

Streaming de Dados em Tempo Real: Aula 2

Prof. Felipe Timbó



Ementa (dia 2)

- Data Ingestion com Apache Kafka
- Kafka Connect
- Kafka Web Project

Apache Kafka

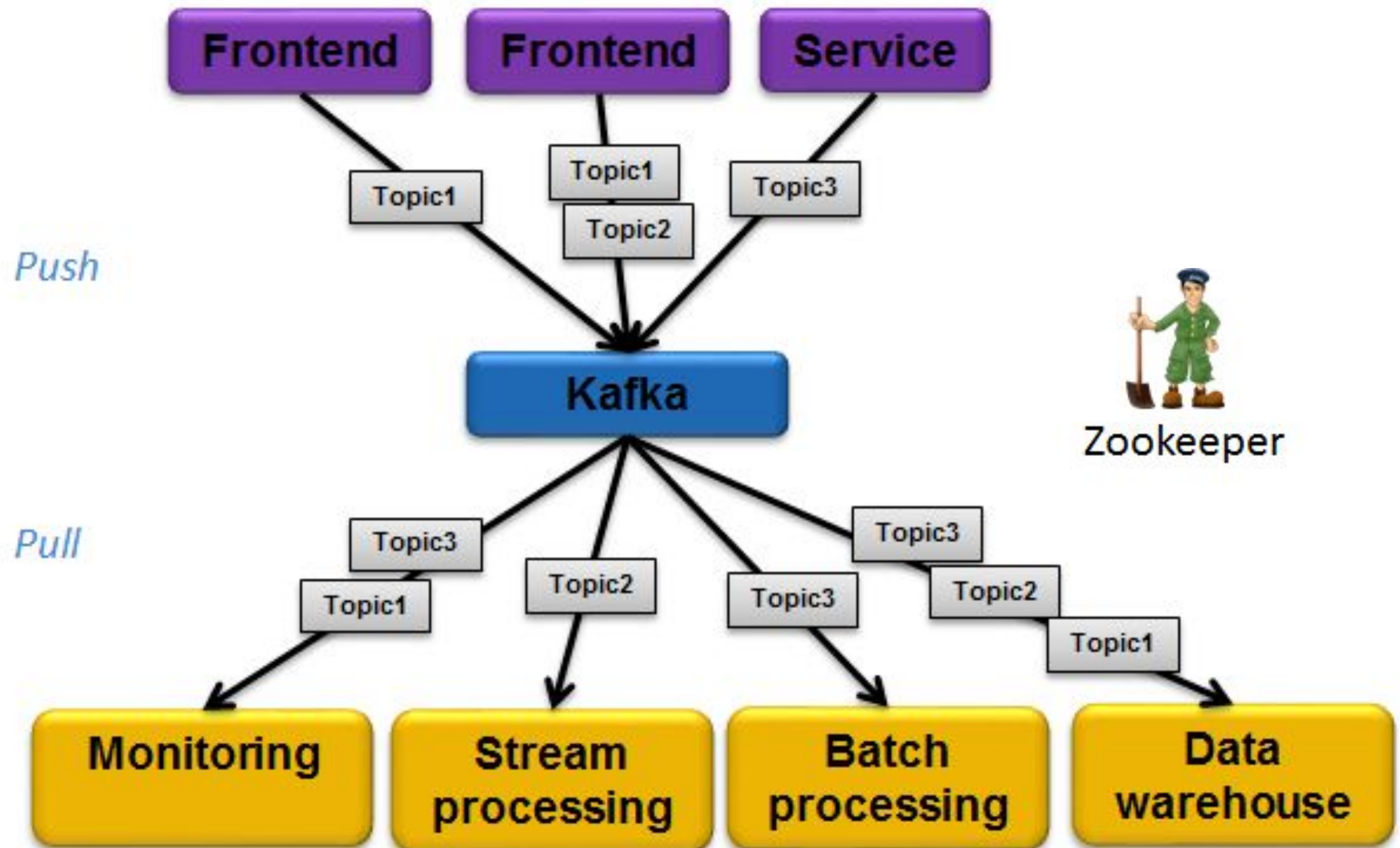
(cont.)

Kafka: relembrando

Producers

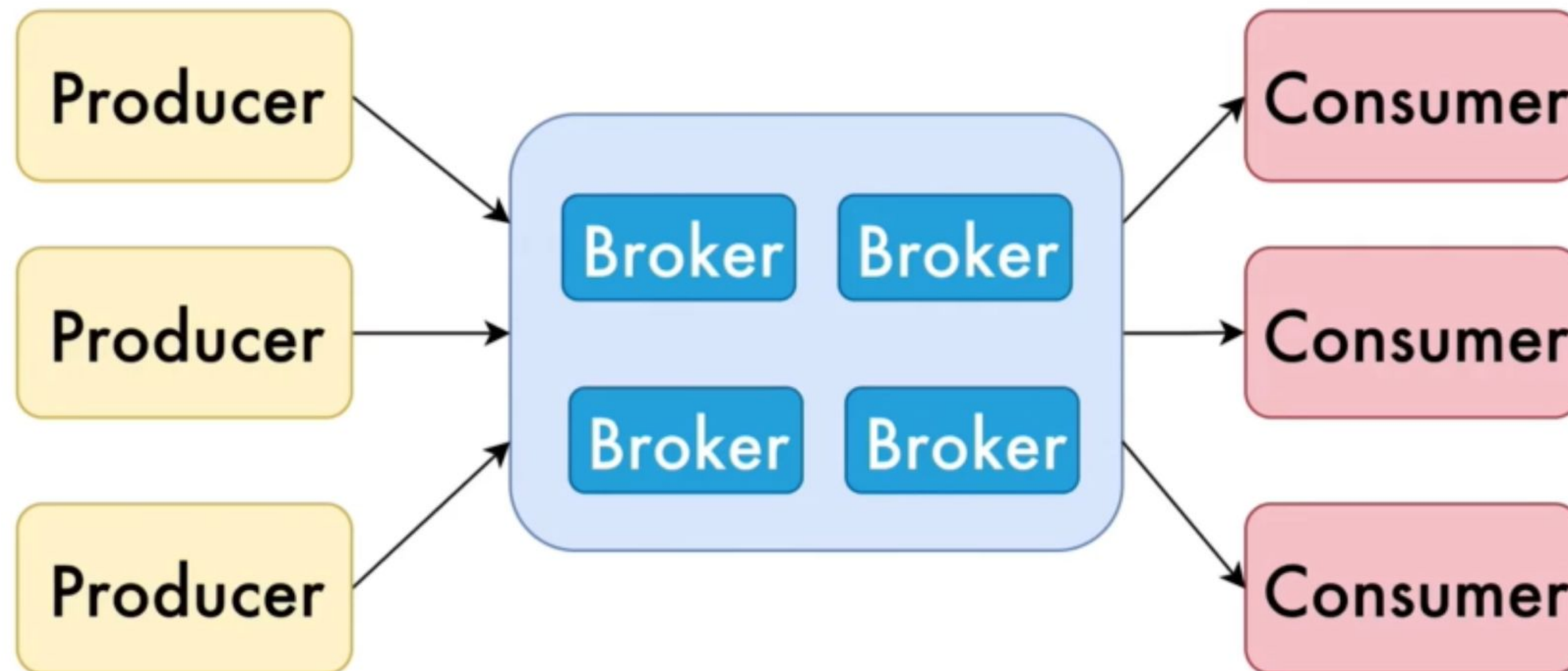
Broker

Consumers



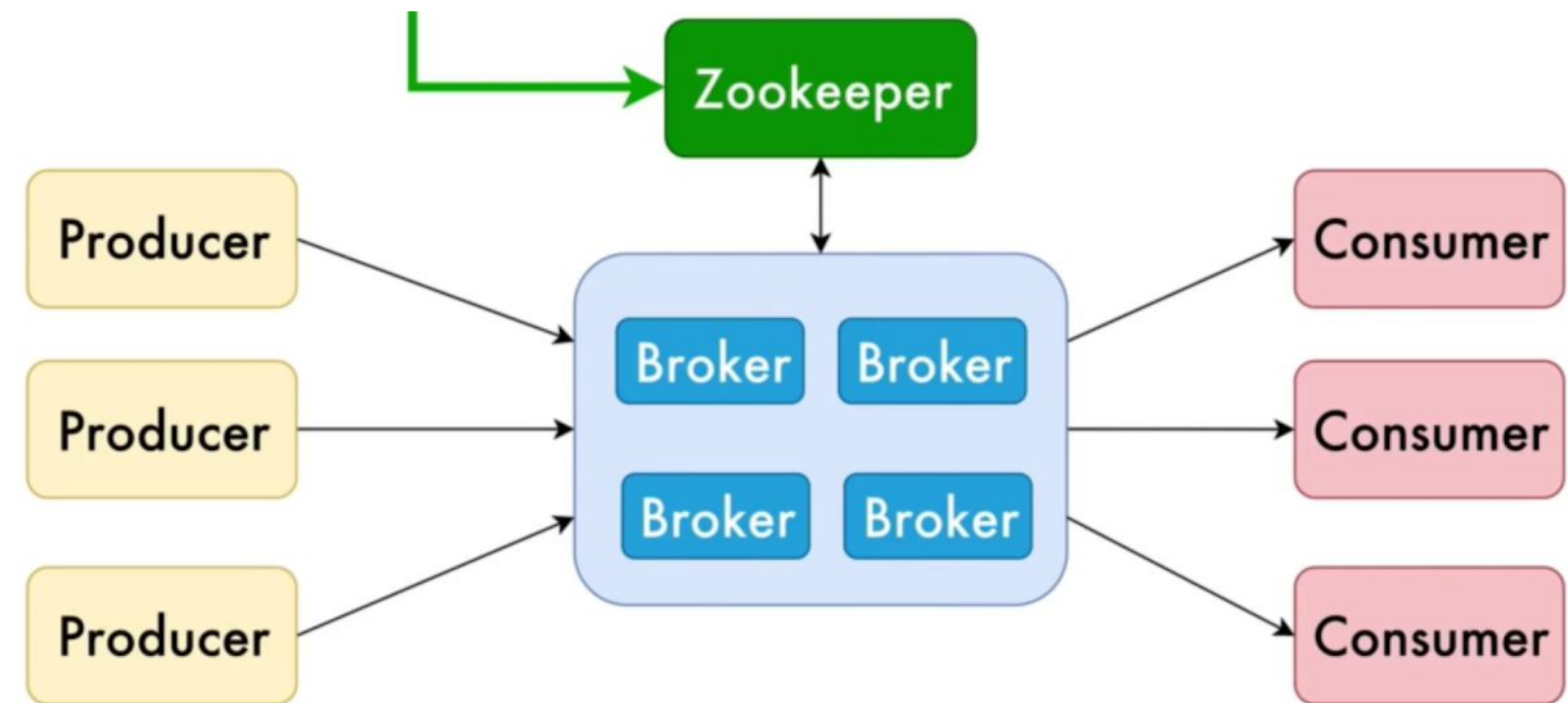
Kafka na vida real

- Em aplicações do mundo real: LinkedIn, Netflix:
 - Aplicações distribuídas, isto é, mais de um Broker



