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

Tem 3 réplicas do `Cassandra`. Estao nos containers `scylla-1`, `scylla-2` e `scylla-3`.

```
docker kill scylla-1
```

```
curl -X POST http://localhost:5001/api/v1/transactions \
    --data '{"value_in_cents":39859385,"description":"teste sem scylla-
1","customer_id":"c408a342-5c78-4e31-afc8-c7c710b07340","merchant_id":"0f3eaa5d-
79eb-48bc-ad0d-
c775efa3646e","transaction_timestamp":1624079756,"latitude":93.1334,"longitude":12
.0445}' \
    -H "Content-Type: application/json"
```

Deu esse output:

```
{"transaction_id":"16f051bb-14e0-4fda-95e5-507d01ea37dc"}
```

```
docker exec -ti scylla-2 cqlsh
```

```
USE transactions;
SELECT * FROM transactions
WHERE transaction_id = '16f051bb-14e0-4fda-95e5-507d01ea37dc';
```

Deu esse output:

```
transaction_id                       | customer_id                          |
description        | event_timestamp | latitude | longitude | merchant_id
| status   | transaction_timestamp | value_in_cents
--------------------------------------+--------------------------------------+----
---------------+-----------------+----------+-----------+----------------------
-------------+----------+----------------------+---------------
 16f051bb-14e0-4fda-95e5-507d01ea37dc | c408a342-5c78-4e31-afc8-c7c710b07340 |
teste sem scylla-1 |      1624722171 |  93.1334 |   12.0445 | 0f3eaa5d-79eb-48bc-
ad0d-c775efa3646e | accepted |            1624079756 |       39859385
```

Com uma réplica do `Cassandra` desligada, ainda é possivel registrar transações.

Agora tentamos desligar tambem a réplica `scylla-2`:

```
docker kill scylla-2
```

```
curl -X POST http://localhost:5001/api/v1/transactions \
    --data '{"value_in_cents":39859385,"description":"teste sem scylla-1 e scylla-2: so a réplica scylla-3 rodando","customer_id":"c408a342-5c78-4e31-afc8-c7c710b07340","merchant_id":"0f3eaa5d-79eb-48bc-ad0d-c775efa3646e","transaction_timestamp":1624079756,"latitude":93.1334,"longitude":12.0445}' \
    -H "Content-Type: application/json"
```

Deu esse output:

```
{"transaction_id":"7a63d59c-c6a0-4195-a478-e01933e6f067"}
```

```
docker exec -ti scylla-3 cqlsh
```

```
USE transactions;
SELECT * FROM transactions
WHERE transaction_id = '7a63d59c-c6a0-4195-a478-e01933e6f067';
```

Deu esse output:

```
transaction_id                       | customer_id                          |
description                                                  | event_timestamp |
latitude | longitude | merchant_id                          | status   |
transaction_timestamp | value_in_cents
--------------------------------------+--------------------------------------+----
-------------------------------------------------------------+-----------------+-----
-----+-----------+--------------------------------------+----------+-------------
---------+---------------
 7a63d59c-c6a0-4195-a478-e01933e6f067 | c408a342-5c78-4e31-afc8-c7c710b07340 |
teste sem scylla-1 e scylla-2: so a réplica scylla-3 rodando |      1624722819 |
93.1334 |   12.0445 | 0f3eaa5d-79eb-48bc-ad0d-c775efa3646e | accepted |
1624079756 |       39859385
```

Com 2 réplicas do `Cassandra` desligada, ainda é possivel registrar transações.

O `Cassandra` armazena réplicas de dados em vários nós para garantir confiabilidade e tolerância a falhas. Assim a gente viu que até com 2 nós mortos, ainda é possivel acessar os dados por conta da terceira réplica `Cassandra`.

