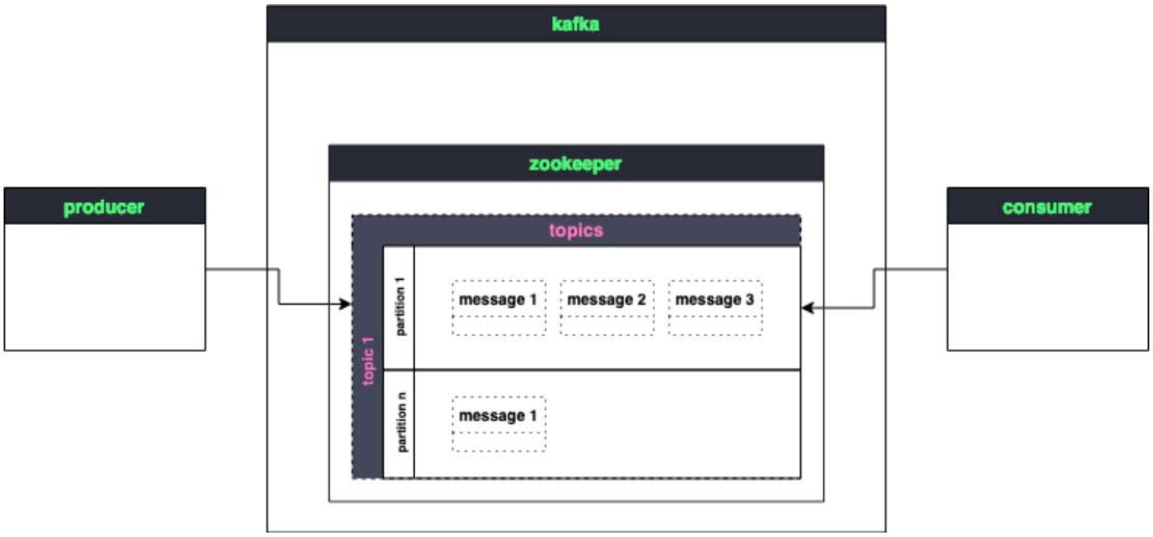
GeoStreaming using Apache Kafka producer and consumer with FastAPI and aiokafka

What is Apache Kafka?

Apache Kafka is a messaging system with a producer producing a message on the one side and a consumer consuming the message on the other side with a broker (zookeeper) in the middle.

When a producer sends a message, the message is pushed into Kafka topics. When a consumer consumes a message it is pulling the message from a Kafka topic. Kafka topics reside within a so-called broker (eg. Zookeeper). Zookeeper provides synchronization within distributed systems and in the case of Apache Kafka keeps track of the status of Kafka cluster nodes and Kafka topics.