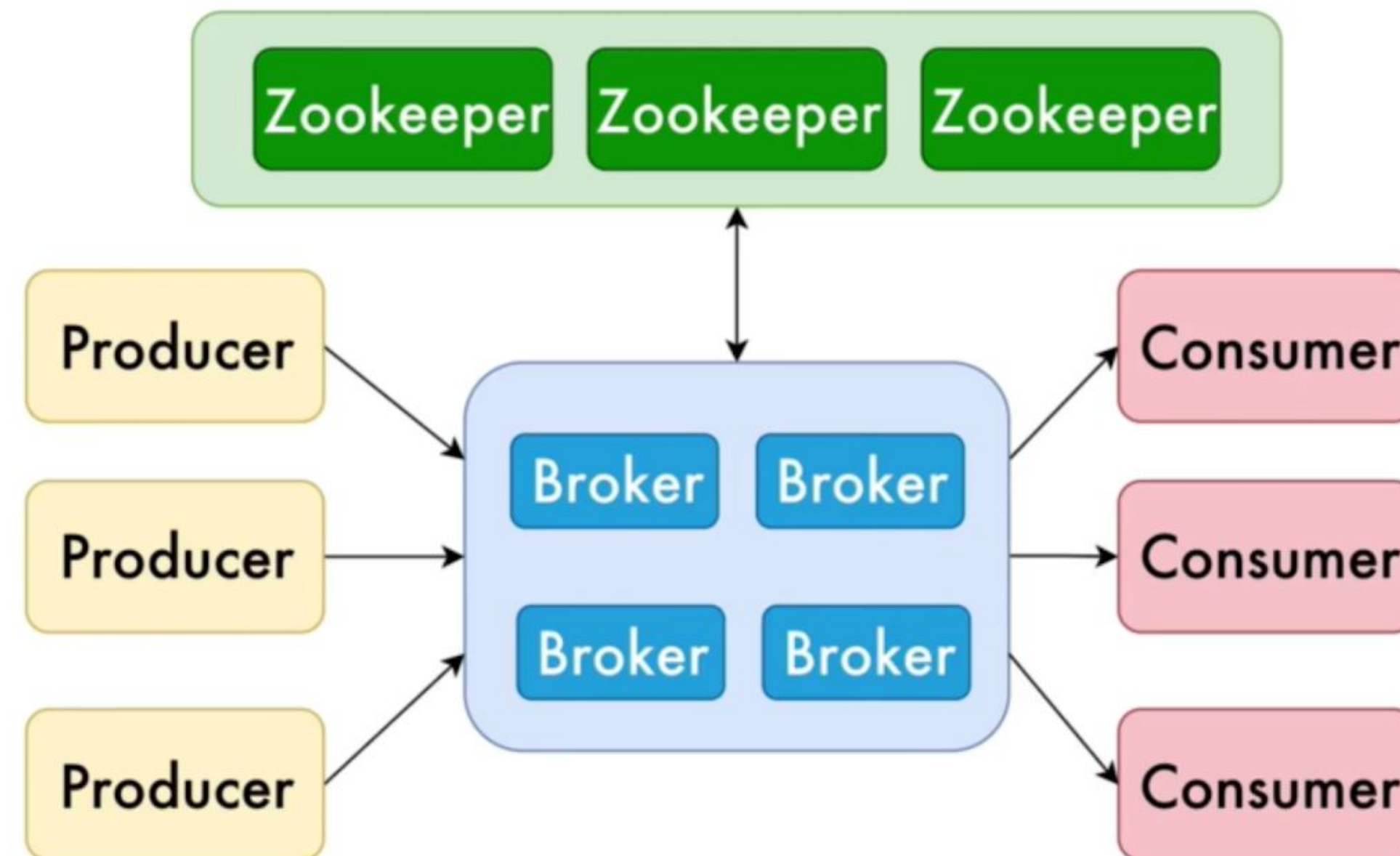
Zookeeper

- Utilizado não só pelo Apache Kafka, mas pelo Apache Hadoop também, por exemplo.
- Mantém uma lista de brokers ativos
- Elege controller
- Gerencia as configurações de tópicos e partições.



Zookeeper na vida real

- Múltiplos zookeepers rodando, para garantir a tolerância a falhas e os benefícios dos sistemas distribuídos.



Inicializando o Apache Kafka

Abrir o terminal

Acessar o diretório kafka:

➤ `cd kafka`

Inicializar o zookeeper:

➤ `bin/zookeeper-server-start.sh config/zookeeper.properties`

Inicializar o servidor kafka (em outra aba):

➤ `bin/kafka-server-start.sh config/server.properties`

Voilà

Zookeeper **localhost:2181**

Kafka server (broker) **localhost:9092**

Utilizando o KAFKA via CLI
(Command-Line Interface)

Criando um tópico

Em outro terminal: **Terminal 1**

➤ `bin/kafka-topics.sh --create --bootstrap-server
localhost:9092 --topic cidades`

Listar os tópicos existentes:

➤ `bin/kafka-topics.sh --list --zookeeper
localhost:2181`

Produzindo mensagens

Acessando o kafka broker para enviar mensagens

```
➤ bin/kafka-console-producer.sh --broker-list  
  localhost:9092 --topic cidades  
    > Fortaleza  
    > Sobral  
    > Canindé  
    > Russas  
    > Quixadá
```

Consumindo mensagens

Em outro terminal: **Terminal 2**

➤ `bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic cidades`

De volta ao **Terminal 1** digitar:

- > New York
- > Dubai
- > Rio de Janeiro
- > Buenos Aires

Consumindo mensagens do início

Terminal 2

Parar o terminal 2:

➤ `ctrl+c`

Executar novamente para obter mensagens desde o início:

➤ `bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic cidades --from-beginning`

Fatos importantes

Kafka armazena mensagens mesmo se elas já tenham sido consumidas por um de seus “consumers”

Algumas mensagens podem ser lidas múltiplas vezes por diferentes “consumers” em diferentes momentos

parâmetro no server.properties:

log.retention.hours=168 # (7 dias)

Criando um novo consumidor

Em outro terminal: **Terminal 3**

➤ `bin/kafka-console-consumer.sh --bootstrap-server
localhost:9092 --topic cidades`

**Múltiplos consumidores e múltiplos produtores podem
trocar mensagens via clusters kafka.**

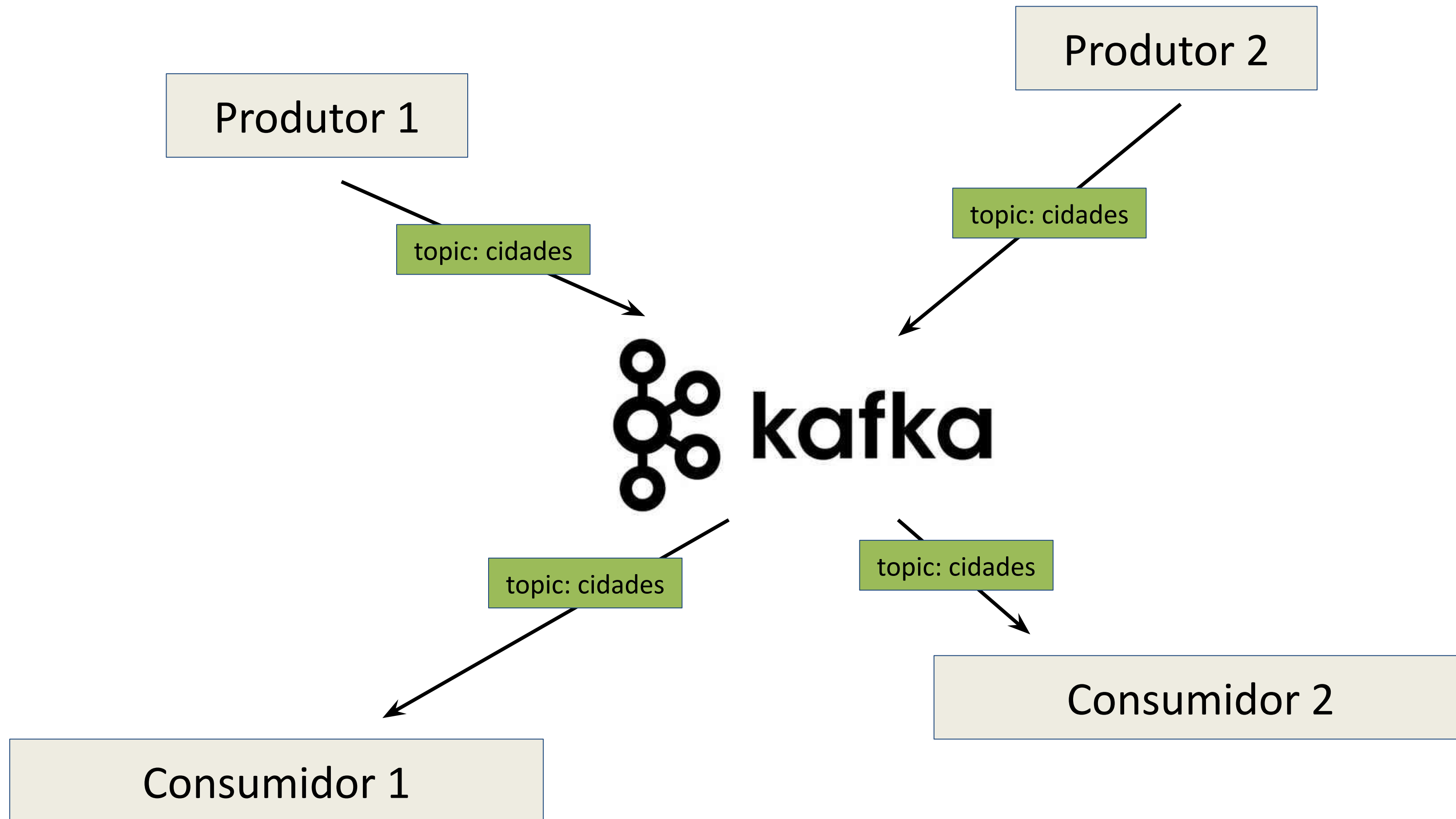
Criando um novo produtor

Em outro terminal: **Terminal 4**

```
➤ bin/kafka-console-producer.sh --broker-list  
  localhost:9092 --topic cidades  
  > Barcelona
```

Produtores não sabem nada sobre outros produtores.

Esquema até então...



Ao encerrar o produtor 2
(Terminal 4) o que acontece
aos consumidores?

