

Streaming de Dados em Tempo Real: Aula 2

Prof. Felipe Timbó



Ementa (dia 2)

- Data Ingestion com Apache Kafka
- Kafka Web Project

Apache Kafka

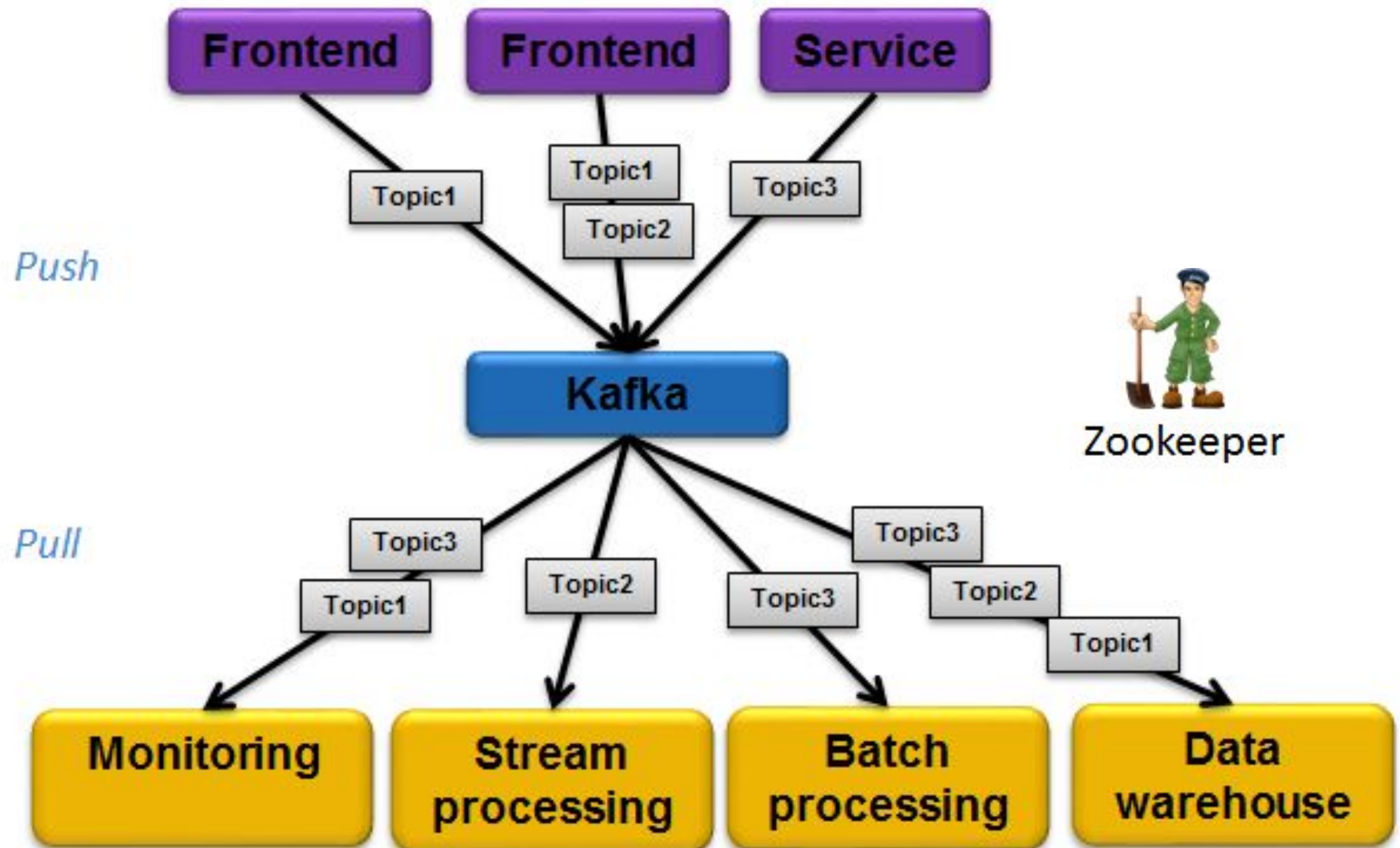
(cont.)

