Prática de sistemas distribuídos

Chauã Queirolo

1 Questões

1. Ainda é possível registrar transações quando uma das réplicas do Cassandra é desligada? E duas? Por quê?

Sim, ainda é possível. Nos testes realizados, as transações foram salvas mesmo que uma ou duas réplicas fossem desligadas. Isto ocorre porque o Cassandra distribui a gravação dos dados entre diversos nós, com base em um fator de replicação. Cada nó fica com um subset dos dados do dataset, que são replicados entre os nós. Deste modo, mesmo que um nó fique indisponível, ainda é possível armazenar e recuperar os dados com um certo grau de confiabilidade.

2. Quando o Redis está indisponível, ainda é possível consultar transações? Justifique.

Sim, ainda é possível. O Redis é um banco de dados em memória usado para realizar cache de transações. Ele é útil para colocar em cache consultas de alta demanda, permitindo assim respostas mais rápidas para o cliente e desafogar o número de transações de consultas ao banco de dados.

3. Caso o serviço transactions esteja fora do ar quando uma nova transação for publicada no kafka pelo bff, a transação será perdida? O que acontece quando transactions voltar ao ar?

Não, a transação não será perdida. O kafka trabalha com o modelo de *pull*. Todas as mensagens enviadas pelo bff ficam esperando na fila até o momento que o transactions esteja disponível para consumir a próxima mensagem. O kafka utiliza o zookeeper para armazanenar todos os eventos que são recebidos.

4. Quais são as vantagens de transmitir os dados em Avro em vez de JSON do bff para o Kafka?

Em avro, os dados são armazenados em formato binário, sendo recuperados através de um esquema previamente definido. O formato binário ocupa um tamanho menor de bytes, quando comparado com a representação em formato texto do JSON. Desta forma, temos ganho de velocidade no tráfego de informações pela rede e menor consumo de memória e disco.

5. O Kafka, ao possibilitar o processamento assíncrono de transações pelo serviço transactions, permite com que o bff consiga responder com um throughput maior, pois não é preciso sofrer a latência do Cassandra e antifraud. Porém isso também traz suas desvantagens para a arquitetura. Liste duas desvantagens e justifique.

Um problema pode ocorrer quando o número de transações geradas seja muito maior que o transactions possa processar. Dependendo do tipo de aplicação, a demora da resposta do transactions de volta para o bff pode gerar do lado do cliente. Por exemplo, uma aplicação de ponto de venda onde o vendedor está na frente do cliente esperando que a venda seja efetivada. Caso o sistema demore muito para responder, o cliente pode se aborrecer e cancelar a compra. Uma solução seria tentar escalonar o serviço de transactions para que o tempo de resposta fique próximo ao tempo de resposta sícrono.

6. Analise o código do serviço de transactions. Quando o serviço antifraud estiver desligado, ainda é possível registrar transações? Qual vai ser o status final das transações nesse caso?

Sim, ainda é possível. Neste caso, as transações são registradas como "*rejected*". O log do transactions registra a seguinte mensagem de erro:

```
failed to contact antifraud service: <_InactiveRpcError of RPC
↪  that terminated with:
        status = StatusCode.UNAVAILABLE
        details = "upstream connect error or disconnect/reset
        ↪  before headers. reset reason: connection failure"
```

```
debug_error_string = "{"created":"@1657672901.867635006","
↪ description":"Error received from peer
↪ ipv4:172.18.0.9:5005","file":"src/core/lib/surface/cal
↪ l.cc","file_line":1066,"grpc_message":"upstream
↪ connect error or disconnect/reset before headers.
↪ reset reason: connection failure","grpc_status":14}"
```

7. Quais são as duas vantagens e desvantagens de utilizar o envoy como load balancer no contexto da interação entre os serviços transactions e antifraud?

Uma vantagem é que o transactions não precisa conhecer todas as instâncias de antifraud que estão em execução. Ele apenas direciona a requisição para o envoy que dispara para alguma instância disponível. Se tivermos 4 ou 10 instâncias rodando, o transactions não precisa conhecer nenhuma delas.

Outra vantagem é que o envoy gerencia o balanceamento das cargas, evitando que todas requisições sejam concentradas em um único serviço. Por exemplo, ele pode utilizar a estratégia round robin para distribuição das requisições.

8. O código de enviar uma notificação de status por webhook para o bff está estruturado assim:

```python
def notify_status(transaction_id, status):
    try:
        resp = requests.patch(
            f"{BFF_HOST}/api/v1/transactions/{transaction_id}/
                status",
            json={"status": status},
        )

    except Exception as err:
        logging.error(f"failed to update status: {str(err)}")
        raise BFFStatusWebhookError(Exception)
```

8.1 Modifique esse código para que quando não for possível se comunicar com o serviço bff, continuar tentando enviar por mais 10 vezes antes de retornar um erro.

```
from retry import retry

@retry(BFFStatusWebhookError, delay=3, tries=10)
def notify_status(transaction_id, status):
    try:
        resp = requests.patch(
            f"{BFF_HOST}/api/v1/transactions/{transaction_id}/
                status",
            json={"status": status},
        )

    except Exception as err:
        logging.error(f"failed to update status: {str(err)}")
        raise BFFStatusWebhookError(Exception)
```

8.1 Ainda nesse contexto de interação entre os serviços transactions e bff, quais são as desvantagens de receber o status por webhook no bff?

Uma desvantagem do uso de webhooks é que não é possível saber quando o sistema de retorno caiu. Como o transactions apenas envia o status, seria necessário criar um outro mecanismo de controle para avisar o bff quando ele retornar para operação. Neste caso, o problema do uso de webhooks é que o bff pode não receber a notificação que a transação foi efetivada, e com isso ele mandar a mesma transação ou uma transação nova para processamento.

9.Cite algumas diferenças entre Protocol Buffers e Avro.

Em Avro, a tipificação dos dados é dinâmica. Os dados são sempre acompanhados do seu esquema. Isto permite o processamento dos dados sem a geração de código estático. Além disso, como o esquema está presente junto com os dados, são necessárias menos informações para sua codificação.

Por outro lado, em Protocol Buffers a tipificação é estática. Os esquemas precisam ser compilados para gerar código nativo para alguma linguagem de programação específica. Protocol Buffers também possui suporte a alguns tipos de dados complexos, como timestamp, os quais não são suportados em Avro.

Outra diferença, em Avro todos campos são obrigatórios, enquanto Protocol Buffers possui suporte a campos opcionais.