Ao deletar o produtor 2

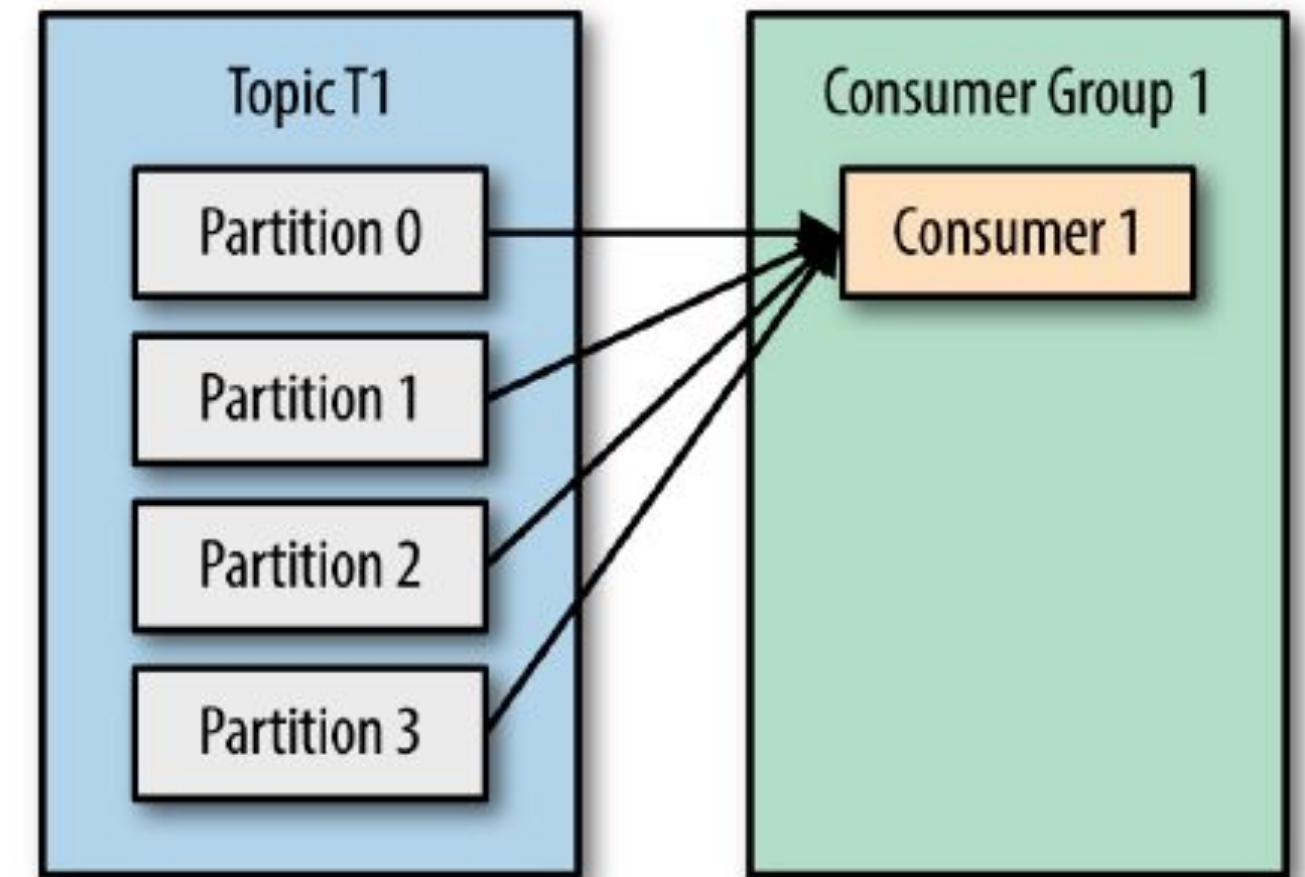
Resposta: Apenas deixam de receber mensagens do produtor excluído, mas continuam operando normalmente.

Consumidores recebem a mensagem do cluster kafka, independente dos produtores.

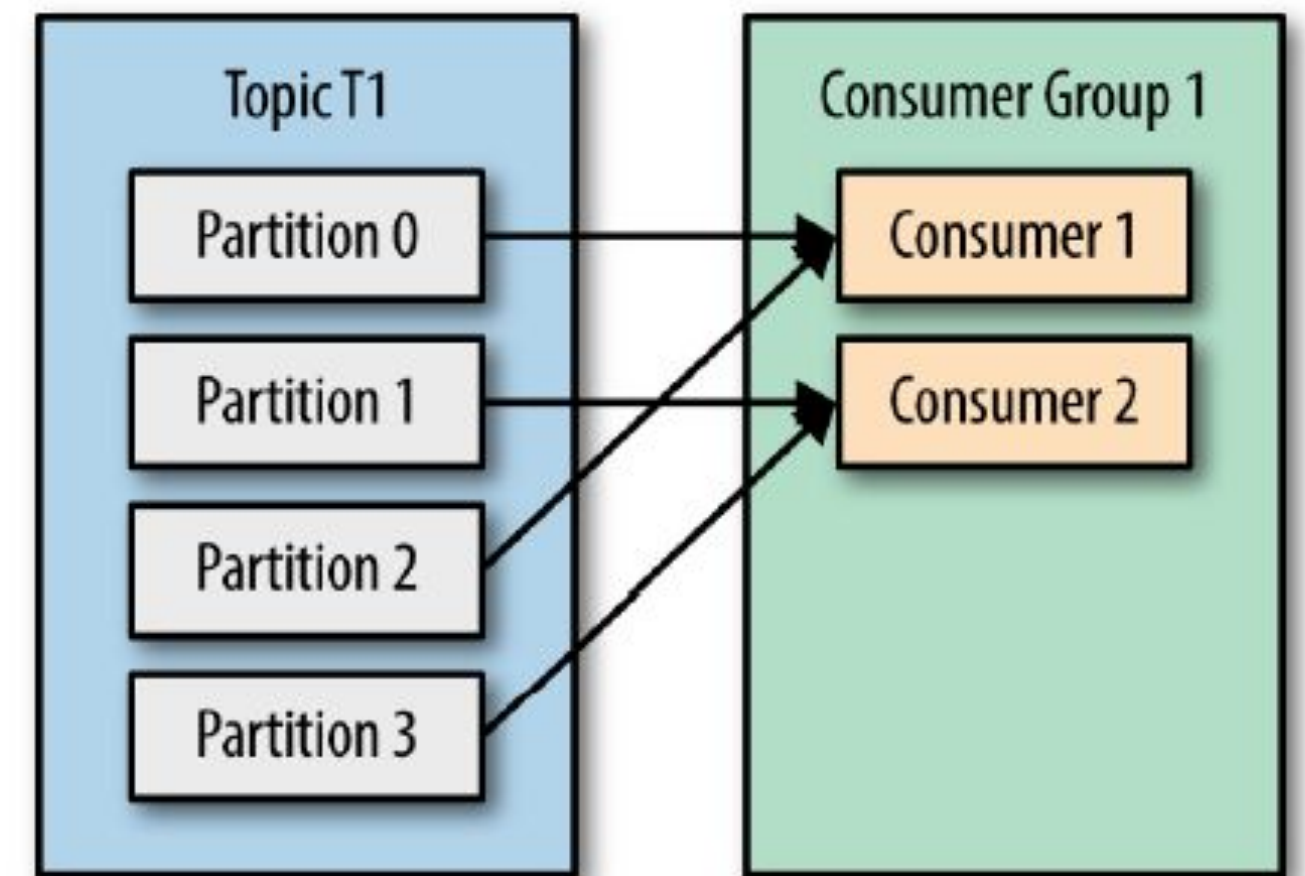
Agora, você deve deletar todos os produtores/consumidores

Offset e Partition

- Internamente, o Kafka quebra os tópicos em partições.
- O número de partições é indicado quando o tópico é criado.
- Não há limite de partições.
- Para facilitar a alta disponibilidade, as partições de um tópico são espalhadas entre os brokers do cluster.



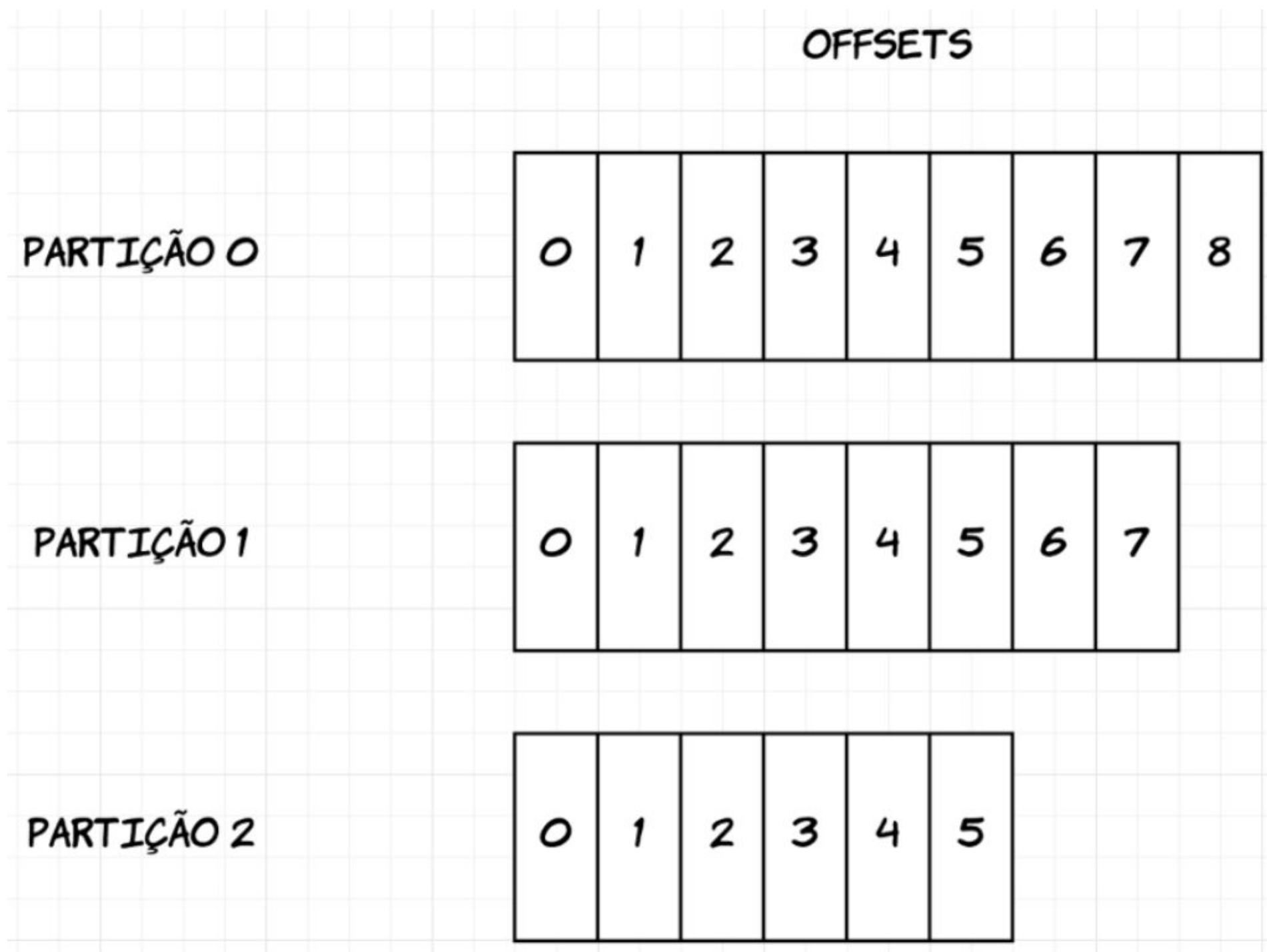
OU



...