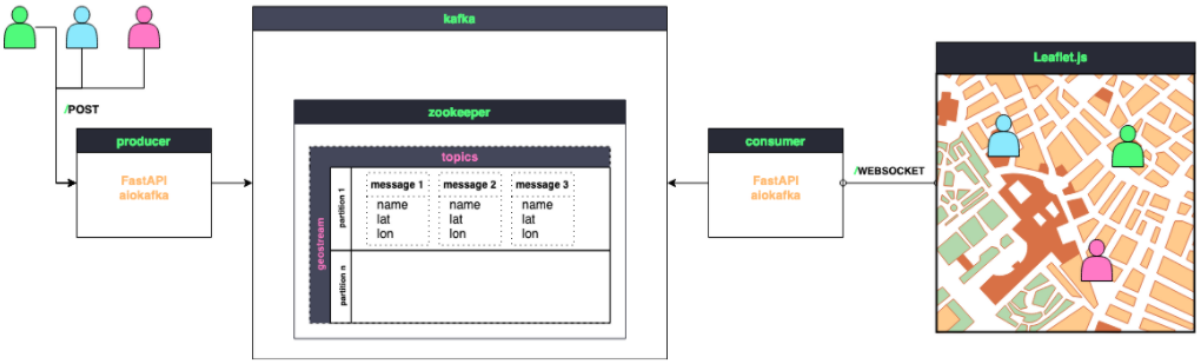
Kafka core principles



- When a producer sends a message, the message is pushed into Kafka topics. When a consumer consumes a message it is pulling the message from a Kafka topic.
- Kafka topics reside within a so-called broker (eg. Zookeeper). Zookeeper provides synchronization within distributed systems and in the case of Apache Kafka keeps track of the status of Kafka cluster nodes and Kafka topics.
- Multiple producers pushing messages into one topic, or you can have them push to different topics
- Messages within topics can be retained indefinitly or be discarded after a certain time, depending on the needs.
- When a consumer starts consuming a message, it starts from the first message that has been pushed to the topic and continues from thereon.
- Kafka stores the so-called *offset*, basically a pointer telling the consumer which messages have been consumed and what is still left to indulge.

- Messages will stay within the topic, yet when the same consumer pulls messages from the topic it will only receive messages from the offset onwards.