Agora a gente reexecuta o `docker-compose up` para poder responder as outras questoes:

```
docker-compose down
docker-compose up
```

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

Com o `Redis` disponivel:

```
curl -X GET http://localhost:5001/api/v1/transactions | jq
```

Deu esse output:

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1845  100  1845    0     0  16621      0 --:--:-- --:--:-- --:--:-- 16621
{
  "transactions": [
    {
      "customer_id": "b2ee7f5f-5a30-4f8f-bf1b-b39de168c0da",
      "description": "some purchase",
      "event_timestamp": 1624724275,
      "latitude": 5.2295059260530685,
      "longitude": 22.88748010417722,
      "merchant_id": "8c6e4e65-32b4-4e94-b8bc-1098b6d17950",
      "status": "rejected",
      "transaction_id": "f449f21a-f4e5-48e5-bb35-fde21450681a",
      "transaction_timestamp": 1242547325,
      "value_in_cents": 1296489851
    },
    {
      "customer_id": "2d79c2ef-9ca6-4062-a1d7-e0d884b43b64",
      "description": "some purchase",
      "event_timestamp": 1624724266,
      "latitude": 10.480258837364037,
      "longitude": 11.313191857151939,
      "merchant_id": "2d3b58db-c8c5-48cf-91e0-b009da3bb5ca",
      "status": "accepted",
      "transaction_id": "f69d2b50-2f87-414d-aaf2-83ec7b1aeb15",
      "transaction_timestamp": 1422157435,
      "value_in_cents": 557635095
    },...
```

Agora com o `Redis` indisponível:

```
docker kill redis
```

```
curl -X GET http://localhost:5001/api/v1/transactions | jq
```

Deu esse output:

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    22  100    22    0     0      1      0  0:00:22  0:00:16  0:00:06     5
{
  "transactions": null
}
```

No codigo, na funçao `register_transaction`, pedimos pro `Kafka` publicar a transaçao. Assim que ele conseguir, ele ja escreve no cache do `Redis`:

```
try:
    transaction_client.create(txn)
    cache.add_transaction(txn)
```

E nessa hora, quando dou o **GET** nessa requisiçao, eu nao consulto o banco, mas o cache do `Redis`.
Por isso com o `Redis` indisponível nao é mais possivel consultar transaçoes.

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?

```
docker kill transactions
```

```
curl -X POST http://localhost:5001/api/v1/transactions \
    --data '{"value_in_cents":39859385,"description":"teste com o serviço
transactions fora do ar","customer_id":"c408a342-5c78-4e31-afc8-
c7c710b07340","merchant_id":"0f3eaa5d-79eb-48bc-ad0d-
c775efa3646e","transaction_timestamp":1624079756,"latitude":93.1334,"longitude":12
.0445}' \
    -H "Content-Type: application/json"
```

Deu esse output:

```
{"transaction_id":"dfcd7893-4fb5-4839-91d5-b2a250862d30"}
```

```
docker exec -ti scylla-1 cqlsh
```

```
USE transactions;
SELECT * FROM transactions
WHERE transaction_id = 'dfcd7893-4fb5-4839-91d5-b2a250862d30';
```

Deu esse output:

```
transaction_id | customer_id | description | event_timestamp | latitude |
longitude | merchant_id | status | transaction_timestamp | value_in_cents
---------------+-------------+-------------+-----------------+----------+--------
---+-------------+--------+----------------------+---------------
```

Depois que a gente reativou o serviço `transactions`, a mesma query no `Cassandra` deu o resultado seguinte:

```
transaction_id                       | customer_id                           |
description                                | event_timestamp | latitude |
longitude | merchant_id                          | status   |
transaction_timestamp | value_in_cents
--------------------------------------+---------------------------------------+----
----------------------------------------+-----------------+----------+-----------
+--------------------------------------+----------+----------------------+-------
---------
 dfcd7893-4fb5-4839-91d5-b2a250862d30 | c408a342-5c78-4e31-afc8-c7c710b07340 |
teste com o serviço transactions fora do ar |      1624732633 |  93.1334 |
12.0445 | 0f3eaa5d-79eb-48bc-ad0d-c775efa3646e | accepted |            1624079756
|       39859385
```

E podemos observar nos logs do `Kafka` que todas as transaçoes que foram feitas, quando o serviço `transactions` estava fora do ar, sao consumidas pelo mesmo serviço quando ele volta no ar.
Ou seja, `Kafka` se comporta como uma fila de mensagens (*message queue*) e permite ao serviço `transactions` executar a logica dele (salvar no `Cassandra` e no `Redis`) nas transaçoes na ordem de chegada. o `Kafka` tambem permite o serviço recuperar de uma falha sem perder nenhuma das transaçoes feitas quando ele estava fora do ar.