Offset e Partition

- Exemplo:



- O tópico possui 23 mensagens separadas em 3 partições.
- Dentro de cada partição, as mensagens são ordenadas por um metadado chamado **offset**.
- O offset serve para ordenar apenas no contexto da partição.
- Offsets iguais em partições diferentes resultam em mensagens diferentes.

Por que Partition?

	consumer1	consumer2	consumer3	...
	OFFSETS			
PARTIÇÃO 0	0	1	2	3 4 5 6 7 8
PARTIÇÃO 1	0	1	2	3 4 5 6 7
PARTIÇÃO 2	0	1	2	3 4 5

- Proporciona paralelismo tanto para consumir quanto para produzir mensagens.
- Alguns consumidores trabalham com faixas de offset para não serem sobrecarregados.

Consumindo mensagens com offset

Terminal 2

Parar o terminal 2:

➤ `ctrl+c`

Executar novamente para obter mensagens com offset:

➤ `bin/kafka-console-consumer.sh --bootstrap-server
localhost:9092 --topic cidades --partition 0
--offset 4`

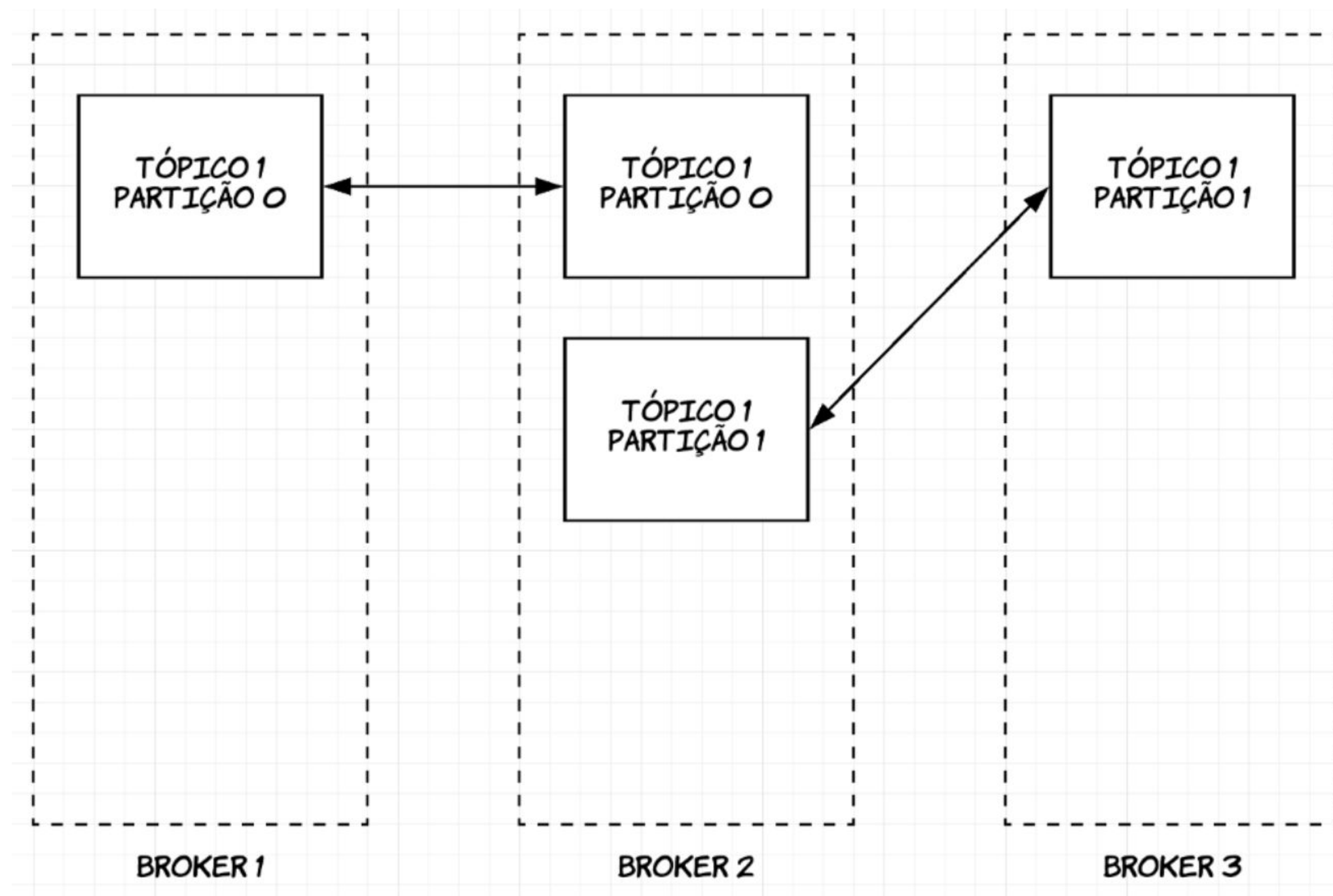
(Deixa de fora os 4 primeiros registros da partição 0)

Fator de replicação

- O fator de replicação também é configurado quando o tópico é criado.
- O fator de replicação gera cópias das partições em outros brokers, assim se algum broker ficar indisponível, o Kafka continua sendo capaz de servir as mensagens sem interrupção no serviço.

Fator de replicação

- Exemplo:



cluster com 3 brokers e 1 tópico, sendo o tópico
com 2 partições e fator de replicação 2

- Caso o broker 2 fique indisponível o cluster não é afetado.
- Para descobrir o número de brokers que podem ser interrompidos sem afetar o cluster basta realizar o cálculo:
 - fator de replicação - 1.
- Em nosso exemplo, o resultado seria ($2 - 1 = 1$ broker).

Criando um tópico com partições e rf

Em outro terminal: **Terminal 3**

➤ `bin/kafka-topics.sh --bootstrap-server localhost:9092
--create --replication-factor 1 --partitions 3 --topic
test`

Criar um produtor test: **Terminal 1**

➤ `bin/kafka-console-producer.sh --broker-list
localhost:9092 --topic test`

Criar um consumidor test: **Terminal 2**

➤ `bin/kafka-console-consumer.sh --bootstrap-server
localhost:9092 --topic test --partition 0
--from-beginning`

Testar com offset

Comandos básicos do KAFKA - Resumo

Start Broker

```
bin/kafka-server-start.sh config/server.properties
```

Start Zookeeper

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```

```
bin/kafka-topics.sh \  
--bootstrap-server localhost:9092 \  
--create \  
--replication-factor 1 \  
--partitions 3 \  
--topic test
```

Create new topic

```
bin/kafka-topics.sh \  
--bootstrap-server localhost:9092 \  
--list
```

List all topics

```
bin/kafka-topics.sh \  
--bootstrap-server localhost:9092 \  
--describe \  
--topic test
```

Details about the topic

```
bin/kafka-console-producer.sh \  
--broker-list localhost:9092 \  
--topic test
```

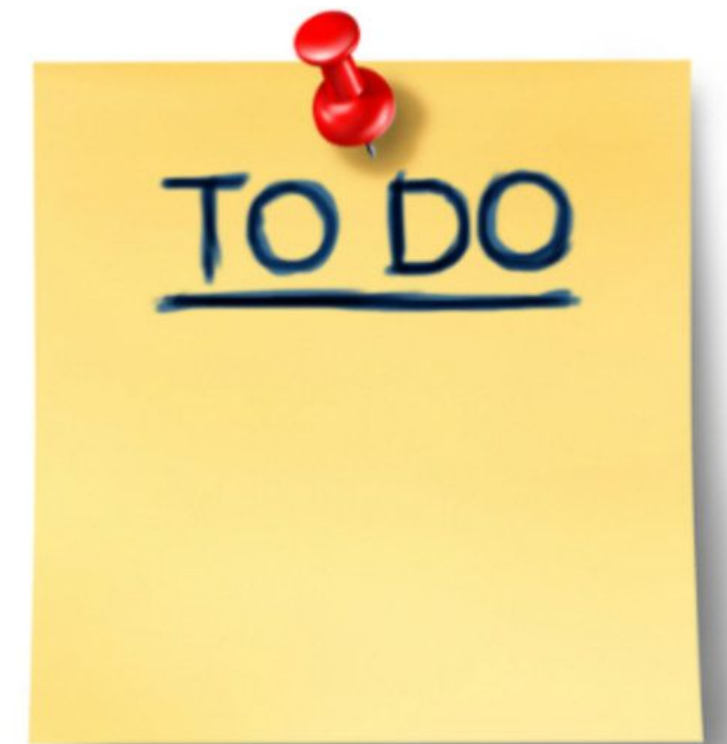
Start console producer

```
bin/kafka-console-consumer.sh \  
--bootstrap-server localhost:9092 \  
--topic test \  
--from-beginning
```

Start console consumer

KAFKA - Prática

Atividade 1



1. Crie um tópico com o nome de vocês;
2. Liste os tópicos e verifique se o seu foi criado;
3. Gere dados para o tópico criado.