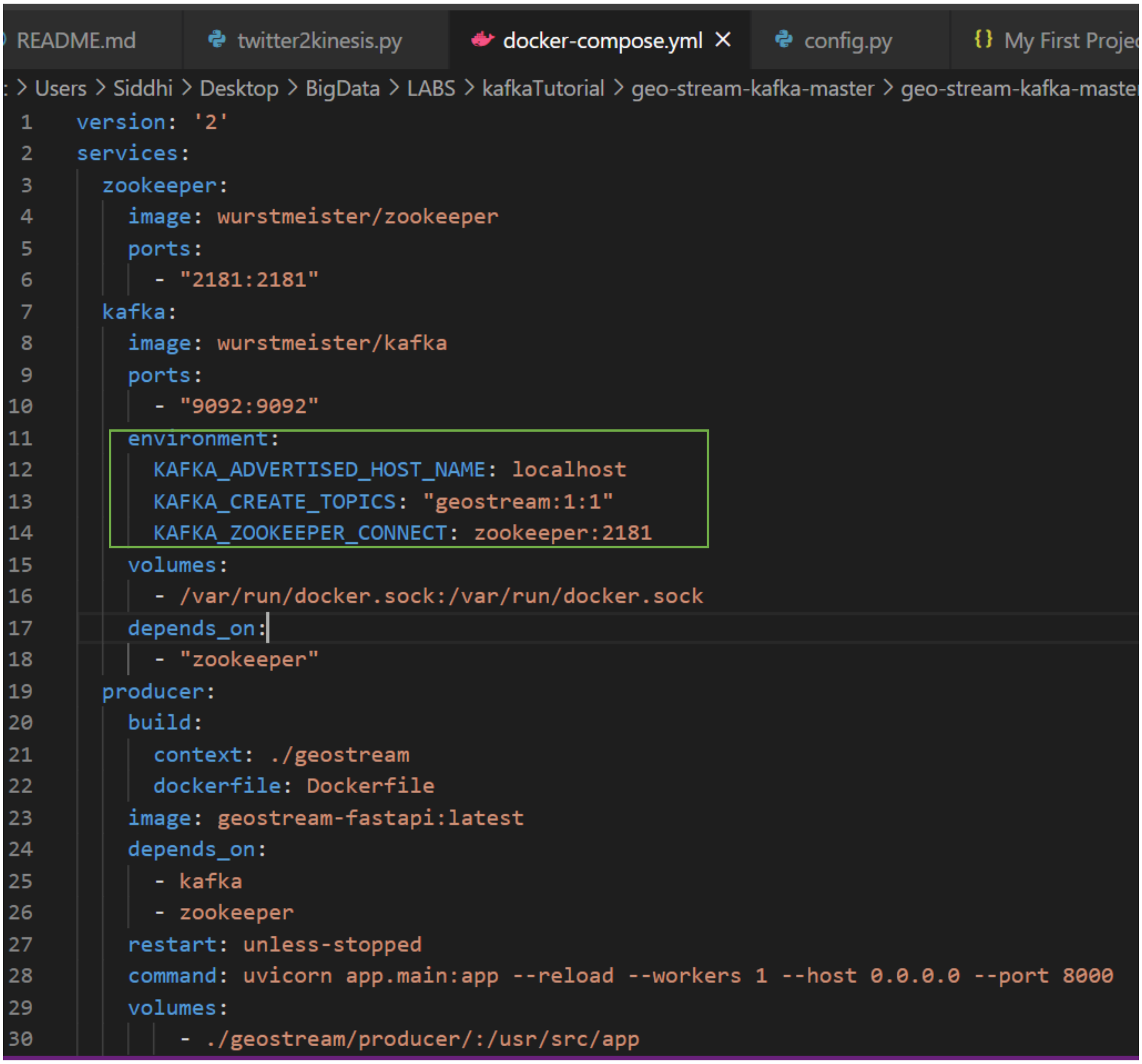
Geo Stream Kafka architecture



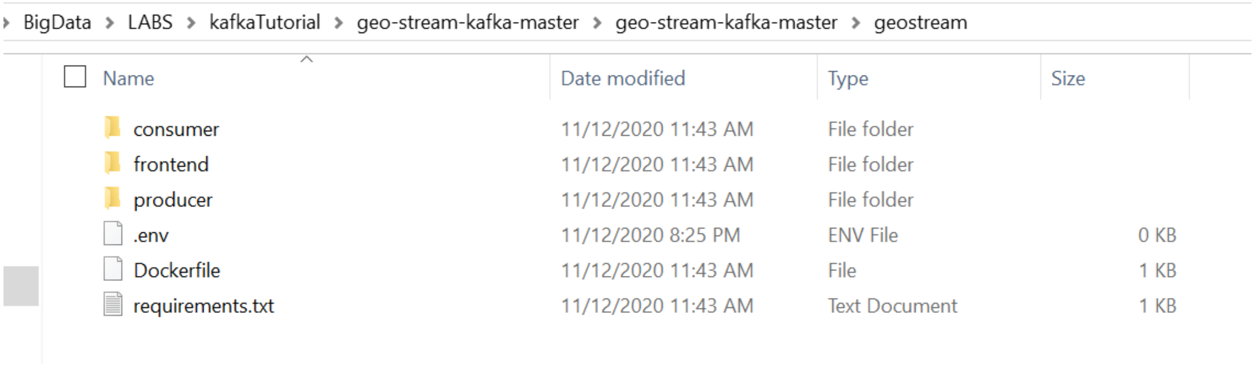
Steps to geostream latitude longitude data using Kafka on static website

Setup an Apache Kafka cluster

Change the KAFKA_ADVERTISED_HOST_NAME in the docker-compose.yaml and create a blank .env file in geostream folder


README.md
twitter2kinesis.py
docker-compose.yml
config.py
My First Proje
Users > Siddhi > Desktop > BigData > LABS > kafkaTutorial > geo-stream-kafka-master > geo-stream-kafka-maste

```
version: '2'
services:
  zookeeper:
    image: wurstmeister/zookeeper
    ports:
      - "2181:2181"
  kafka:
    image: wurstmeister/kafka
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: localhost
      KAFKA_CREATE_TOPICS: "geostream:1:1"
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    depends_on:
      - "zookeeper"
  producer:
    build:
      context: ./geostream
      dockerfile: Dockerfile
    image: geostream-fastapi:latest
    depends_on:
      - kafka
      - zookeeper
    restart: unless-stopped
    command: uvicorn app.main:app --reload --workers 1 --host 0.0.0.0 --port 8000
    volumes:
        - ./geostream/producer/:/usr/src/app
```

