.1. Como pode ser feito essa API?

É possível criar um lambda e disponibilizá-lo como API através do serviço de API Gateway. Esse lambda ficará responsável por receber a requisição, buscar a informação referente a última localização e devolver como resposta da requisição.

3.2. Como armazenar a última localização do pedido, enquanto se recebe os dados de localização?

Criando um campo para armazenar a última localização na tabela de detalhes do pedido, e atualizando ele no lambda que é acionado pelo Kinesis.

4. Como implementar um sistema de notificação sobre o status do pedido, enquanto coleta-se os dados de localização?

Utilizando o serviço de gerenciamento de notificações SNS. Esse serviço permite ou criar um tópico, onde aplicativos podem se inscrever para receber as notificações enviadas no tópico ou então enviar mensagem diretamente ao aplicativo do cliente, através de um endpoint móvel.

5. Como armazenar os dados de forma a disponibilizá-los para o time de Data Analytics realizar pesquisas sobre eles?

Os dados serão persistidos em um bucket do S3. Utiliza-se então o Data Warehouse Redshift para fazer consultas de forma otimizada aos dados armazenados no bucket, podendo ser usado por ferramentas de Análise de Dados como o Kibana, por exemplo.

Conclusão

É possível construir uma solução completa para o problema proposto usando apenas serviços disponíveis na plataforma da AWS. Essa solução tem a vantagem de pode se integrar com diversos outros possíveis serviços da AWS para a evolução do produto, sem a necessidade de criar uma infraestrutura nova ou separada.