Kafka: relembrando

Producers

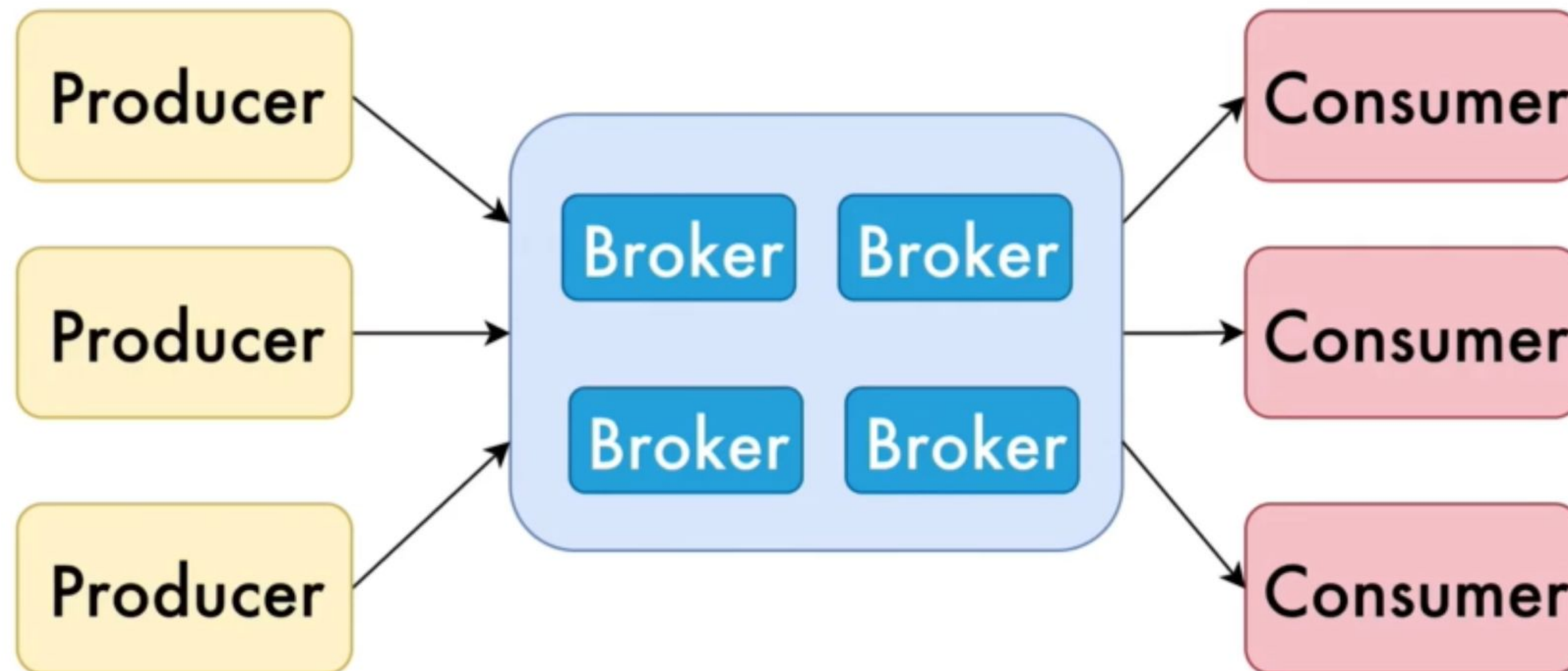
Broker

Consumers



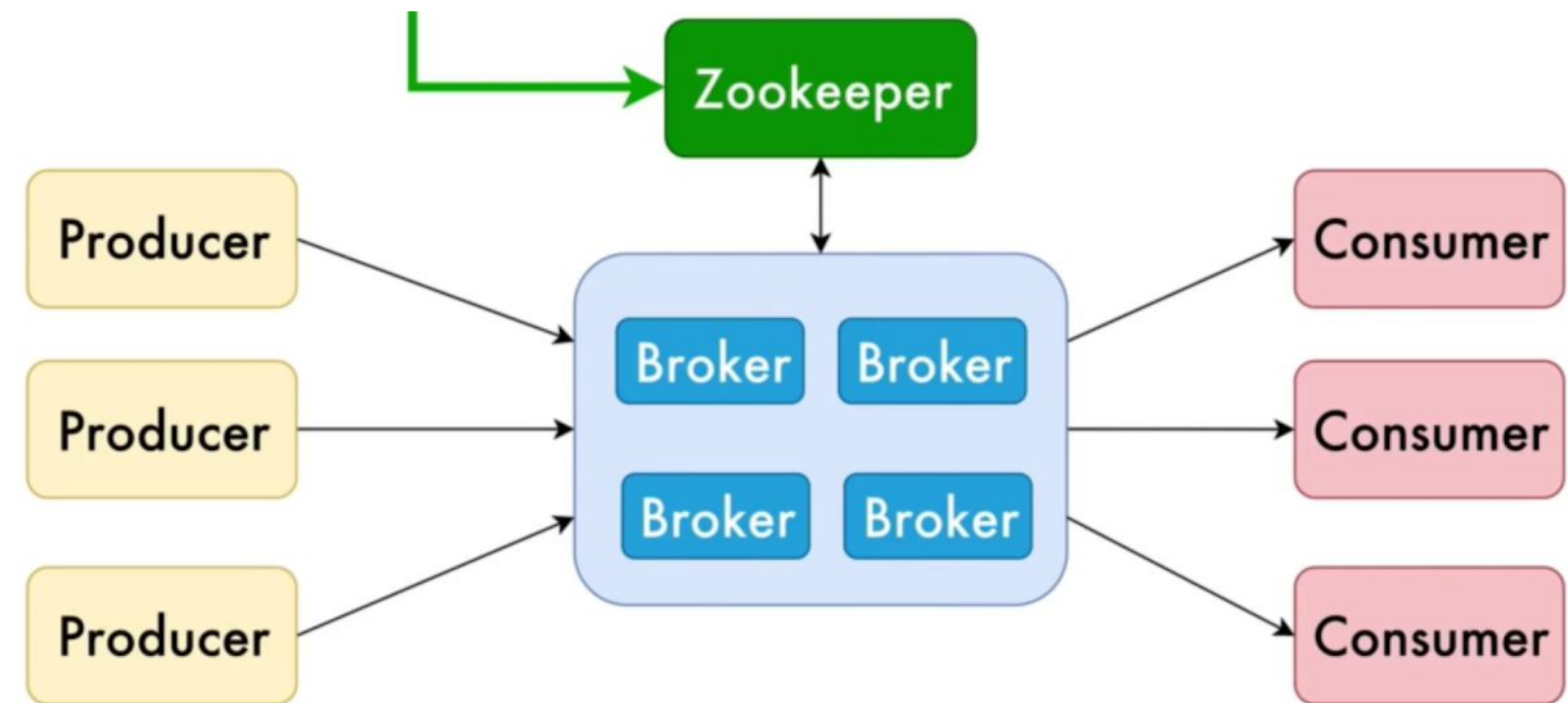
Kafka na vida real

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



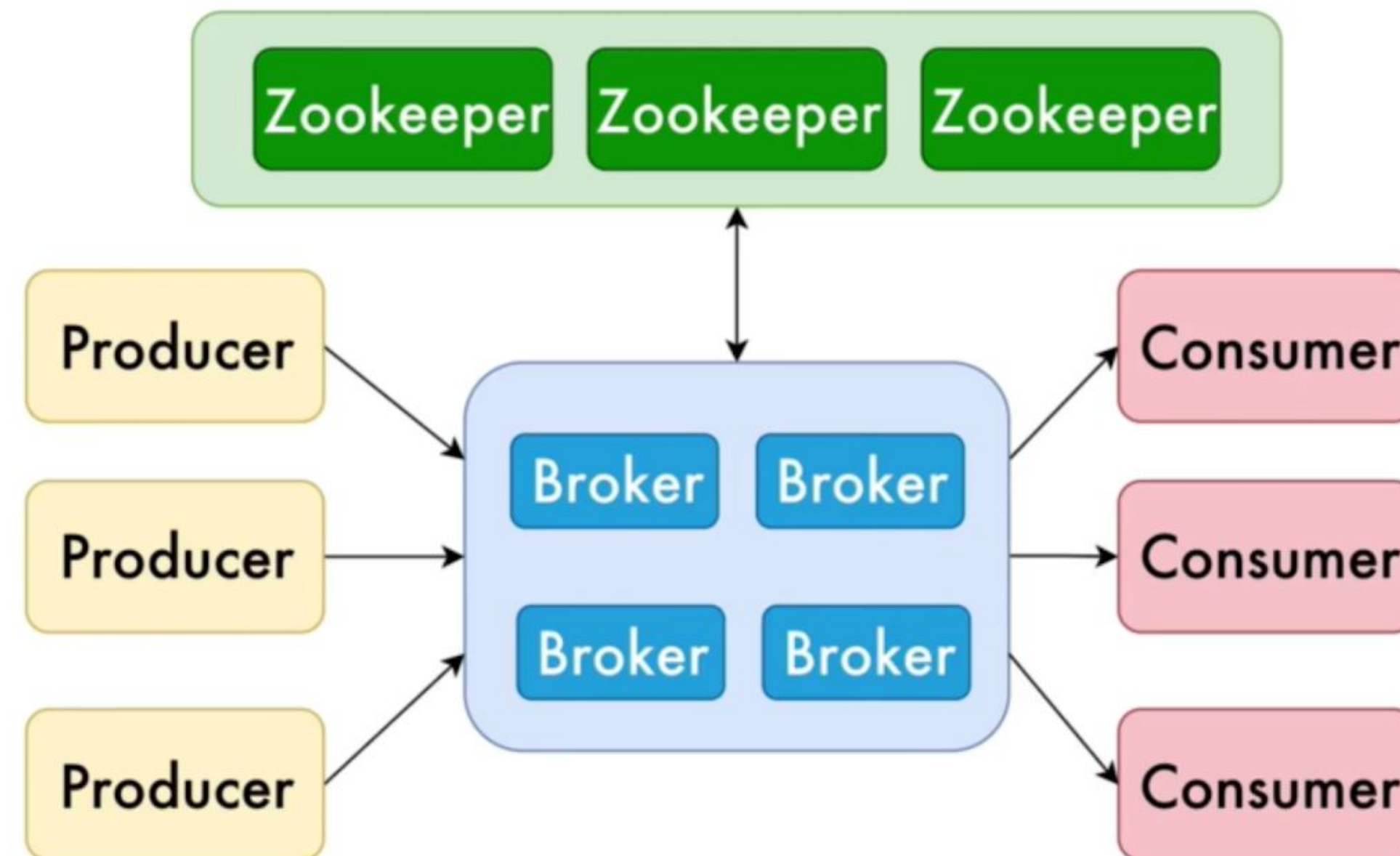
Zookeeper

- 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.



Instalando o Apache Kafka

Realizar o download do Apache Kafka:

➤ `curl`

```
http://ftp.unicamp.br/pub/apache/kafka/2.8.1/kafka_2.12-2.8.1.tgz -o ~/Downloads/kafka.tgz
```

```
% Total      % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             %         Dload  Upload   Total   Spent    Left     Speed
100 62.6M  100 62.6M    0     0   528k      0  0:02:01  0:02:01 --:--:-- 345k
posgrad@posgrad-vm:~$
```


Instalando o Apache Kafka

Criar um diretório chamado kafka e entrar nele:

- `mkdir kafka`
- `cd kafka`

Extrair os arquivos que estão na pasta download para o diretório criado:

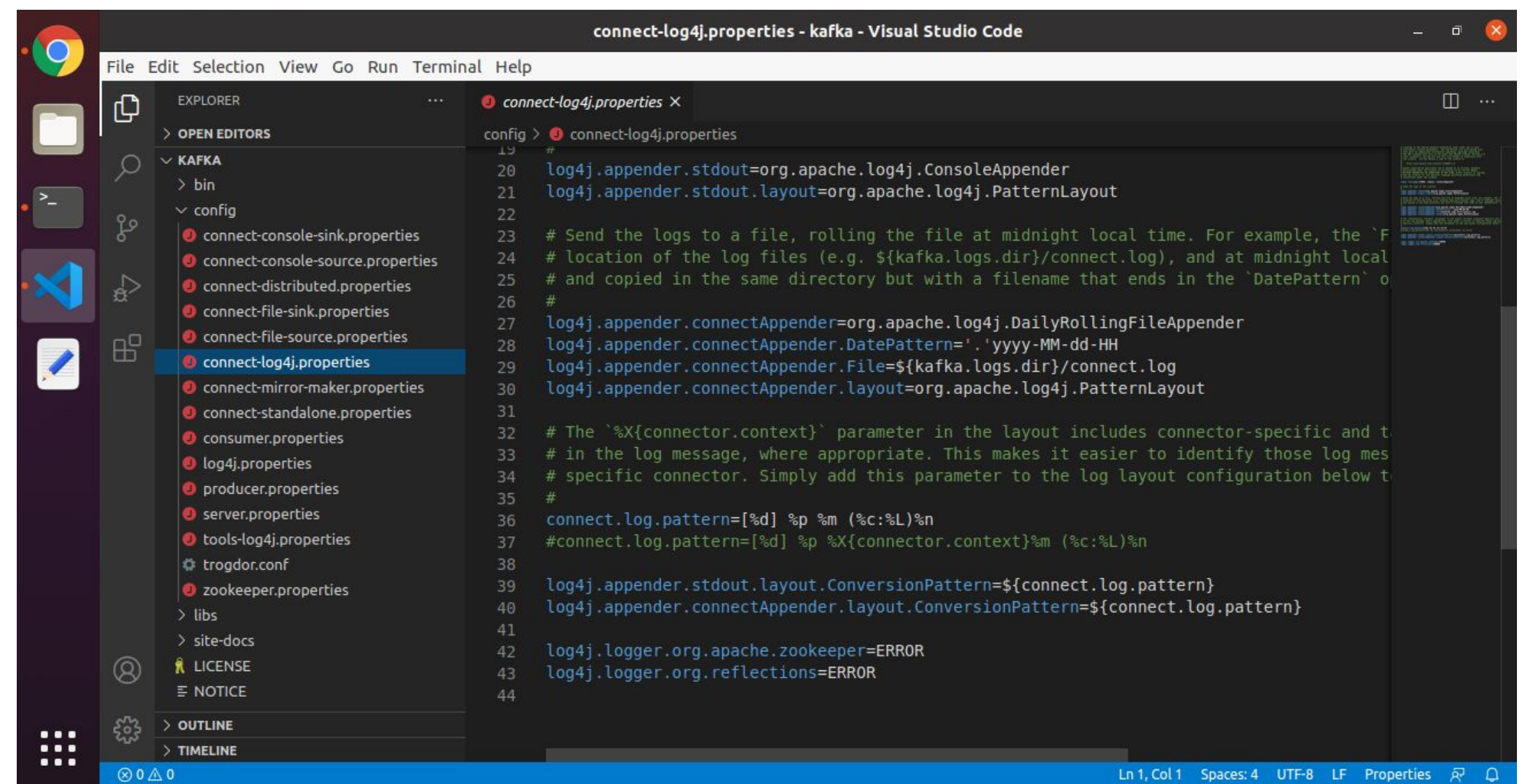
- `tar -xvzf ~/Downloads/kafka.tgz --strip 1`