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

O Avro usa JSON para definir tipos de dados e protocolos e serializa dados em um formato binário compacto. A vantagem do JSON é de ser facil para humanos ler e escrever.

- Agora, é obvio que para o dado ser transmitido, o formato Avro, mais compacto, é mais relevante. Pois isso acelera o processo de transmissao: o fluxo de dado sera mais compacto.
- O `Kafka` tambem nao tem uma capacidade muito grande de armazenamento, pois nao é uma banco de dados. Ou seja, quando o serviço na saida do `Kafka` (aqui é o `transactions`) estiver com problema de latencia ou estiver fora do ar, o dado armazenado temporiaramente pelo `Kafka` precisa ser num formato mais compacto. A vantagem do Avro comparado a um outro formato é que a conversao do JSON pro Avro ou do Avro pro JSON é mais rapida, devido a uma estrutura parecida.

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.

- Uma primeira desvantagem é a necessidade de comprimir e descomprimir o fluxo de dados na entrada e na saida do `Kafka` o que afecta a perfomance.
- Se o `Kafka` falhar, todos os serviços que dependem da fila de logs do Kafka vao parar de receber os requests.
- E apesar disso o `Kafka` não contém um conjunto completo de ferramentas de monitoramento e gerenciamento, entao as falhas no `Kafka` sao dificilmente monitoraveis.

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?

Tem 2 containers antifraud: `antifraud-1` e `antifraud-2`.
Os 2 fazem executam o mesmo codigo.
Porque ter 2 serviços antifaud? A verificaçao do status da transaçao demora entao é melhor balancear o fluxo das transaçoes com 2 serviços em vez de apenas 1.

Se a gente desligar `antifraud-1`:

```
docker kill antifraud-1
```

Entao as transaçoes ainda estao registradas, mas um em cada 2 logs chegando vem assim:

```
new transaction arrived: {'transaction_id': '2ea2474e-4947-4990-8848-
5edee8eb9e4b', 'value_in_cents': 998750637, 'description': 'some purchase',
'customer_id': '1dd55167-0cfc-48eb-b97c-7f8ddd9b1ba4', 'merchant_id': 'd30caa48-
f1f4-49f6-b456-cac9ea2ee959', 'transaction_timestamp': 1494552745,
```

```
'event_timestamp': 1624739064, 'latitude': 1.6733094453811646, 'longitude':
29.754209518432617}
antifraud-2   | WARNING:tensorflow:Model was constructed with shape (None, 1, 4)
for input KerasTensor(type_spec=TensorSpec(shape=(None, 1, 4), dtype=tf.float32,
name='dense_3016_input'), name='dense_3016_input', description="created by layer
'dense_3016_input'"), but it was called on an input with incompatible shape (None,
4).
transactions  | new transaction status: accepted
transactions  | new transaction arrived: {'transaction_id': 'd753a505-04af-409a-
a42a-c443c3add6d1', 'value_in_cents': 132112523, 'description': 'some purchase',
'customer_id': '2b13b74f-f94c-433c-b8ef-1c7a0f99fb59', 'merchant_id': '7b6eb2ca-
729a-4e07-aa4b-107936efc3eb', 'transaction_timestamp': 1281639157,
'event_timestamp': 1624739067, 'latitude': 26.97125816345215, 'longitude':
29.92924690246582}
transactions  | failed to contact antifraud service: <_InactiveRpcError of RPC
that terminated with:
transactions  |         status = StatusCode.UNAVAILABLE
transactions  |         details = "upstream connect error or disconnect/reset
before headers. reset reason: local reset"
transactions  |         debug_error_string = "
{"created":"@1624739067.469222200","description":"Error received from peer
ipv4:172.28.0.9:5005","file":"src/core/lib/surface/call.cc","file_line":1066,"grpc
_message":"upstream connect error or disconnect/reset before headers. reset
reason: local reset","grpc_status":14}"
transactions  | >
transactions  | new transaction status: rejected
```