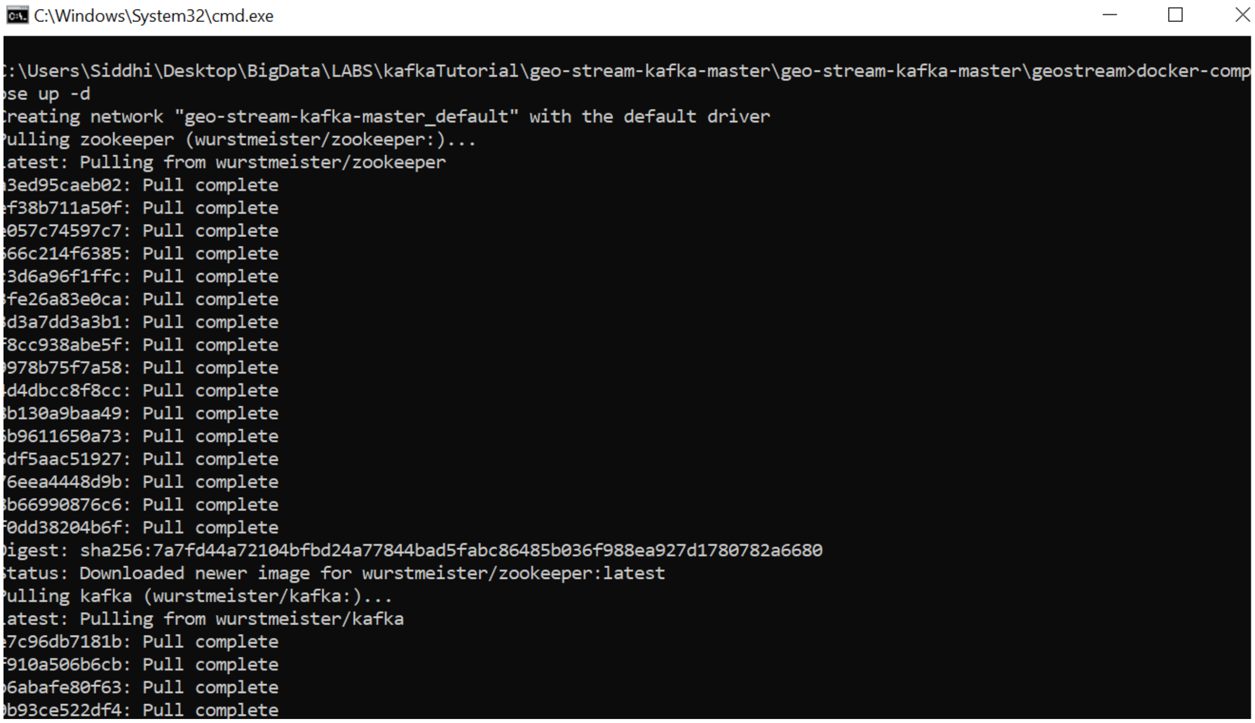


BigData › LABS › kafkaTutorial › geo-stream-kafka-master › geo-stream-kafka-master › geostream

Name	Date modified	Type	Size
consumer	11/12/2020 11:43 AM	File folder	
frontend	11/12/2020 11:43 AM	File folder	
producer	11/12/2020 11:43 AM	File folder	
.env	11/12/2020 8:25 PM	ENV File	0 KB
Dockerfile	11/12/2020 11:43 AM	File	1 KB
requirements.txt	11/12/2020 11:43 AM	Text Document	1 KB