Visualizando os arquivos do Kafka

Abrir o VS Code

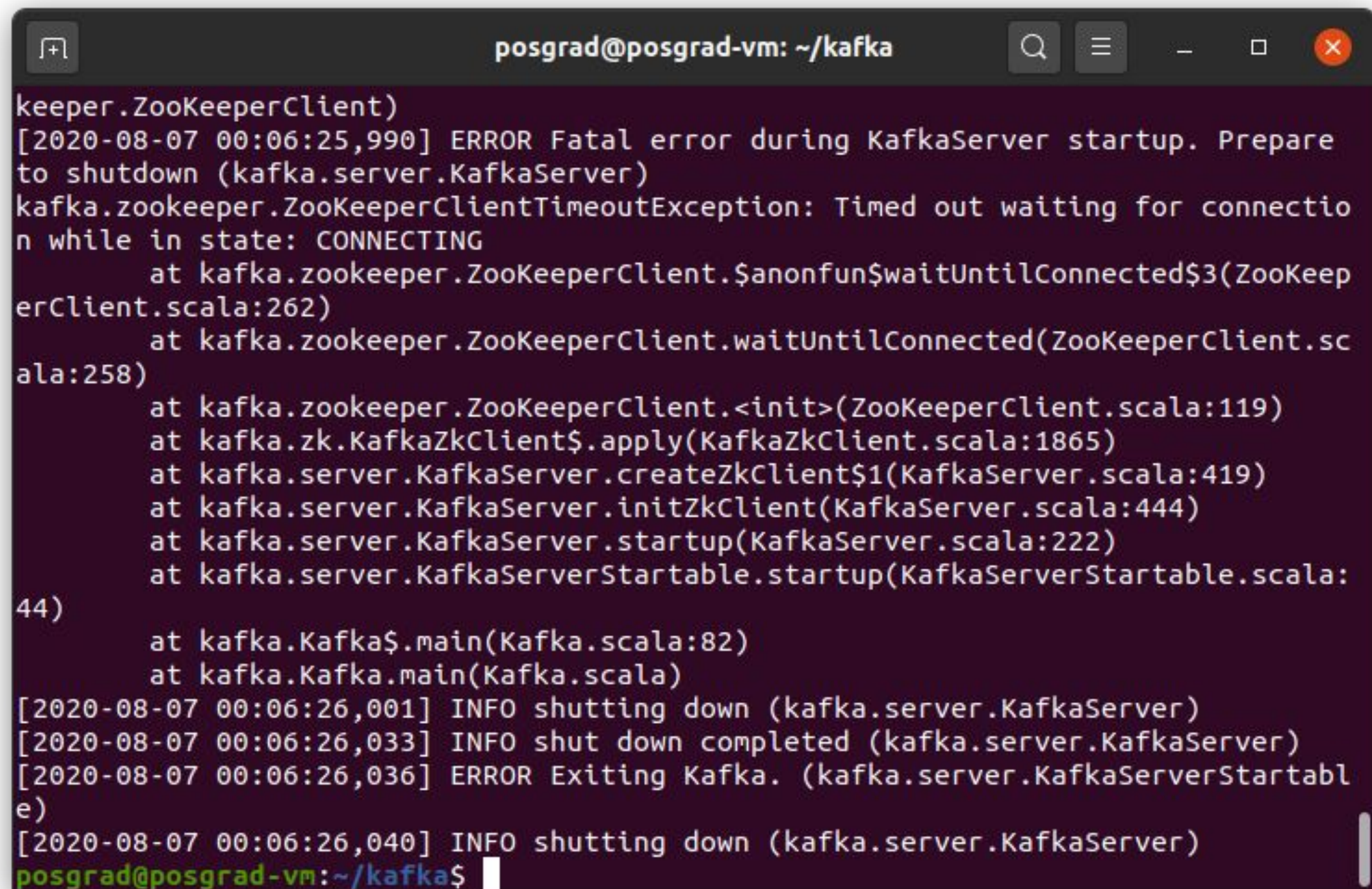
Clicar em Open Folder

Selecionar a pasta kafka



Inicializando o Servidor Kafka

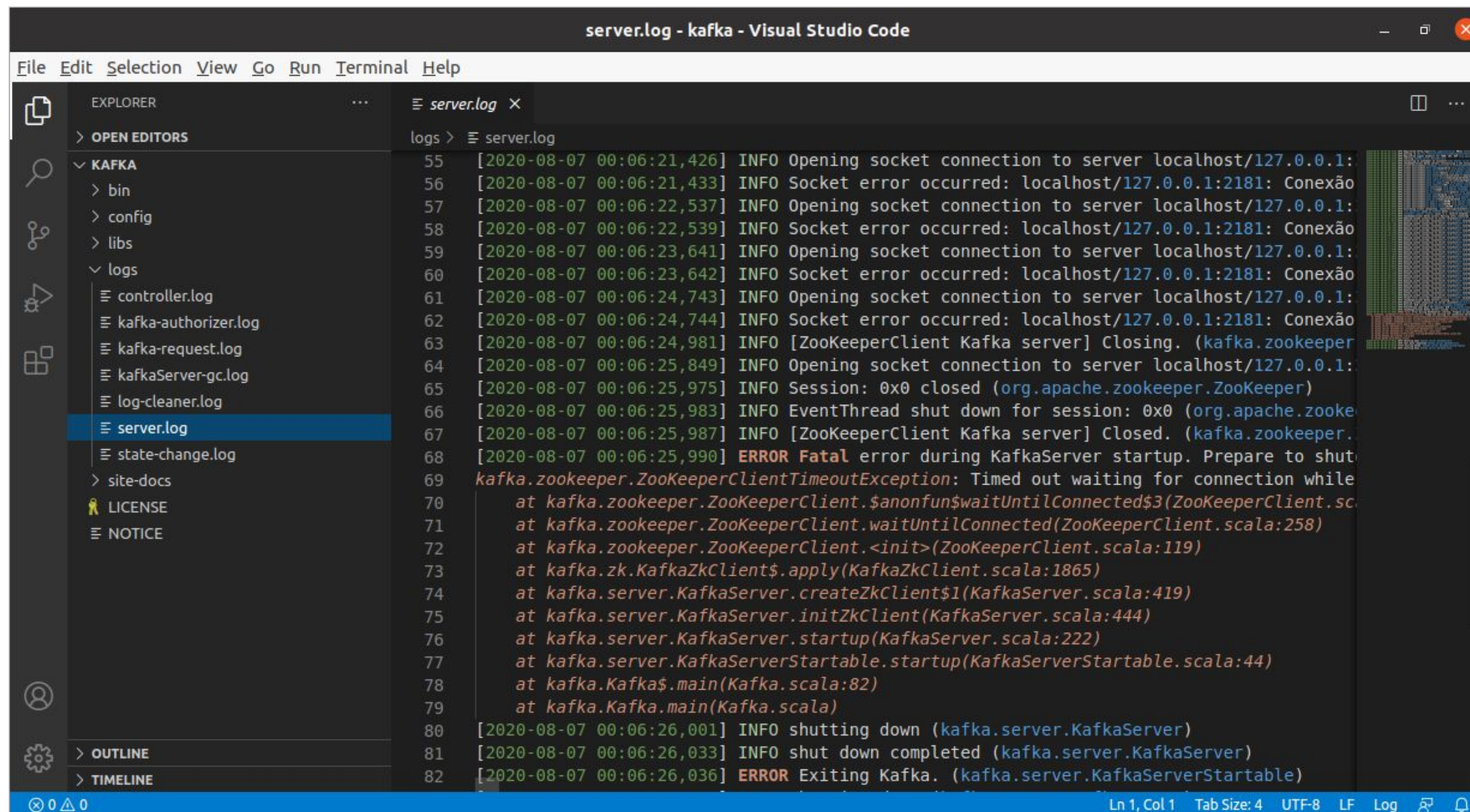
➤ `bin/kafka-server-start.sh config/server.properties`

A terminal window titled 'posgrad@posgrad-vm: ~/kafka' showing the execution of the 'kafka-server-start.sh' command. The output displays a fatal error: 'kafka.zookeeper.ZooKeeperClientTimeoutException: Timed out waiting for connection while in state: CONNECTING'. The stack trace indicates the failure occurs during the initialization of the ZooKeeper client within the Kafka server startup process. Subsequent log messages show the server shutting down and exiting.

```
keeper.ZooKeeperClient)
[2020-08-07 00:06:25,990] ERROR Fatal error during KafkaServer startup. Prepare
to shutdown (kafka.server.KafkaServer)
kafka.zookeeper.ZooKeeperClientTimeoutException: Timed out waiting for connectio
n while in state: CONNECTING
    at kafka.zookeeper.ZooKeeperClient.$anonfun$waitUntilConnected$3(ZooKeep
erClient.scala:262)
    at kafka.zookeeper.ZooKeeperClient.waitUntilConnected(ZooKeeperClient.sc
ala:258)
    at kafka.zookeeper.ZooKeeperClient.<init>(ZooKeeperClient.scala:119)
    at kafka.zk.KafkaZkClient$.apply(KafkaZkClient.scala:1865)
    at kafka.server.KafkaServer.createZkClient$1(KafkaServer.scala:419)
    at kafka.server.KafkaServer.initZkClient(KafkaServer.scala:444)
    at kafka.server.KafkaServer.startup(KafkaServer.scala:222)
    at kafka.server.KafkaServerStartable.startup(KafkaServerStartable.scala:
44)
    at kafka.Kafka$.main(Kafka.scala:82)
    at kafka.Kafka.main(Kafka.scala)
[2020-08-07 00:06:26,001] INFO shutting down (kafka.server.KafkaServer)
[2020-08-07 00:06:26,033] INFO shut down completed (kafka.server.KafkaServer)
[2020-08-07 00:06:26,036] ERROR Exiting Kafka. (kafka.server.KafkaServerStartabl
e)
[2020-08-07 00:06:26,040] INFO shutting down (kafka.server.KafkaServer)
posgrad@posgrad-vm:~/kafka$
```