Com a impossibilidade de entrar em contato com o antifraud service, o status da transaçao é *rejected*

Agora se a gente desligar `antifraud-1` e `antifraud-2`:

```
docker kill antifraud-2
```

Agora todas as transaçoes estao registradas mas todas com o status *rejected*.

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`?

vantagens

- Utilizar o envoy como proxy para alternar entre os serviços `antifraud-1` e `antifraud-2` permite idealmente (ou seja, se tiver os recursos necessarios), como expliquei na minha ultima resposta, de quase dividir por 2 o tempo de processamento pelo modelo de ML anti fraud do fluxo das transaçoes. Pois ele paraleliza o fluxo entre 2 serviços de prediçao em vez de um so.
- Caso um desses 2 serviços falhar, entao ainda tem a metade das transaçoes que vao ser avaliadas pelo modelo anti fraude. Isso deixa a arquitetura mais resiliente.

desvantagens

- Caso um desses serviços antifraud falhar, entao a metade das transaçoes vao ser identificadas como fraudes. O ideal seria o proxy conseguir identificar qual dos 2 serviços esta fora do ar para redirecionar todas as transaçoes para o serviço funcionando.
- Apesar de paralelizar o fluxo das transaçoes entre 2 serviços de prediçao, o que ajuda a otimizar o tempo global de processamento, o load balancer envoy adiciona uma latencia extra. Entao se tiver um dos 2 serviços anti fraude fora do ar ou com problema, o tempo total gasto em avaliar o status da transaçao sera maior: uma latencia extra para cada uma das transaçoes.

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.

```
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)
```

```
def notify_status(transaction_id, status):
    acc = 0
    while True:
        acc += 1
        try:
            resp = requests.patch(
                f"{BFF_HOST}/api/v1/transactions/{transaction_id}/status",
                json={"status": status},
            )
        except Exception as err:
            if acc >= 10:
                raise BFFStatusWebhookError(err)
            else:
                logging.error(f"Not possible to communicate with bff: {str(err)}")
                time.sleep(1)
```

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

- Esse webhook adiciona uma latencia extra
- Se o serviço `bff` nao for disponivel na hora da execuçao do webhook, entao o serviço `bff` perde a notificaçao e nao consegue atualizar a transaçao no cache do `Redis`.

9. Cite algumas diferenças entre Protocol Buffers e Avro.

3 aspectos que diferenciam o Avro do Protocol Buffers:

- *Dynamic typing*: Avro não requer que o código seja gerado. Os dados são sempre acompanhados por um esquema que permite o processamento completo desses dados sem geração de código, tipos de dados estáticos, etc. Isso facilita a construção de sistemas e linguagens genéricas de processamento de dados.
- *Untagged data*: Como o esquema está presente quando os dados são lidos, consideravelmente menos informações de tipo precisam ser codificadas com os dados, resultando em um tamanho de serialização menor.
- Sem IDs de campo atribuídos manualmente: quando um esquema muda, tanto o esquema antigo quanto o novo estão sempre presentes ao processar dados, portanto, as diferenças podem ser resolvidas simbolicamente, usando nomes de campo.

Apesar do tamanho de dados codificados ligeiramente menor para Avro, a capacidade de atualizar as definições de mensagem Protobuf de uma maneira compatível sem ter que prefixar os dados codificados com um identificador de esquema torna-o uma escolha melhor para outros casos (como a transmissão de dados de dispositivos IoT).