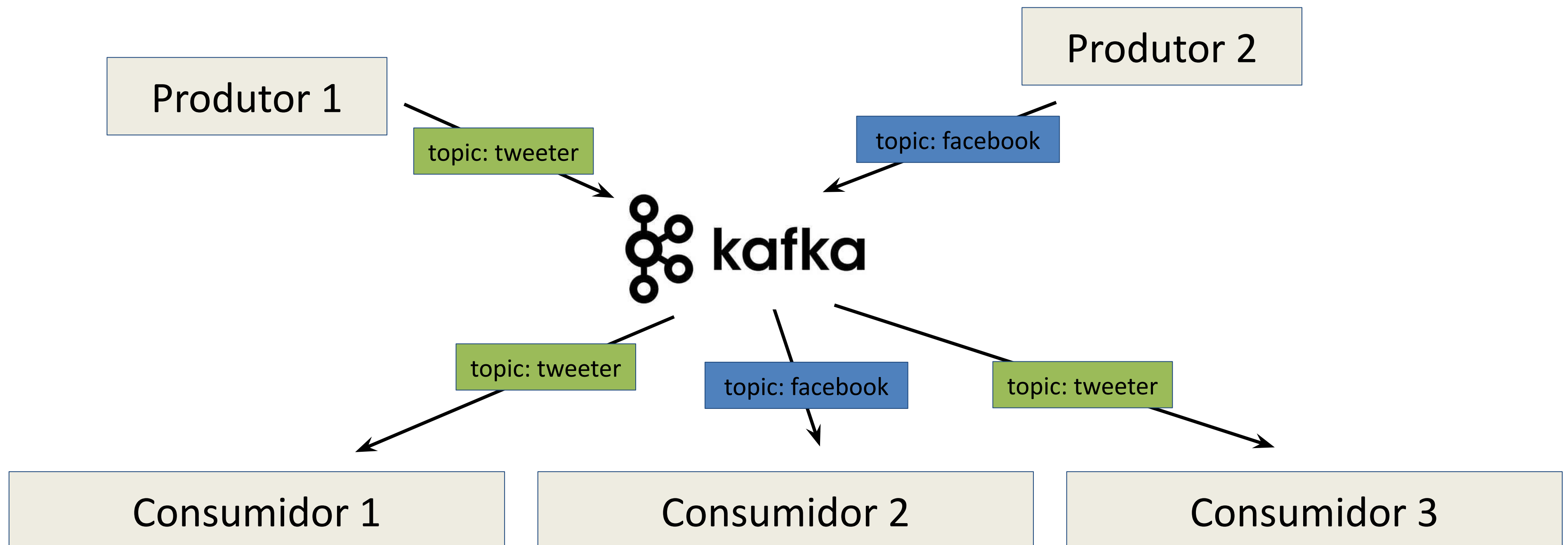
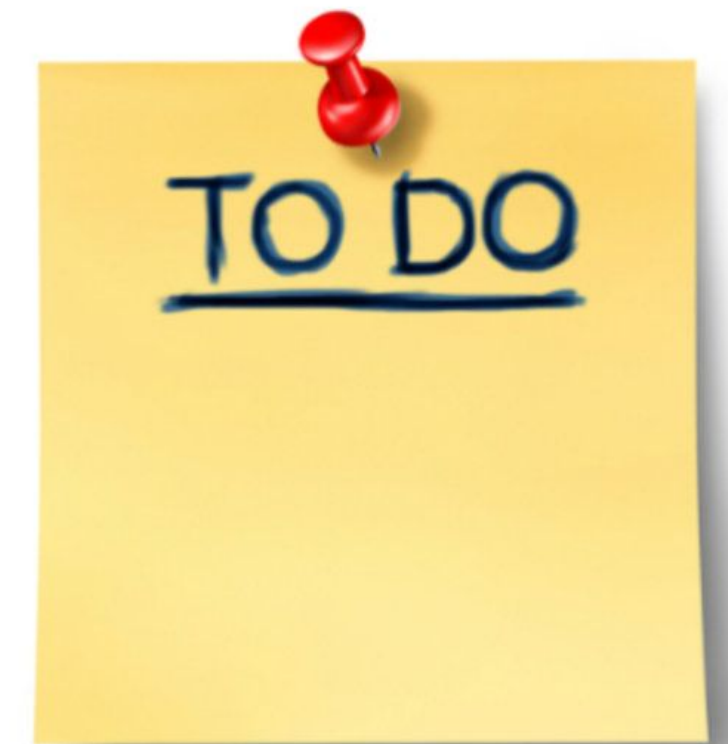
Obs.:

Endereço do zookeeper: `localhost:2181`

broker-list: `localhost:9092`

Atividade 2

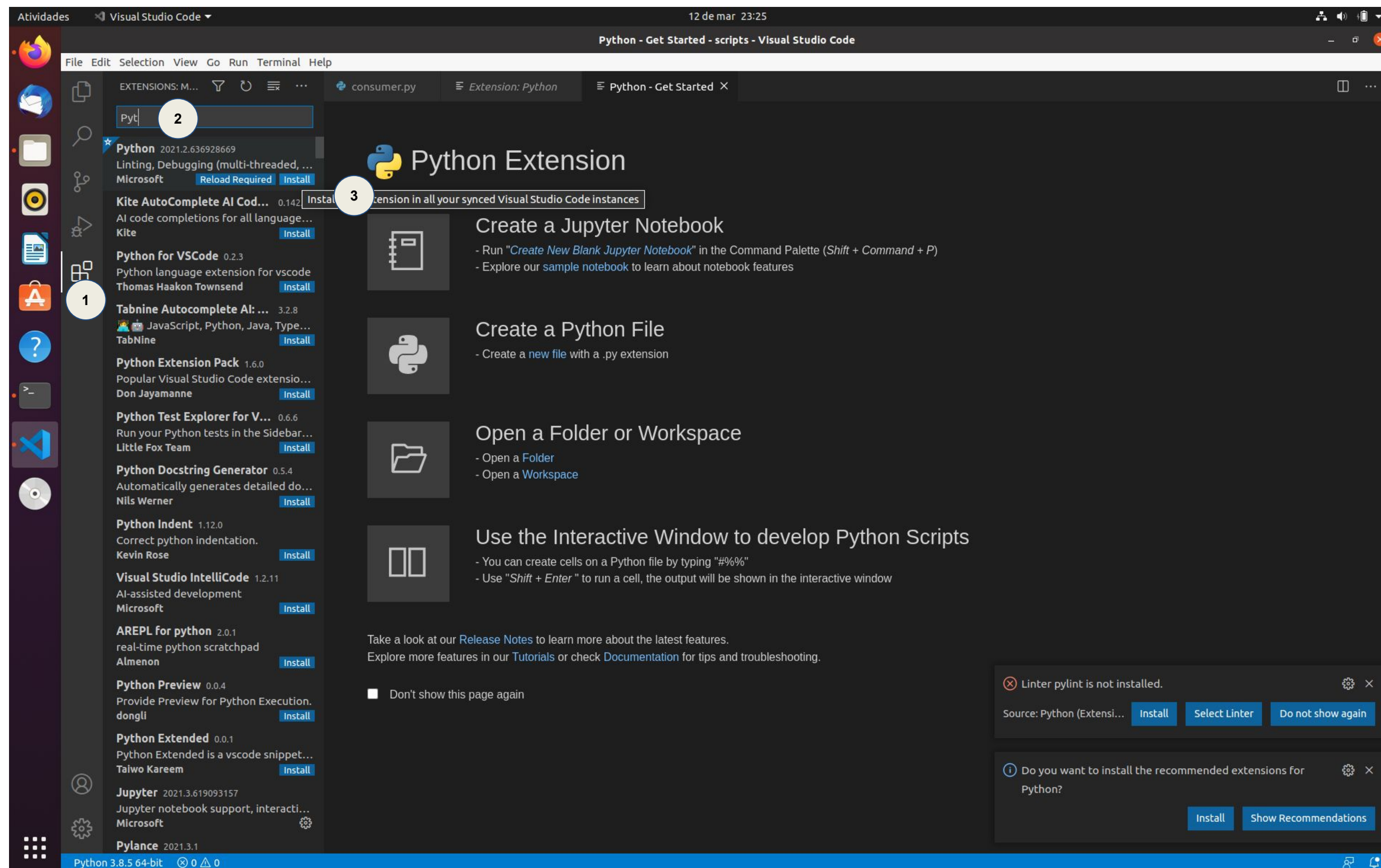
Crie o seguinte esquema de troca de mensagens via streaming de dados no Apache KAKFA:



KAFKA - Prática 2

Produtor/consumidor com Python

Instalar dependência do Python no VSCode



Produtor/consumidor com Python

Instalar a interface do Python com o KAKFA no Terminal

- `pip3 install kafka-python`

Criar uma pasta scripts no diretório ~ e abrir no VS Code

- `mkdir ~/scripts`

- `code ~/scripts`

criar os seguintes scripts:

- `consumer.py`

- `producer.py`

Criar e executar os scripts juntamente com o Professor.

Produtor/consumidor com Python

consumer.py

```
from kafka import KafkaConsumer

consumer = KafkaConsumer(
    'names',
    bootstrap_servers=['localhost:9092']
)

for message in consumer:
    print(message)
```

Produtor/consumidor com Python

producer.py

```
import time
from kafka import KafkaProducer

producer = KafkaProducer(bootstrap_servers=['localhost:9092'])
producer.send('names', 'Felipe'.encode('utf-8'))

time.sleep(20)
```


KAFKA - Prática 3

Produtor/consumidor

Instalar dependência do fake names no python

➤ `pip3 install Faker`

Baixar códigos python produtor/consumidor

➤ `cd ~/scripts`

➤ `wget www.lia.ufc.br/~timbo/streaming/producer.py`

➤ `wget www.lia.ufc.br/~timbo/streaming/consumer.py`

Abrir scripts no VS Code

Atividade 3



1. Altere o script para serem enviados dados de 2 em 2 segundos;
2. Crie um segundo consumidor, e altere o produtor para enviar os dados para ambos os consumidores;
3. Imprima nos consumidores apenas o nome das pessoas Fake que estão sendo produzidas, ao invés do conteúdo completo da mensagem;
4. Imprima no consumidor 1 o nome, e no consumidor 2 o sobrenome da pessoa Fake que é produzido.

KAFKA - Prática 4

Produtor/consumidor

Baixar código único python produtor/consumidor

➤ `wget www.lia.ufc.br/~timbo/streaming/producer_consumer.py`

Rodar script no VS Code

➤ `python3 producer_consumer.py`

Atividade 4



TO DO

1. Gere dados de quatro producers simultâneos
2. Aumente a frequência de geração das tuplas (geração mais rápida);
3. Filtre e imprima apenas por tuplas que possuem valores de peso maiores que 80;
4. Filtre e imprima apenas por tuplas que possuem valores de IMC acima de 35 ($IMC = peso/altura^2$).