Observando os logs de erro do kafka

No VS Code, abrir a pasta logs, e o arquivo server.log



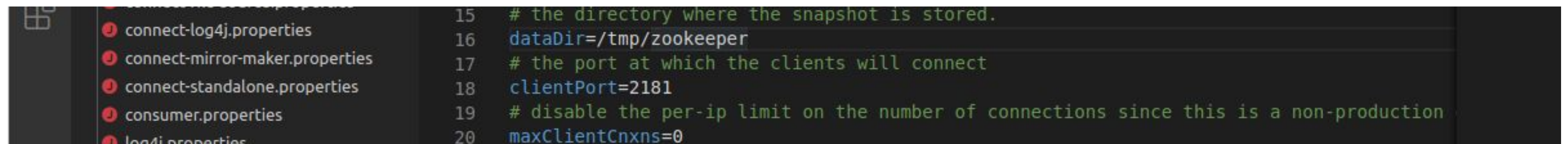
```
server.log - kafka - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
> OPEN EDITORS
KAFKA
  > bin
  > config
  > libs
  > logs
    controller.log
    kafka-authorizer.log
    kafka-request.log
    kafkaServer-gc.log
    log-cleaner.log
    server.log
    state-change.log
  > site-docs
  LICENSE
  NOTICE
OUTLINE
TIMELINE
55 [2020-08-07 00:06:21,426] INFO Opening socket connection to server localhost/127.0.0.1:
56 [2020-08-07 00:06:21,433] INFO Socket error occurred: localhost/127.0.0.1:2181: Conexão
57 [2020-08-07 00:06:22,537] INFO Opening socket connection to server localhost/127.0.0.1:
58 [2020-08-07 00:06:22,539] INFO Socket error occurred: localhost/127.0.0.1:2181: Conexão
59 [2020-08-07 00:06:23,641] INFO Opening socket connection to server localhost/127.0.0.1:
60 [2020-08-07 00:06:23,642] INFO Socket error occurred: localhost/127.0.0.1:2181: Conexão
61 [2020-08-07 00:06:24,743] INFO Opening socket connection to server localhost/127.0.0.1:
62 [2020-08-07 00:06:24,744] INFO Socket error occurred: localhost/127.0.0.1:2181: Conexão
63 [2020-08-07 00:06:24,981] INFO [ZooKeeperClient Kafka server] Closing. (kafka.zookeeper
64 [2020-08-07 00:06:25,849] INFO Opening socket connection to server localhost/127.0.0.1:
65 [2020-08-07 00:06:25,975] INFO Session: 0x0 closed (org.apache.zookeeper.ZooKeeper)
66 [2020-08-07 00:06:25,983] INFO EventThread shut down for session: 0x0 (org.apache.zooke
67 [2020-08-07 00:06:25,987] INFO [ZooKeeperClient Kafka server] Closed. (kafka.zookeeper.
68 [2020-08-07 00:06:25,990] ERROR Fatal error during KafkaServer startup. Prepare to shut
69 kafka.zookeeper.ZooKeeperClientTimeoutException: Timed out waiting for connection while
70   at kafka.zookeeper.ZooKeeperClient.$anonfun$waitUntilConnected$3(ZooKeeperClient.sc
71   at kafka.zookeeper.ZooKeeperClient.waitUntilConnected(ZooKeeperClient.scala:258)
72   at kafka.zookeeper.ZooKeeperClient.<init>(ZooKeeperClient.scala:119)
73   at kafka.zk.KafkaZkClient$.apply(KafkaZkClient.scala:1865)
74   at kafka.server.KafkaServer.createZkClient$1(KafkaServer.scala:419)
75   at kafka.server.KafkaServer.initZkClient(KafkaServer.scala:444)
76   at kafka.server.KafkaServer.startup(KafkaServer.scala:222)
77   at kafka.server.KafkaServerStartable.startup(KafkaServerStartable.scala:44)
78   at kafka.Kafka$.main(Kafka.scala:82)
79   at kafka.Kafka.main(Kafka.scala)
80 [2020-08-07 00:06:26,001] INFO shutting down (kafka.server.KafkaServer)
81 [2020-08-07 00:06:26,033] INFO shut down completed (kafka.server.KafkaServer)
82 [2020-08-07 00:06:26,036] ERROR Exiting Kafka. (kafka.server.KafkaServerStartable)
Ln 1, Col 1 Tab Size: 4 UTF-8 LF Log
```


Inicializando o Zookeeper

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

Zookeeper por padrão escuta a porta 2181

Observar arquivo ***zookeeper.properties*** na pasta ***config***

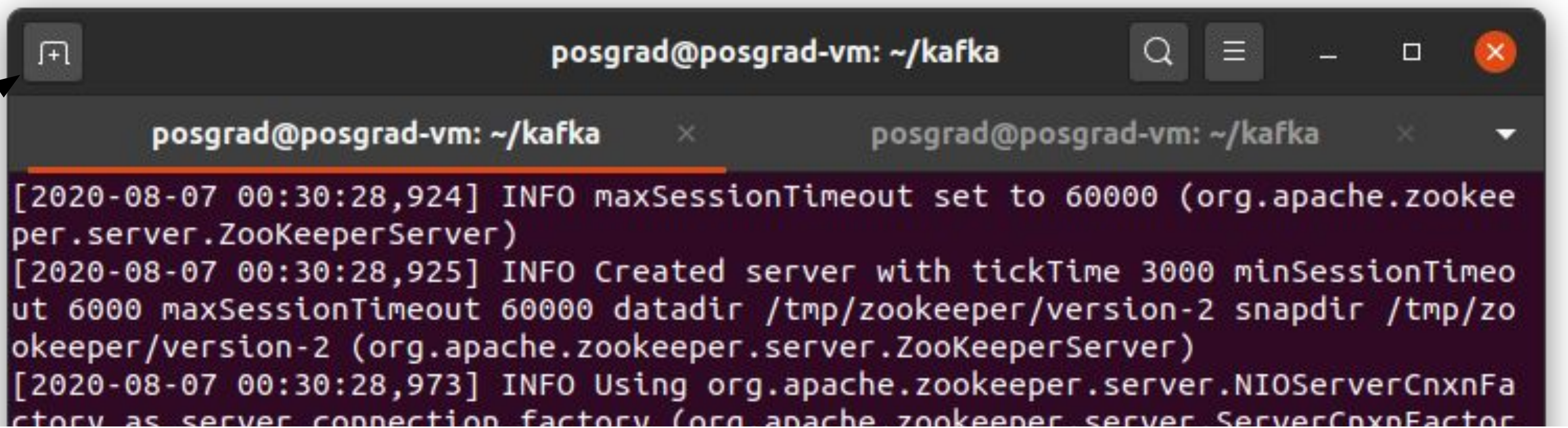


The screenshot shows a file explorer window with a sidebar on the left containing several files: connect-log4j.properties, connect-mirror-maker.properties, connect-standalone.properties, consumer.properties, and log4j.properties. The main pane displays the contents of the selected file, zookeeper.properties, with line numbers 15 through 20. The configuration includes the snapshot directory, dataDir, clientPort, and maxClientCnxns.

```
15 # the directory where the snapshot is stored.  
16 dataDir=/tmp/zookeeper  
17 # the port at which the clients will connect  
18 clientPort=2181  
19 # disable the per-ip limit on the number of connections since this is a non-production  
20 maxClientCnxns=0
```


Inicializando o Servidor Kafka (novamente)

Em outro terminal



A terminal window titled 'posgrad@posgrad-vm: ~/kafka' is shown. It contains two tabs, both with the same title. The active tab displays the following log messages:

```
[2020-08-07 00:30:28,924] INFO maxSessionTimeout set to 60000 (org.apache.zookeeper.  
per.server.ZooKeeperServer)  
[2020-08-07 00:30:28,925] INFO Created server with tickTime 3000 minSessionTimeo  
ut 6000 maxSessionTimeout 60000 datadir /tmp/zookeeper/version-2 snapdir /tmp/zo  
ookeeper/version-2 (org.apache.zookeeper.server.ZooKeeperServer)  
[2020-08-07 00:30:28,973] INFO Using org.apache.zookeeper.server.NIOServerCnxnFa  
ctory as server connection factory (org.apache.zookeeper.server.ServerCnxnFactor
```

An arrow points to the top-left corner of the terminal window, specifically to the window control buttons (minimize, maximize, close).

Inicie o Kafka novamente:

➤ `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`

Deletar os tópicos existentes:

➤ `bin/kafka-topics.sh --delete --bootstrap-server localhost:9092 --topic <nome_topico>`

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.