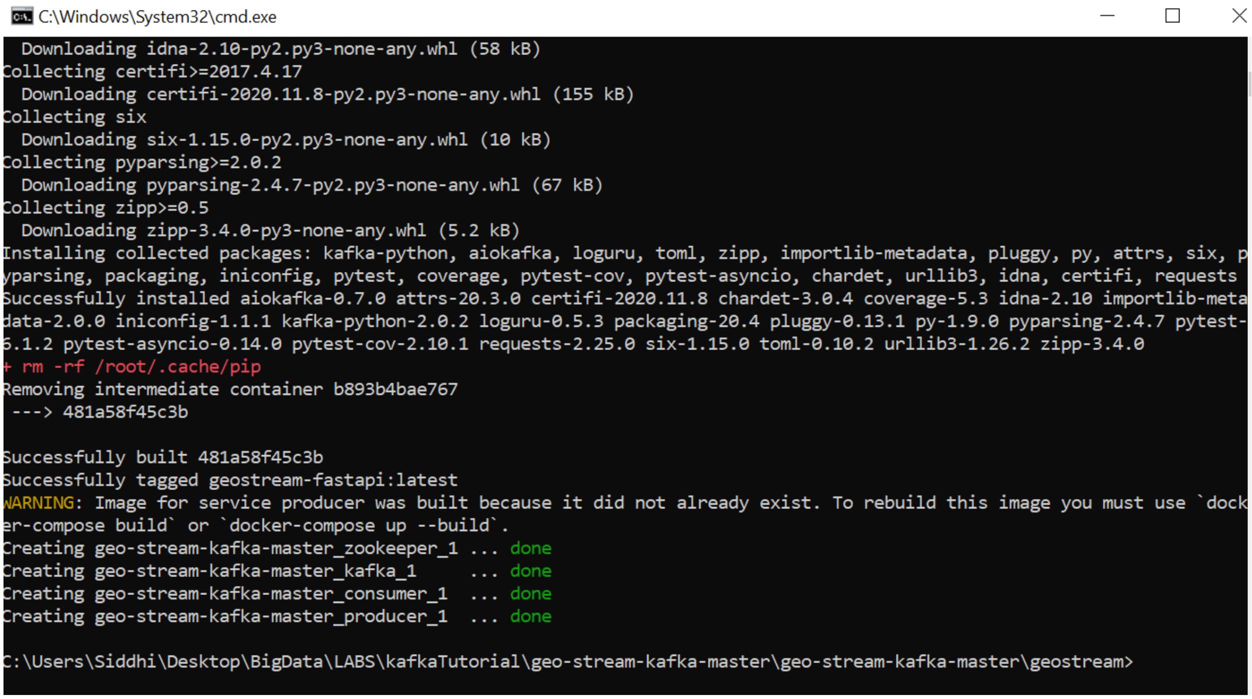
Run docker and use cmd to run the following command

This command would help spin up multiple docker containers for your kafka client, zookeeper, a producer and a consumer