Kafka + MongoDB

Produtor/consumidor com Python + MongoDB

Instalar o mongodb

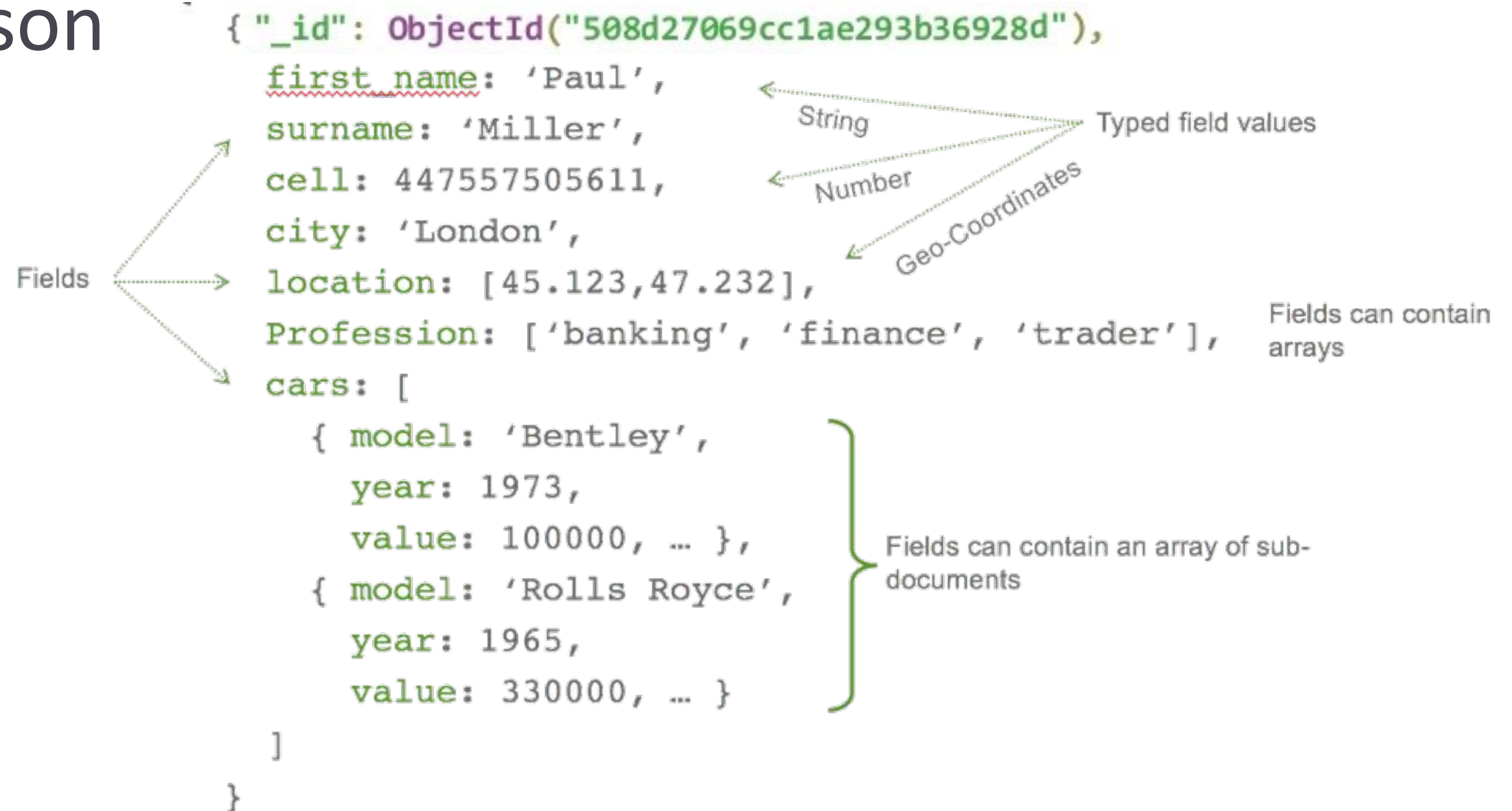
➤ `sudo apt install mongodb -y`

Instalar dependência do mongodb no python

➤ `pip3 install pymongo`

MongoDB - Introdução

- Banco de dados NoSQL Orientado a Documentos
- Semelhante a json

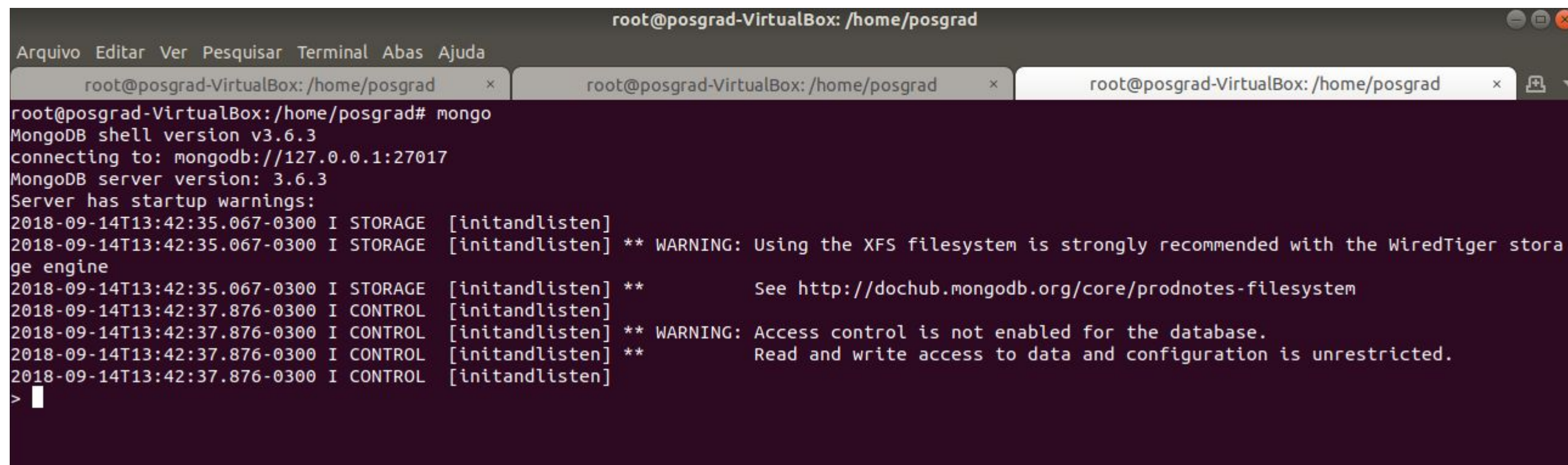


MongoDB - Terminologia

- Bancos de dados
- Coleções
- Documentos
- Colunas

MongoDB na Prática

No terminal, abrir nova aba e iniciar o cliente mongo



```
root@posgrad-VirtualBox: /home/posgrad
Arquivo Editar Ver Pesquisar Terminal Abas Ajuda
root@posgrad-VirtualBox: /home/posgrad x root@posgrad-VirtualBox: /home/posgrad x root@posgrad-VirtualBox: /home/posgrad x
root@posgrad-VirtualBox:/home/posgrad# mongo
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.3
Server has startup warnings:
2018-09-14T13:42:35.067-0300 I STORAGE [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine
2018-09-14T13:42:35.067-0300 I STORAGE [initandlisten] ** See http://dochub.mongodb.org/core/prodnotes-filesystem
2018-09-14T13:42:37.876-0300 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-09-14T13:42:37.876-0300 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
2018-09-14T13:42:37.876-0300 I CONTROL [initandlisten]
>
```

➤ mongo

➤ use aula1

Operações do MongoDB

Criar um documento e armazená-lo em uma coleção chamada "colors"

➤ `db.colors.save({name:"red",value:"FF0000"});`

Verificar se o documento foi armazenado no banco de dados

➤ `db.colors.find();`

Operações do MongoDB

Criar uma coleção de documentos de caracteres

```
➤ var chars = "abcdefghijklmnopqrstuvwxyz"
➤ for(var i =0; i<chars.length; i++) {
...  var char = chars.substr(i, 1);
...  var doc = {char:char, code: char.charCodeAt(0)};
...  db.alfabeto.save(doc);
... }
```

Este loop criará 26 documentos, um para cada letra minúscula do alfabeto, cada documento contendo o caractere em si e seu código de caractere ASCII.

Operações do MongoDB

Inserir dados

```
➤ db.usuarios.insert({  
  nome: "Higor",  
  cidade: "Porto Alegre",  
  estado: "Rio Grande do Sul"  
})
```

```
CREATE TABLE USUARIOS (  
  
  id  
  MEDIUMINT NOT NULL AUTOINCREMENTS,  
  nome  
  Varchar(30),  
  cidade  
  Varchar(60),  
  estado  
  Varchar(60),  
  PRIMARY  
  KEY (id))
```


Operações do MongoDB

Atualizar dados

➤ `db.usuarios.update({ cidade: "Rio de Janeiro"}, { estado: "Rio de Janeiro" })`

```
UPDATE usuarios SET estado = "Rio de Janeiro" WHERE cidade = "Rio de Janeiro"
```

➤ `db.usuarios.update({ nome: "Higor"}, {
 nome: "Higor",
 cidade: "Porto Alegre",
 estado: "Rio Grande do Sul",
 idade: 26
})`

Operações do MongoDB

Buscar dados

➤ `db.usuarios.find({estado: {$eq: "Rio de Janeiro"}})`

```
SELECT * FROM usuarios WHERE estado = "Rio de Janeiro"
```

➤ `db.usuarios.find({estado: {$eq: "Rio Grande do Sul"}})`

➤ `db.usuarios.find({ idade: { $gt: 25, $lte: 50 } })`

Operações do MongoDB

Deletar dados

➤ `db.usuarios.remove({estado: "Rio Grande do Sul" })`

```
DELETE FROM usuarios WHERE estado = "Rio Grande do Sul"
```

➤ `db.usuarios.remove({})`

```
DELETE FROM usuarios
```

Operações do MongoDB

Buscar dados

- **\$eq:** exatamente igual (=)
- **\$ne:** diferente (<> ou !=)
- **\$gt:** maior do que (>)
- **\$lt:** menor do que (<)
- **\$lte:** menor ou igual a (<=)
- **\$in:** o valor está contido em um array de possibilidades, como em um OU. Ex: {idade: {\$in: [10,12] }}

Operações do MongoDB

Buscar dados

➤ `db.usuarios.find({ nome: /^Higor/ })`

```
SELECT * FROM usuarios WHERE nome like "Higor%"
```

`db.usuarios.find({cidade : "Porto Alegre"}).sort({nome:1})`

```
SELECT * FROM usuarios WHERE cidade = "Porto Alegre" ORDER BY nome ASC
```

KAFKA - Prática 5

Produtor/consumidor

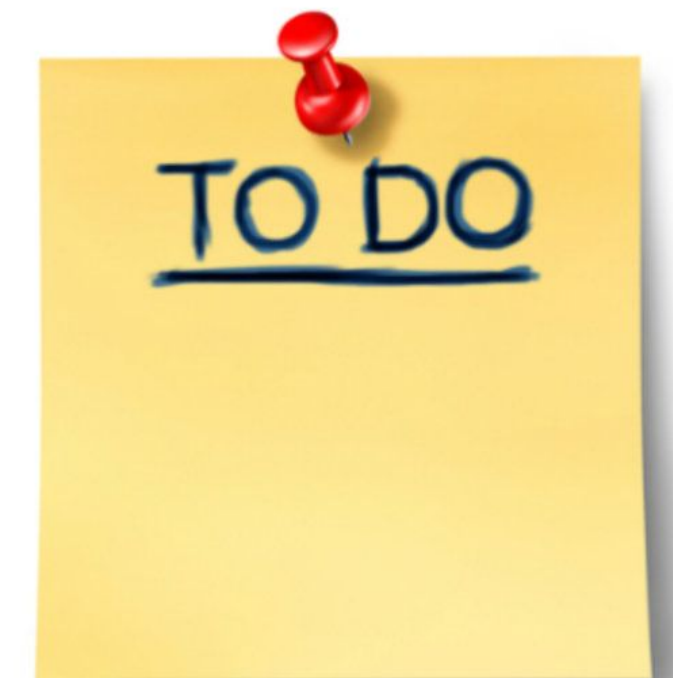
Baixar código único python produtor/consumidor

➤ `wget www.lia.ufc.br/~timbo/streaming/producer_consumer_mongodb.py`

Rodar script no VS Code

➤ `python3 producer_consumer_mongodb.py`

Atividade 5



1. Inclua na geração de tuplas o nome das pessoas como primeiro atributo. A estrutura do seu arquivo json gerado via streaming será: nome, idade, altura e peso. O nome pode ser gerado pela biblioteca Faker.
2. Salve no banco de dados MongoDB (via consumidor), apenas as pessoas que começam com a letra J.