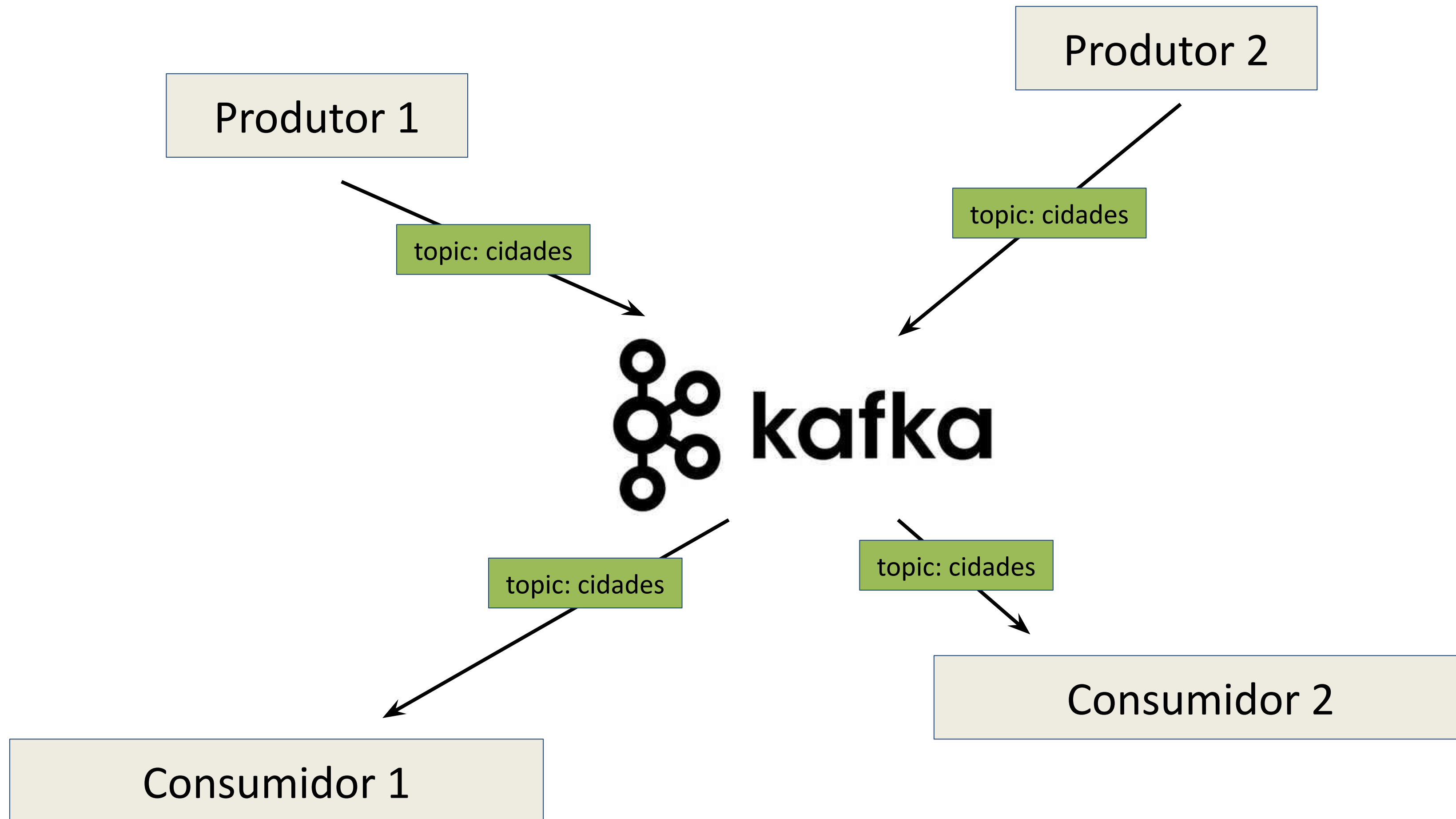
Criando um novo produtor

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

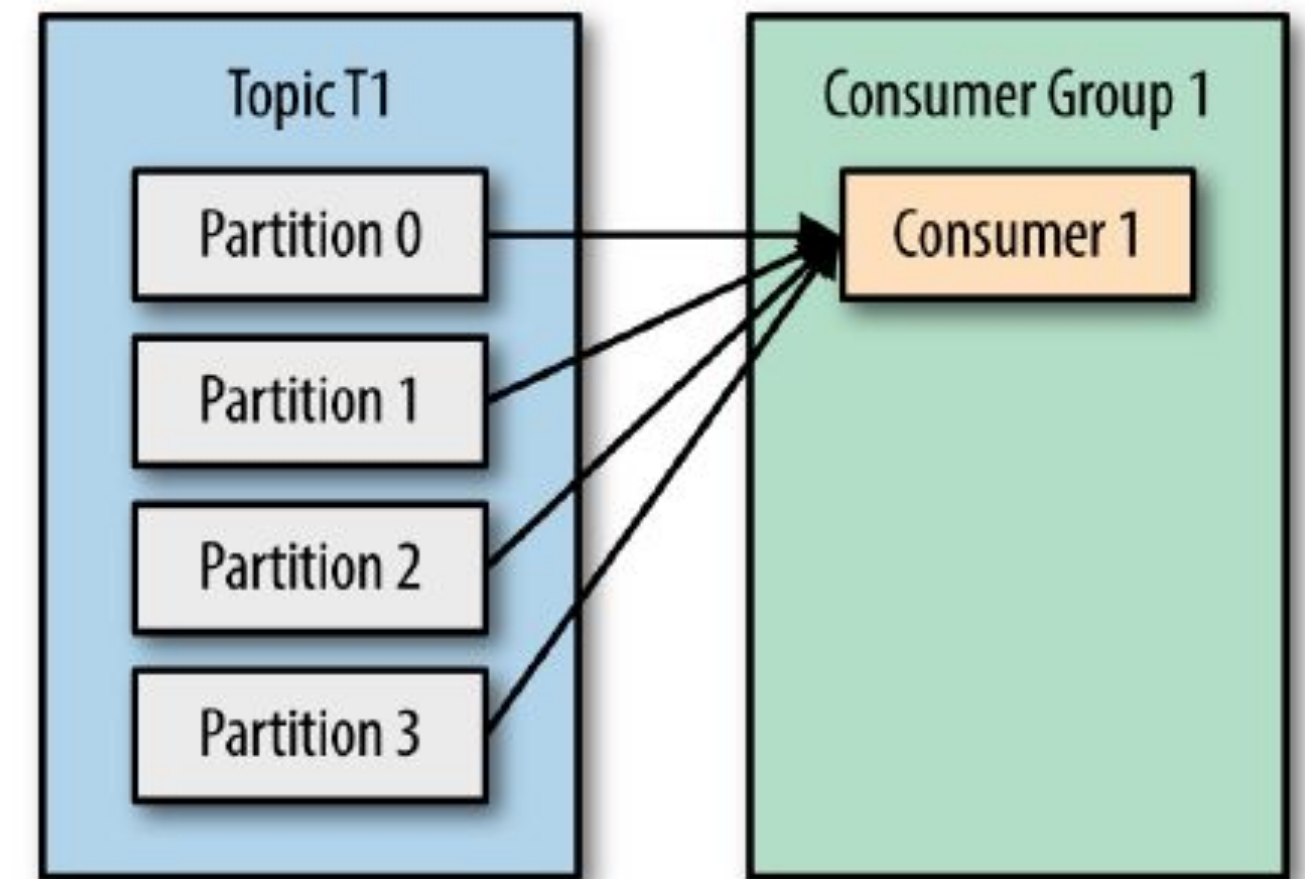
Resposta: Apenas deixam de receber mensagens do produtor excluído, mas continuam operando normalmente.

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

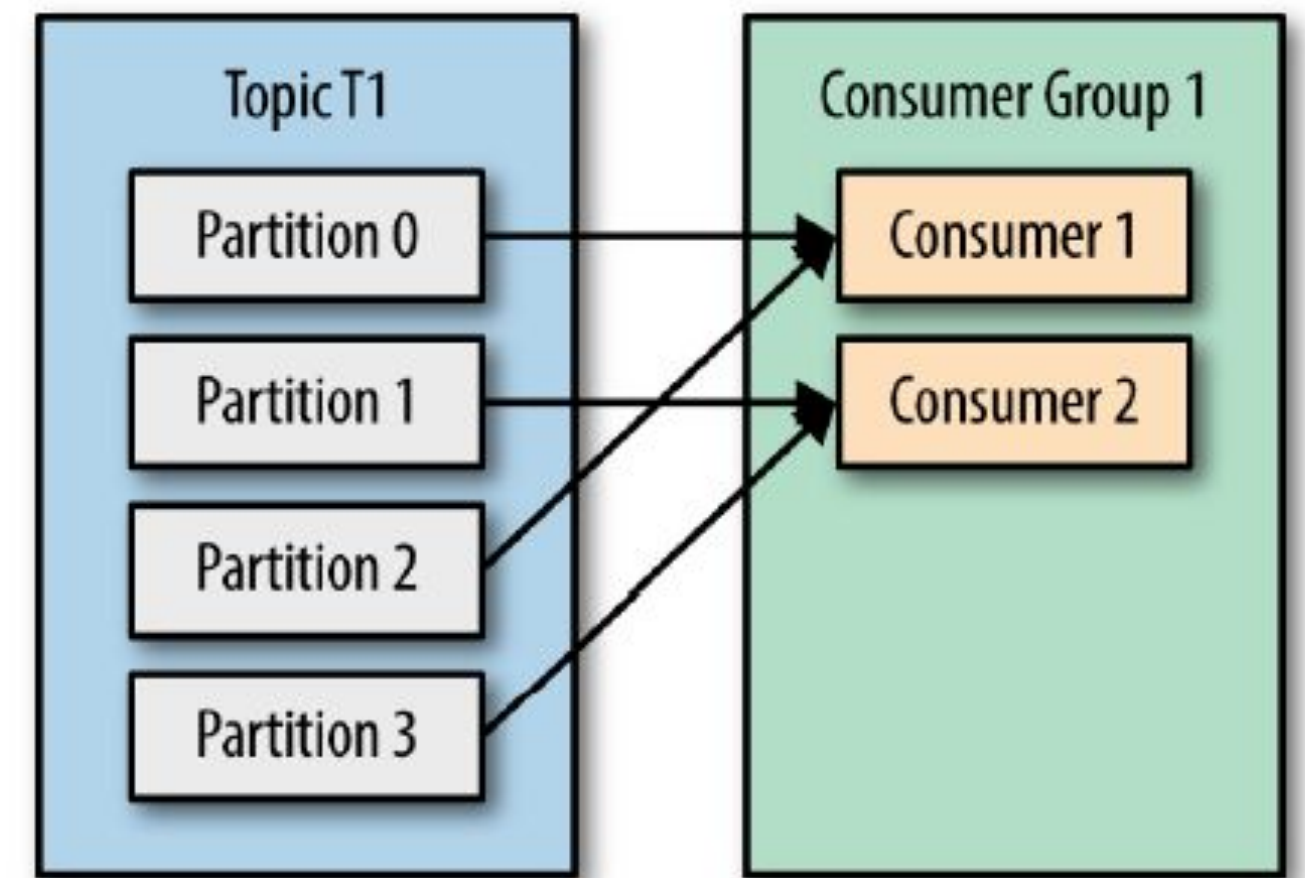
Agora, você deve deixar apenas um produtor e um consumidor

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`

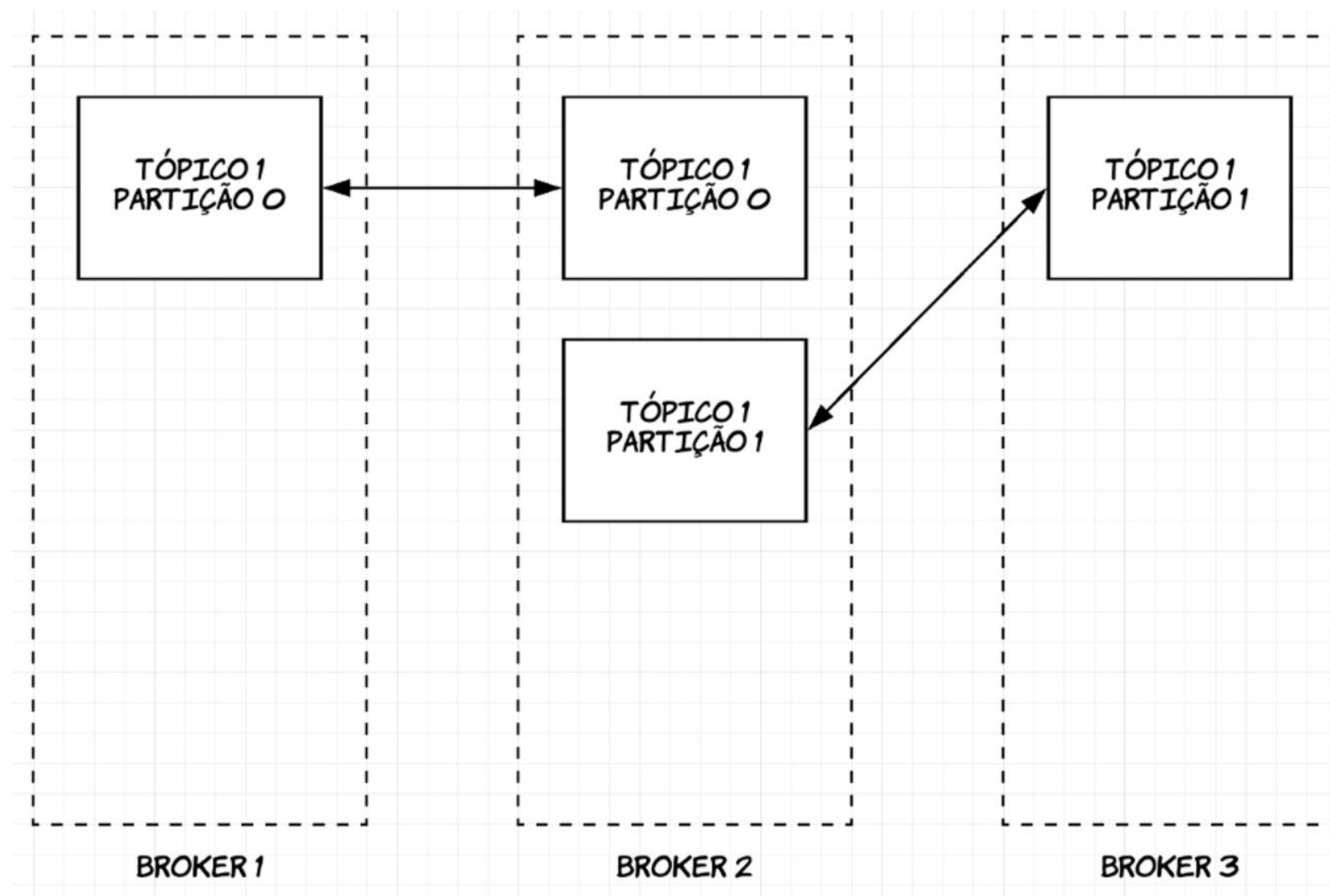
(Deixa de fora os 4 primeiros registros da partição 0)

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 e inserir mensagens: **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.

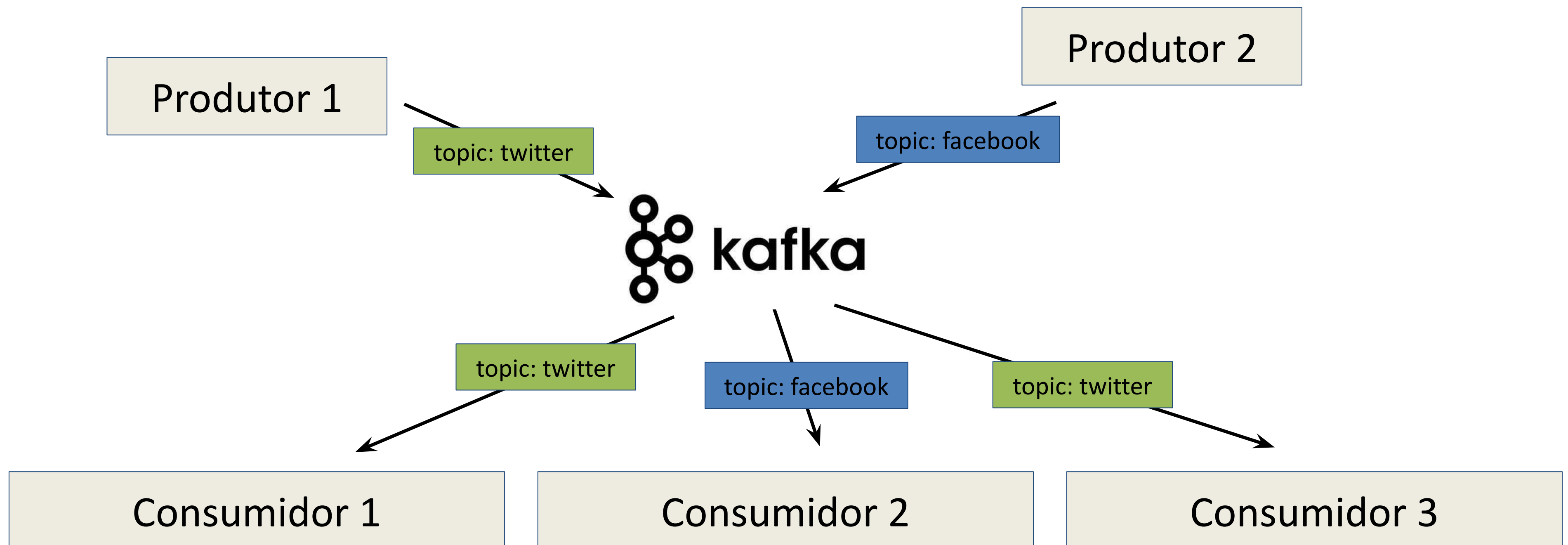
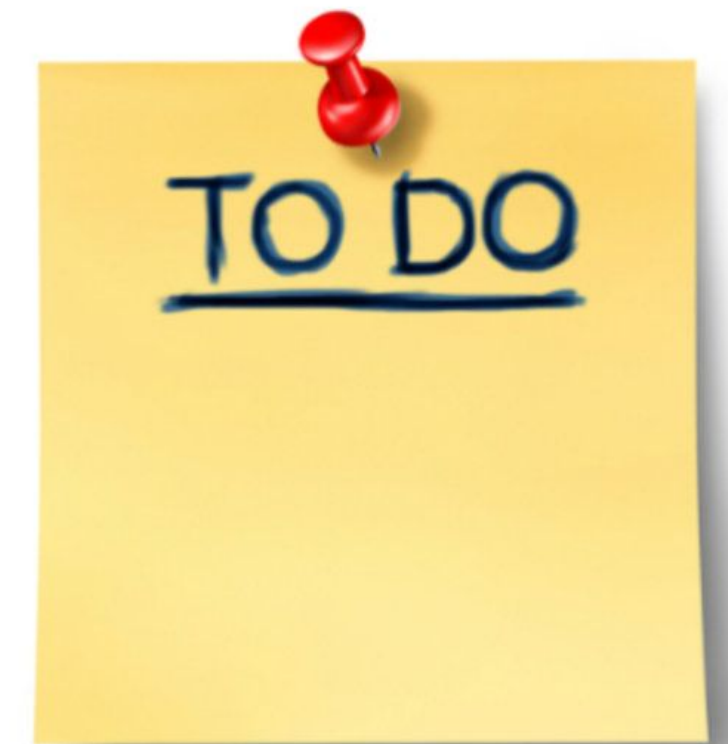
Obs.:

Endereço do zookeeper: `localhost:2181`

broker-list: `localhost:9092`

Atividade 2

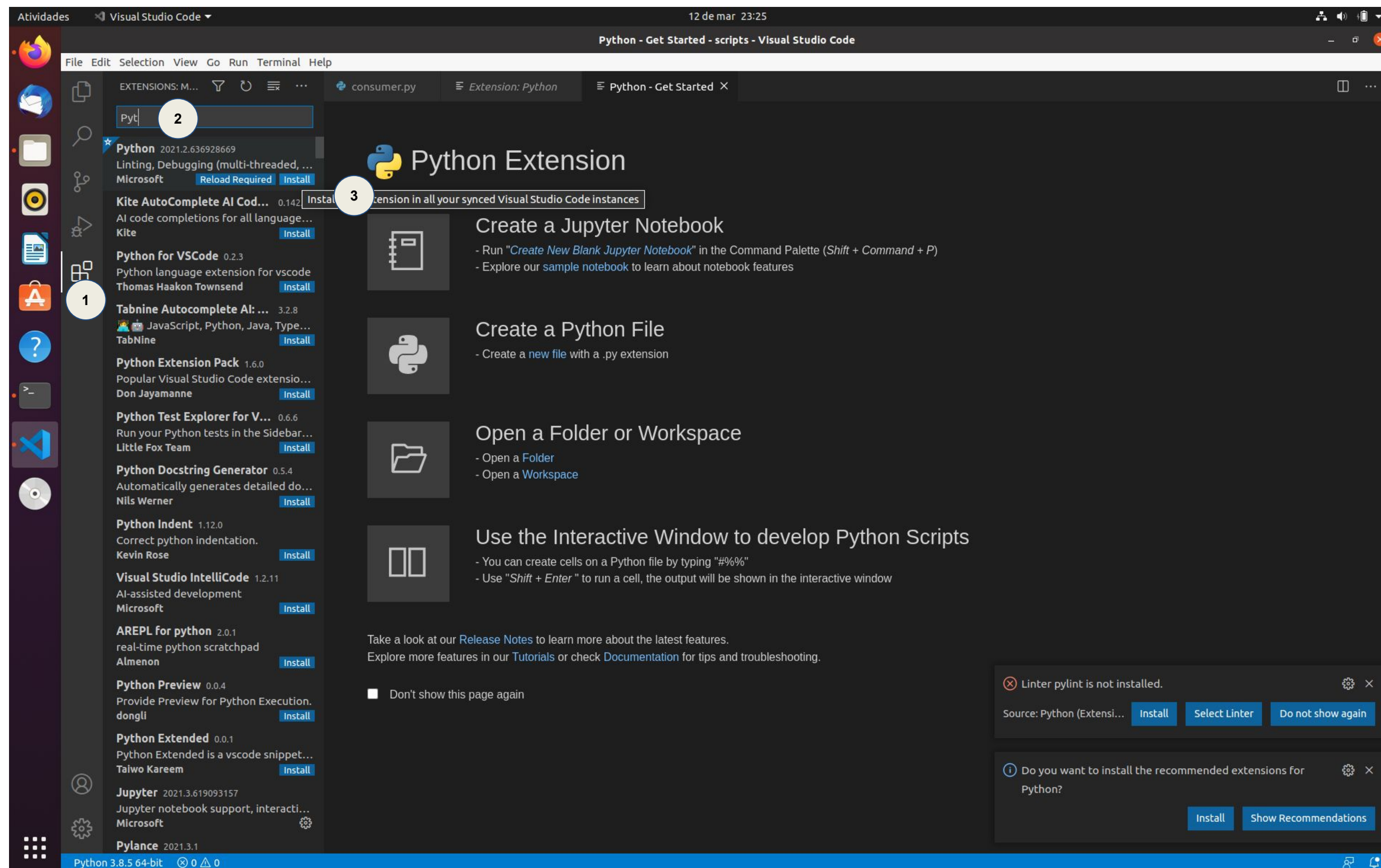
Crie o seguinte esquema de troca de mensagens via streaming de dados no Apache KAKFA:



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

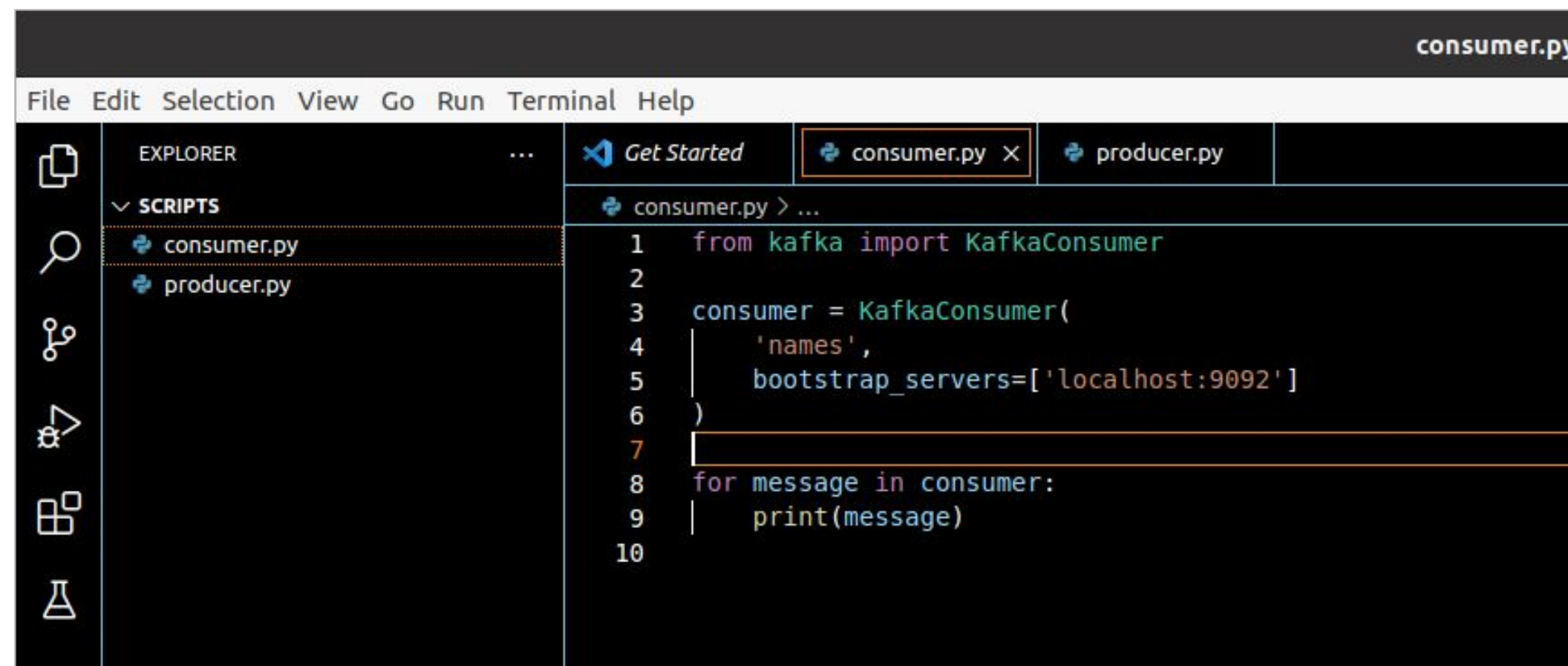
```
import time
from kafka import KafkaProducer

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

time.sleep(20)
```

Produtor/consumidor com Python

Rodar o consumer.py e em seguida o producer.py

A screenshot of a code editor interface, likely Visual Studio Code, with a dark theme. The title bar at the top right says 'consumer.py'. The menu bar includes 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help'. The Explorer sidebar on the left shows a folder named 'SCRIPTS' containing two files: 'consumer.py' and 'producer.py'. The main editor area shows the code for 'consumer.py' with line numbers 1 through 10. The code imports 'KafkaConsumer' from the 'kafka' module, creates a 'KafkaConsumer' object with 'names' and 'bootstrap_servers=[\"localhost:9092\"]', and then enters a loop to print messages from the consumer. The code is as follows:

```
1 from kafka import KafkaConsumer
2
3 consumer = KafkaConsumer(
4     'names',
5     bootstrap_servers=[\"localhost:9092\"]
6 )
7
8 for message in consumer:
9     print(message)
10
```

python3 consumer.py

python3 producer.py

(em outro terminal)

KAFKA - Prática 3

Produtor/consumidor

Instalar dependência do fake names no python

- `pip3 install Faker`

Baixar códigos python produtor/consumidor

- `cd ~/scripts`

- `wget`

- `https://raw.githubusercontent.com/felipetimbo/streaming-data-course/main/producer.py`

- `wget`

- `https://raw.githubusercontent.com/felipetimbo/streaming-data-course/main/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`

```
https://raw.githubusercontent.com/felipetimbo/streaming-data-course/  
main/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 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`

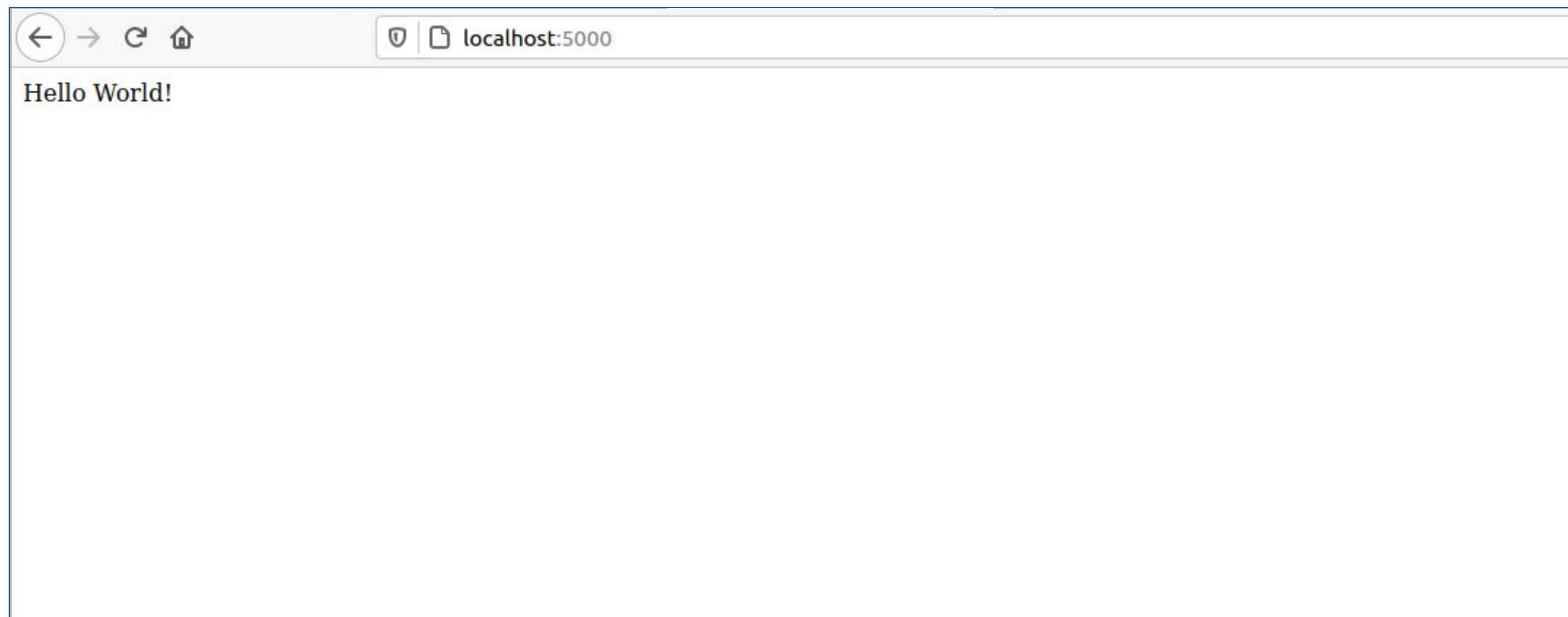
Kafka Project

4. No VSCode, criar arquivo app.py e rodar:

[illegible]

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 na pasta templates

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' na raiz do projeto
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:
'pk.eyJ1IjoiazmVsaXBldGltYm8iLCJhIjoiy2ttNnM0ZGgyMDIzZTJxcXk1bzdyZWxvZCJ9.WdumAW
A63xkYg-K6hrfXFA'
}).addTo(mymap);
```