Atividade 5

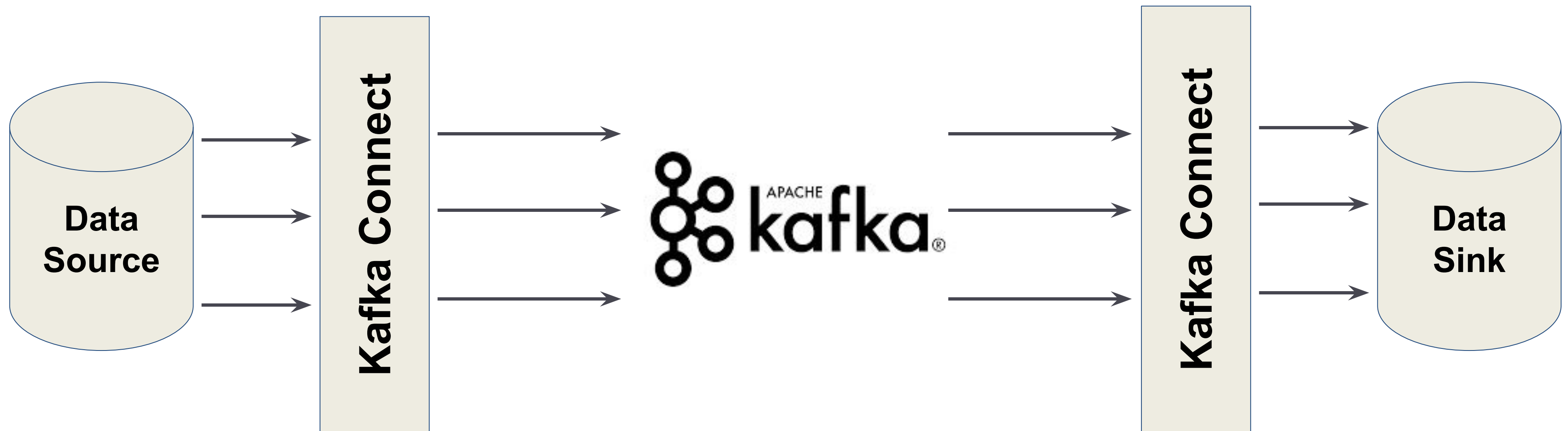


3. Pesquise como seria para salvar os dados em um banco de dados MySQL ou algum outro banco de dados relacional de sua preferência. Salve nesse banco (via consumidor) apenas as pessoas com idade maior que 40 anos.

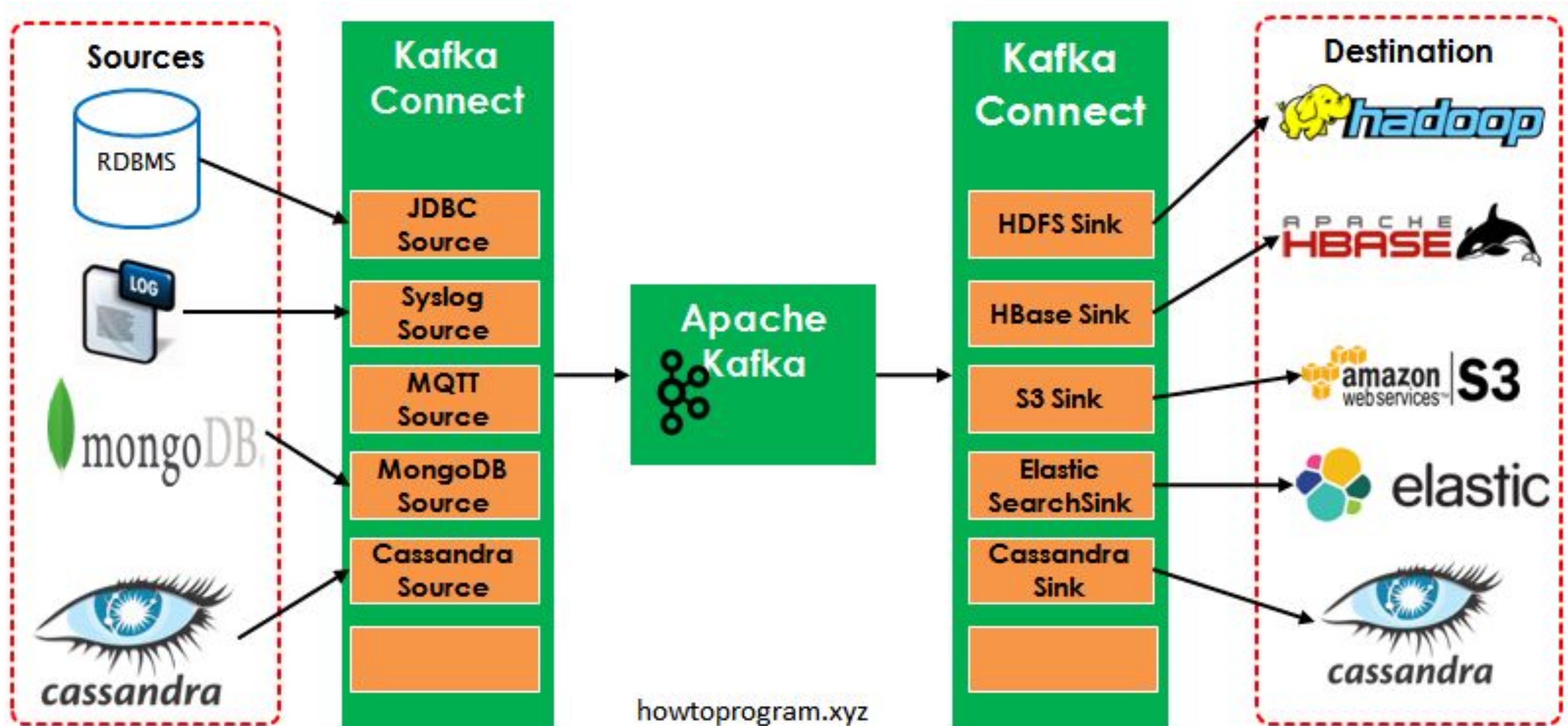
Kafka Connect

Kafka Connect

- Fornece uma maneira escalonável e confiável de mover dados entre o Kafka e outros provedores de dados.



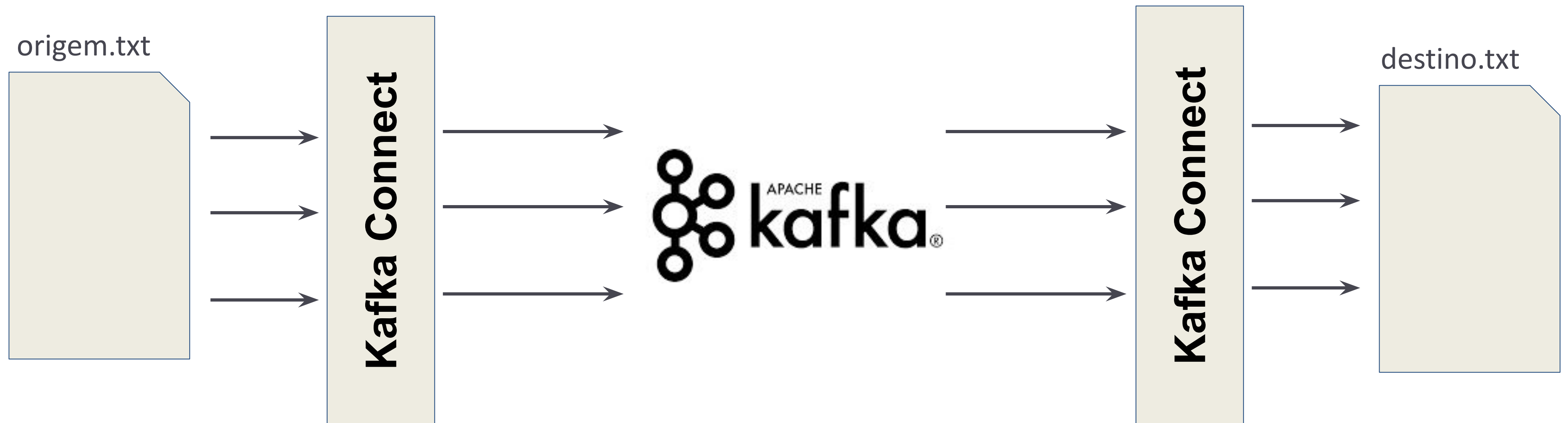
Kafka Connect



Prática

Kafka Connect

- Vamos utilizar Kafka Connect para fazer o Streaming de dados de um arquivo **origem.txt** para um **destino.txt**



Kafka Connect

1. Abrir no VS Code ou no gedit o arquivo

➤ `config\connect-file-source.properties`

2. As linhas do arquivo devem ser as seguintes:

```
name=local-file-source
```

```
connector.class=FileStreamSource
```

```
tasks.max=1
```

```
file=origem.txt
```

```
topic=connect-od
```

Kafka Connect

3. Abrir no VS Code ou no gedit o arquivo

➤ `config\connect-file-sink.properties`

4. As linhas do arquivo devem ser as seguintes:

```
name=local-file-sink
```

```
connector.class=FileStreamSink
```

```
tasks.max=1
```

```
file=destino.txt
```

```
topics=connect-od
```

Kafka Connect

5. Abrir no VS Code ou no gedit o arquivo

➤ `config\connect-standalone.properties`

6. As configurações do arquivo devem ser:

```
bootstrap.servers=localhost:9092
```

```
key.converter=org.apache.kafka.connect.json.JsonConverter
```

```
value.converter=org.apache.kafka.connect.json.JsonConverter
```

```
key.converter.schemas.enable=true
```

```
value.converter.schemas.enable=true
```