Kafka Project

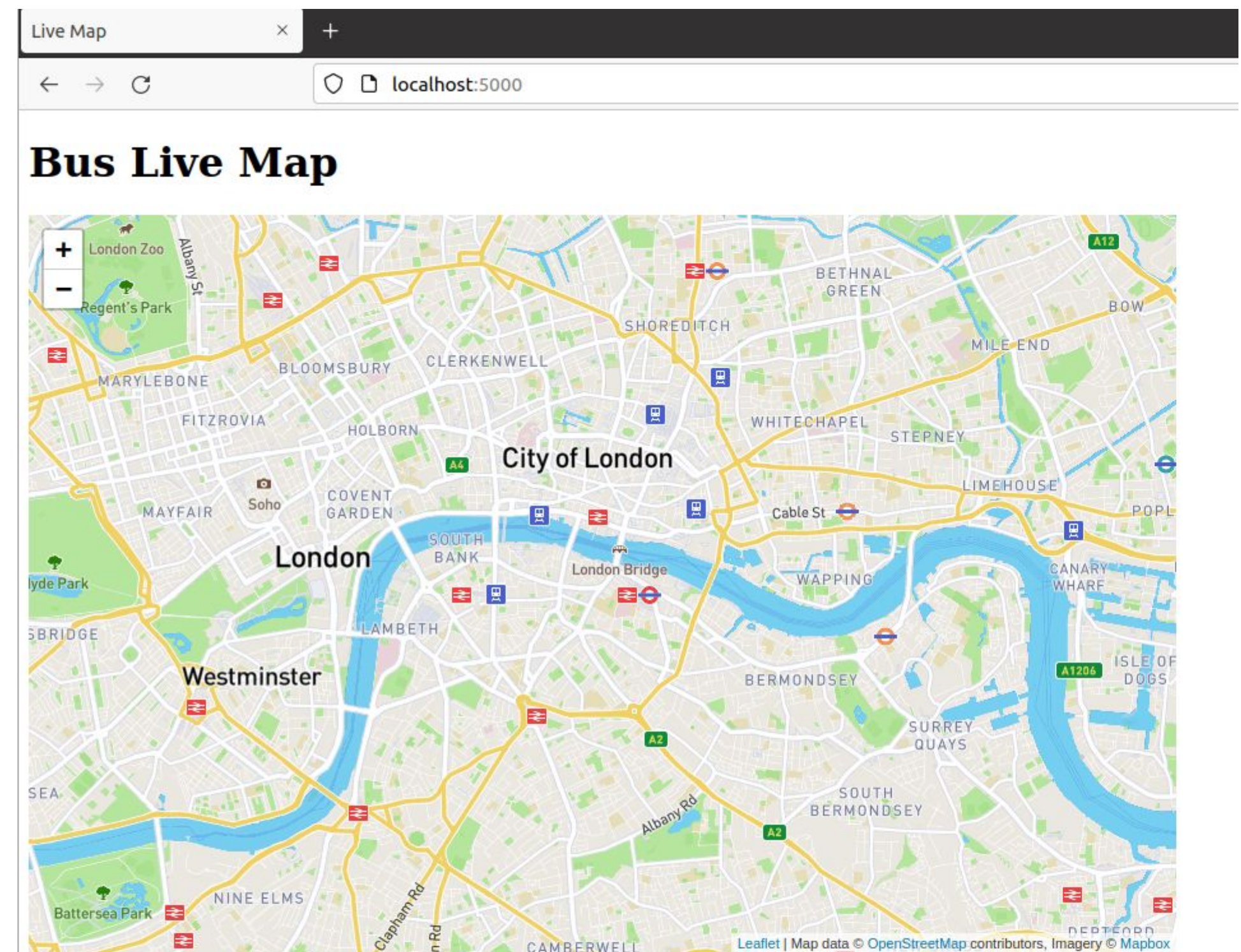
11. Alterar o app.py

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

Kafka Project

12. Derrubar a aplicação e subir novamente

13. No navegador, acessar <http://localhost:5000/> e visualizar o mapa.



Kafka Project

14. No VSCode, Criar uma pasta chamada 'data' na raiz do projeto

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

➤ `cd ~/live-map/data/`

➤ `wget`

`https://raw.githubusercontent.com/felipetimbo/streaming-data-course/main/bus1.json`

Kafka Project

16. 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

17. Executar o arquivo producer1.py

➤ `python3 producer1.py`

18. 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

19. 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

20. Criar um produtor para o consumidor anterior

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

21. Rodar a aplicação:

➤ `python3 app.py`

22. Acessar o browser em: `http://localhost:5000/topic/busao2`

23. Emitir uma mensagem no produtor criado e visualizar ela no browser

Kafka Project

- 24. Derrubar a aplicação (ctrl+c)
- 25. Adicionar ao livemap.js o código a seguir
- 26. Verificar o arquivo final app.py


```
livemap.js
```

```
var mapMarkers1 = []

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

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')

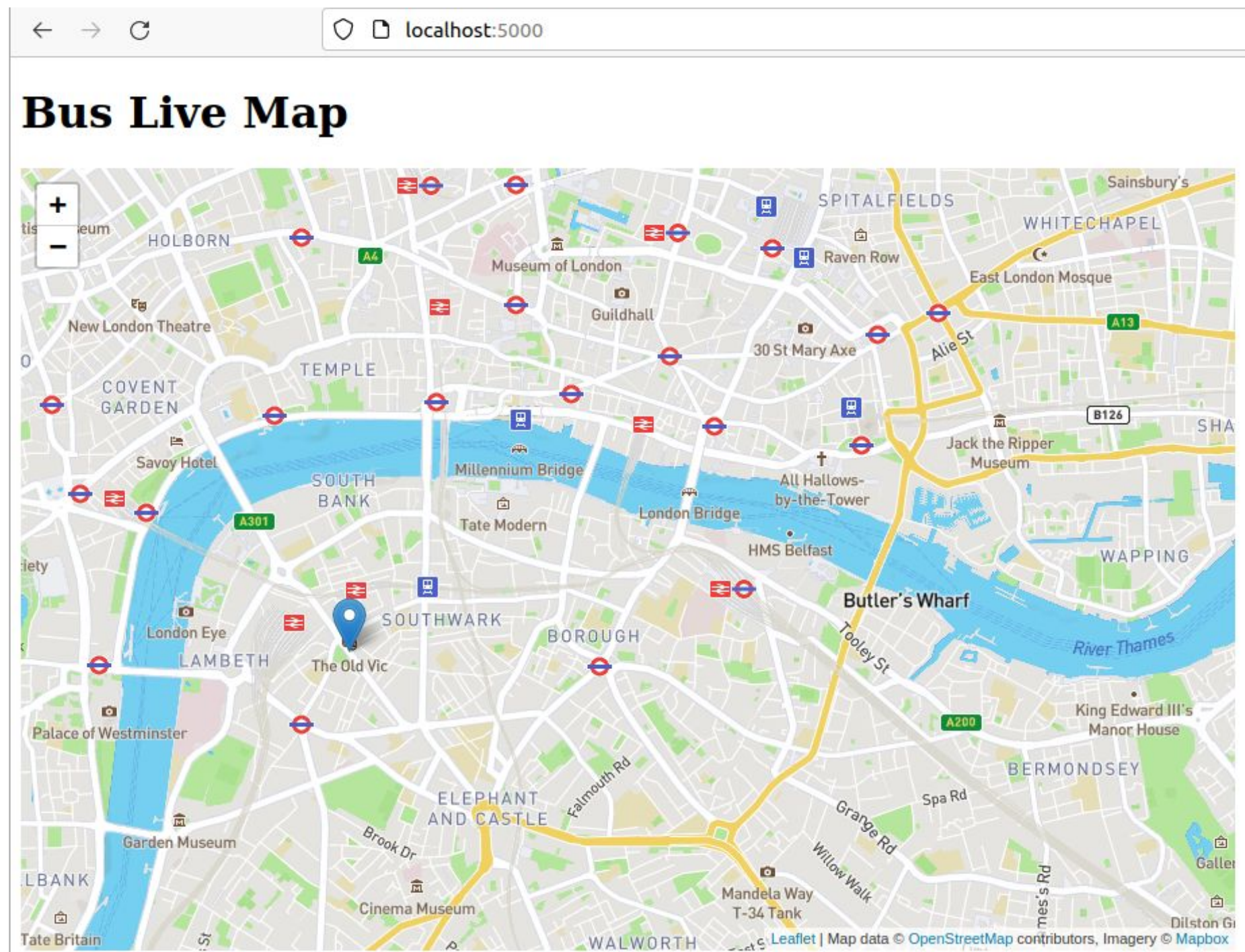
    def generate():
        for i in stream:
            yield 'data:{0}\n\n'.format(i.value.decode())

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

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

Kafka Project

27. Rodar a aplicação



Kafka Project

Para adicionar mais ônibus

28. Baixar os dados de mais dois ônibus no terminal:

➤ `cd ~/live-map/data/`

➤ `wget`

`https://raw.githubusercontent.com/felipetimbo/streaming-data-course/main/bus2.json`

➤ `wget`

`https://raw.githubusercontent.com/felipetimbo/streaming-data-course/main/bus3.json`

Kafka Project

Para adicionar mais ônibus

29. Criar producer2.py e producer3.py e modificar os ID's:

➤ `data['busline'] = '00002' # no producer2.py`

➤ `data['busline'] = '00003' # no producer3.py`

30. Alterar livemap.js

livemap.js

```
mapMarkers1 = [];  
mapMarkers2 = [];  
mapMarkers3 = [];
```

```
var source = new EventSource('/topic/busao');  
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);  
    }
```

```
    if(obj.busline == '00002') {  
        for (var i = 0; i < mapMarkers2.length; i++) {  
            mymap.removeLayer(mapMarkers2[i]);  
        }  
        marker2 = L.marker([obj.latitude, obj.longitude], {color:'red'}).addTo(mymap);  
        mapMarkers2.push(marker2);  
    }
```

```
    if(obj.busline == '00003') {  
        for (var i = 0; i < mapMarkers3.length; i++) {  
            mymap.removeLayer(mapMarkers3[i]);  
        }  
        marker3 = L.marker([obj.latitude, obj.longitude]).addTo(mymap);  
        mapMarkers3.push(marker3);  
    }  
}, false);
```


Dúvidas?