Kafka Connect

7. Lembre-se de iniciar o Zookeeper e o Broker Kafka se já não estiverem sido iniciados.

8. Iniciar o Source e o Sink connectors

➤ `./bin/connect-standalone.sh`
`config/connect-standalone.properties`
`config/connect-file-source.properties`
`config/connect-file-sink.properties`

Kafka Connect

9. Em outro terminal, observar o arquivo destino.txt:

➤ `ls`

➤ `cat destino.txt`

Kafka Connect

10. Criar um arquivo origem.txt:

- `touch origem.txt`

11. Inserir dados no arquivo

- `echo 'hello' >> origem.txt`

- `echo 'salut' >> origem.txt`

12. Visualizar o que acontece no arquivo destino.txt

- `cat destino.txt`

Kafka Connect

13. Visualizar o conector no browser (Firefox)

`http://localhost:8083/connectors/local-file-sink`

14. Deletando um conector:

➤ `curl -X DELETE`

`http://localhost:8083/connectors/local-file-sink`

Kafka Web Project

Kafka Project

1. Criar pasta do projeto

➤ `mkdir ~/live-map`

2. Acessar via VSCode

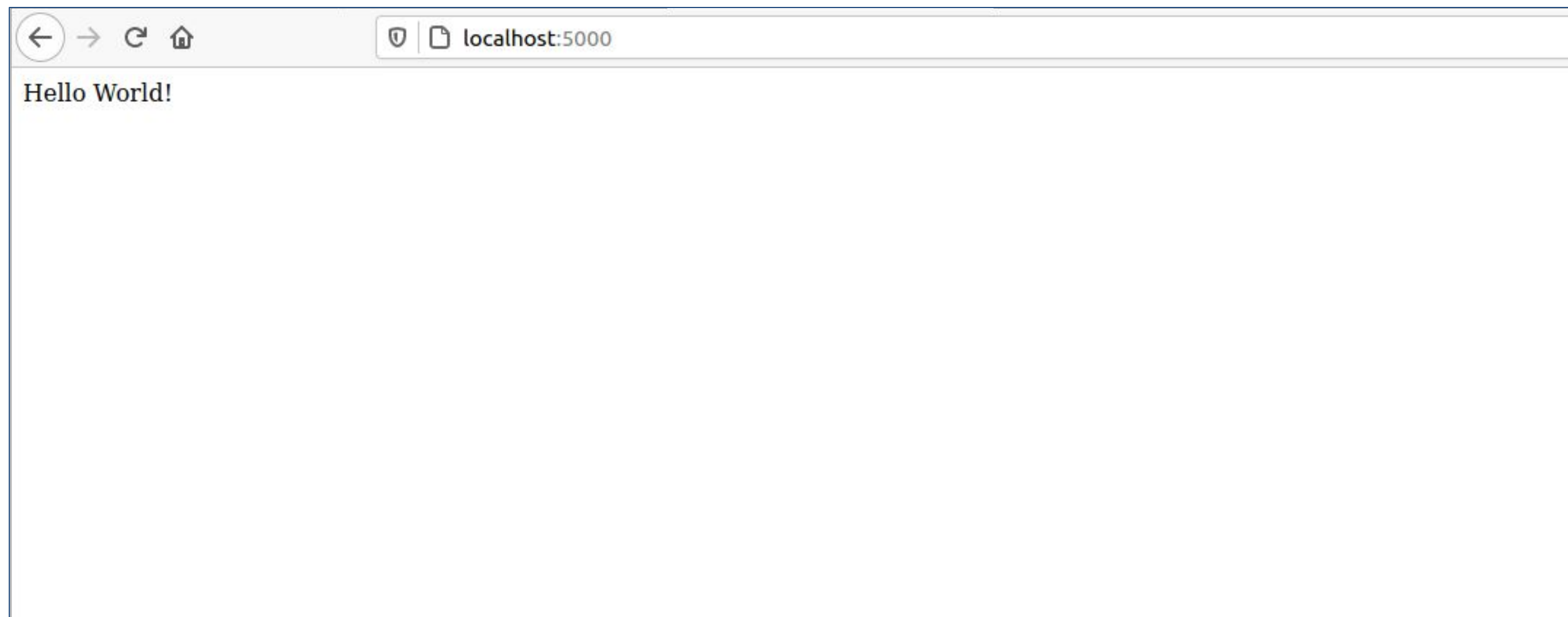
➤ `code ~/live-map`

3. Ainda no terminal, instalar o Flask

➤ `pip3 install flask`

Kafka Project

5. No navegador, Firefox por exemplo, acessar:
<http://localhost:5000/>



Kafka Project

6. No VSCode criar uma pasta chamada 'templates'
7. Criar arquivo index.html

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <!-- LEAFLET -->
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css"
integrity="sha512-xodZBNTC5n17Xt2atTPuE1HxjVMSvLVW9ocqUKLsCC5CXdbqCmblAshOMAS6/keqq/sMZMZ19scR4PsZChSR7A=="
crossorigin=""/>

    <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"
integrity="sha512-XQoYMqMTK8LvdxXYG3nZ448hOEqIglfjqkJs1NOQV44cWnUrBc8PkAOcXy20w0vlaXaVUearIOBhiXZ5V3ynxwA=="
crossorigin=""></script>
    <!-- END LEAFLET -->

    <title>Live Map</title>
  </head>
  <body>
    <h1>Bus Live Map</h1>

    <!-- LEAFLET -->
    <div id="mapid" style = "width:900px; height:580px;"></div>
    <script src="../static/livemap.js"></script>
    <!-- END LEAFLET -->
  </body>
</html>
```

Kafka Project

8. Cadastrar um token no mapbox

<https://account.mapbox.com/>



Dashboard

Tokens

Statistics

Invoices

Settings



Access tokens

[+ Create a token](#)

You need an API access token to configure [Mapbox GL JS](#), [Mobile](#), and [Mapbox web services](#) like routing and geocoding. Read more about [API access tokens](#) in our documentation.

Kafka Project

9. No VSCode criar uma pasta chamada 'static'

10. Criar arquivo livemap.js

livemap.js

```
var mymap = L.map('mapid').setView([51.505, -0.09], 13);

L.tileLayer('https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token={accessToken}', {
  attribution: 'Map data &copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors, Imagery © <a
href="https://www.mapbox.com/">Mapbox</a>',
  maxZoom: 18,
  id: 'mapbox/streets-v11',
  tileSize: 512,
  zoomOffset: -1,
  accessToken: '<INSERT_TOKEN_AQUI>'
}).addTo(mymap);
```

Kafka Project

11. Alterar o app.py

```
from flask import Flask, render_template  
  
...  
  
@app.route("/")  
def index():  
    return(render_template('index.html'))
```

Kafka Project

12. Criar uma pasta chamada 'data'

13. Via terminal, fazer o download dos dados json para a pasta 'data'

➤ `cd ~/live-map/data/`

➤ `wget http://www.lia.ufc.br/~timbo/streaming/bus1.json`

Kafka Project

14. Criar um arquivo chamado producer1.py na raiz do projeto

producer1.py

```
from kafka import KafkaProducer
import json
from datetime import datetime
import uuid
import time

#LER AS COORDENADAS GEOJSON
input_file = open('./data/bus1.json')
json_array = json.load(input_file)
coordinates = json_array['features'][0]['geometry']['coordinates']

#GERAR ID UNICO
def generate_uuid():
    return uuid.uuid4()

#KAFKA PRODUCER
producer = KafkaProducer(bootstrap_servers='localhost:9092')
```

producer1.py

#CONSTRUIR A MENSAGEM E ENVIAR VIA KAFKA

```
data = {}
```

```
data['busline'] = '00001'
```

```
def generate_checkpoint(coordinates):
```

```
    i = 0
```

```
    while i < len(coordinates):
```

```
        data['key'] = data['busline'] + '_' + str(generate_uuid())
```

```
        data['timestamp'] = str(datetime.utcnow())
```

```
        data['latitude'] = coordinates[i][1]
```

```
        data['longitude'] = coordinates[i][0]
```

```
        message = json.dumps(data)
```

```
        print(message)
```

```
        producer.send('busao', message.encode('ascii'))
```

```
        # producer.produce(message.encode('ascii'))
```

```
        time.sleep(1)
```

```
    #if bus reaches last coordinate, start from beginning
```

```
    if i == len(coordinates)-1:
```

```
        i = 0
```

```
    else:
```

```
        i += 1
```

```
generate_checkpoint(coordinates)
```


Kafka Project

15. Executar o arquivo producer1.py

➤ `python3 producer1.py`

16. Em um outro terminal, verificar se os dados estão chegando via consumidor

➤ `bin/kafka-console-consumer.sh --bootstrap-server
localhost:9092 --topic busao`

Kafka Project

17. Criar o consumidor no arquivo 'app.py'

```
from flask import Flask, render_template, Response, stream_with_context
from kafka import KafkaConsumer
...
@app.route('/topic/<topicname>')
def streamed_response(topicname):
    stream = KafkaConsumer(topicname, bootstrap_servers='localhost:9092')
    def generate():
        for i in stream:
            yield "<p>{}</p>".format(i.value.decode())
    return Response(stream_with_context(generate()))
```

Kafka Project

18. Criar um produtor para o consumidor anterior

➤ `bin/kafka-console-producer.sh --broker-list
localhost:9092 --topic busao2`

Kafka Project

19. Adicionar ao livemap.js o código a seguir

20. Verificar o arquivo final app.py

livemap.js

```
var source = new EventSource('/topic/busao); // NOME DO TOPICO
source.addEventListener('message', function(e){

    console.log('Message');
    obj = JSON.parse(e.data);
    console.log(obj);

    if(obj.busline == '00001') {
        for (var i = 0; i < mapMarkers1.length; i++) {
            mymap.removeLayer(mapMarkers1[i]);
        }
        marker1 = L.marker([obj.latitude, obj.longitude]).addTo(mymap);
        mapMarkers1.push(marker1);
    }

}, false);
```

app.py

```
from flask import Flask, render_template, Response, stream_with_context
from kafka import KafkaConsumer

from flask_kafka import FlaskKafka

app = Flask(__name__)

@app.route('/')
def index():
    return(render_template('index.html'))

@app.route('/topic/<topicname>')
def streamed_response(topicname):
    stream = KafkaConsumer(topicname, bootstrap_servers='localhost:9092')

    def generate():
        for i in stream:
            yield 'data:{0}\n\n'.format(i.value.decode())

    return Response(stream_with_context(generate()), mimetype="text/event-stream")

if __name__ == "__main__":
    app.run()
```

Dúvidas?