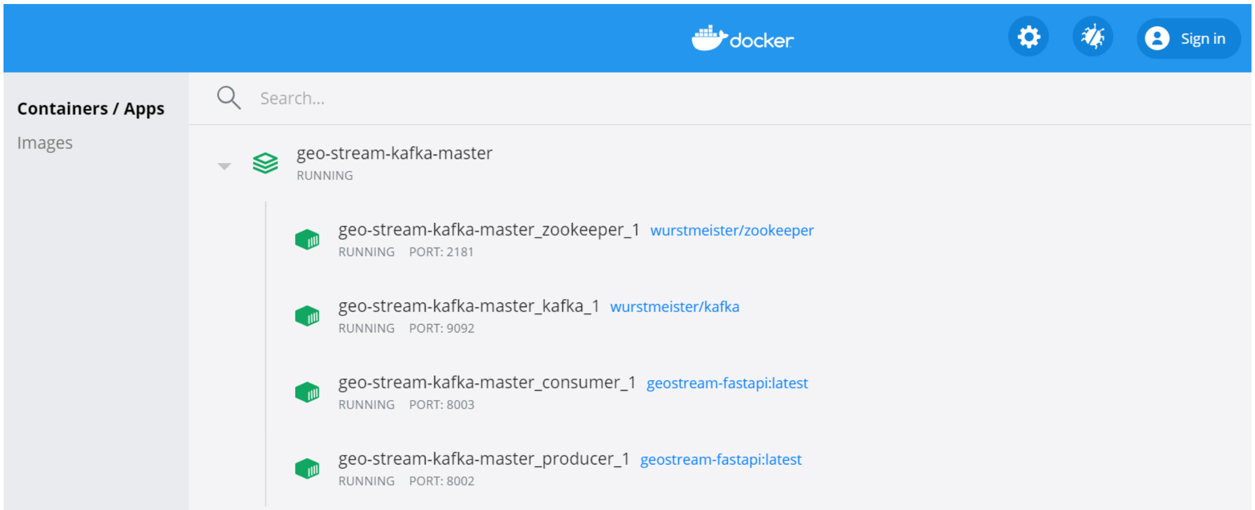
C:\Windows\System32\cmd.exe

```
:\Users\Siddhi\Desktop\BigData\LABS\kafkaTutorial\geo-stream-kafka-master\geo-stream-kafka-master\geostream>docker-comp
se up -d
reating network "geo-stream-kafka-master_default" with the default driver
ulling zookeeper (wurstmeister/zookeeper:)...
atest: Pulling from wurstmeister/zookeeper
3ed95caeb02: Pull complete
f38b711a50f: Pull complete
057c74597c7: Pull complete
66c214f6385: Pull complete
3d6a96f1ffc: Pull complete
fe26a83e0ca: Pull complete
d3a7dd3a3b1: Pull complete
8cc938abe5f: Pull complete
978b75f7a58: Pull complete
d4dbcc8f8cc: Pull complete
b130a9baa49: Pull complete
b9611650a73: Pull complete
df5aac51927: Pull complete
6eea4448d9b: Pull complete
b66990876c6: Pull complete
0dd38204b6f: Pull complete
igest: sha256:7a7fd44a72104bfbd24a77844bad5fabc86485b036f988ea927d1780782a6680
tatus: Downloaded newer image for wurstmeister/zookeeper:latest
ulling kafka (wurstmeister/kafka:)...
atest: Pulling from wurstmeister/kafka
7c96db7181b: Pull complete
910a506b6cb: Pull complete
6abafe80f63: Pull complete
b93ce522df4: Pull complete
```



```
  Downloading idna-2.10-py2.py3-none-any.whl (58 kB)
Collecting certifi>=2017.4.17
  Downloading certifi-2020.11.8-py2.py3-none-any.whl (155 kB)
Collecting six
  Downloading six-1.15.0-py2.py3-none-any.whl (10 kB)
Collecting pyparsing>=2.0.2
  Downloading pyparsing-2.4.7-py2.py3-none-any.whl (67 kB)
Collecting zipp>=0.5
  Downloading zipp-3.4.0-py3-none-any.whl (5.2 kB)
Installing collected packages: kafka-python, aiokafka, loguru, toml, zipp, importlib-metadata, pluggy, py, attrs, six, p
yparsing, packaging, iniconfig, pytest, coverage, pytest-cov, pytest-asyncio, chardet, urllib3, idna, certifi, requests
Successfully installed aiokafka-0.7.0 attrs-20.3.0 certifi-2020.11.8 chardet-3.0.4 coverage-5.3 idna-2.10 importlib-meta
data-2.0.0 iniconfig-1.1.1 kafka-python-2.0.2 loguru-0.5.3 packaging-20.4 pluggy-0.13.1 py-1.9.0 pyparsing-2.4.7 pytest-
6.1.2 pytest-asyncio-0.14.0 pytest-cov-2.10.1 requests-2.25.0 six-1.15.0 toml-0.10.2 urllib3-1.26.2 zipp-3.4.0
+ rm -rf /root/.cache/pip
Removing intermediate container b893b4bae767
 ---> 481a58f45c3b

Successfully built 481a58f45c3b
Successfully tagged geostream-fastapi:latest
WARNING: Image for service producer was built because it did not already exist. To rebuild this image you must use `dock
er-compose build` or `docker-compose up --build`.
Creating geo-stream-kafka-master_zookeeper_1 ... done
Creating geo-stream-kafka-master_kafka_1     ... done
Creating geo-stream-kafka-master_consumer_1  ... done
Creating geo-stream-kafka-master_producer_1  ... done

C:\Users\Siddhi\Desktop\BigData\LABS\kafkaTutorial\geo-stream-kafka-master\geo-stream-kafka-master\geostream>
```

Docker dashboard has the containers running



geo-stream-kafka-master
RUNNING

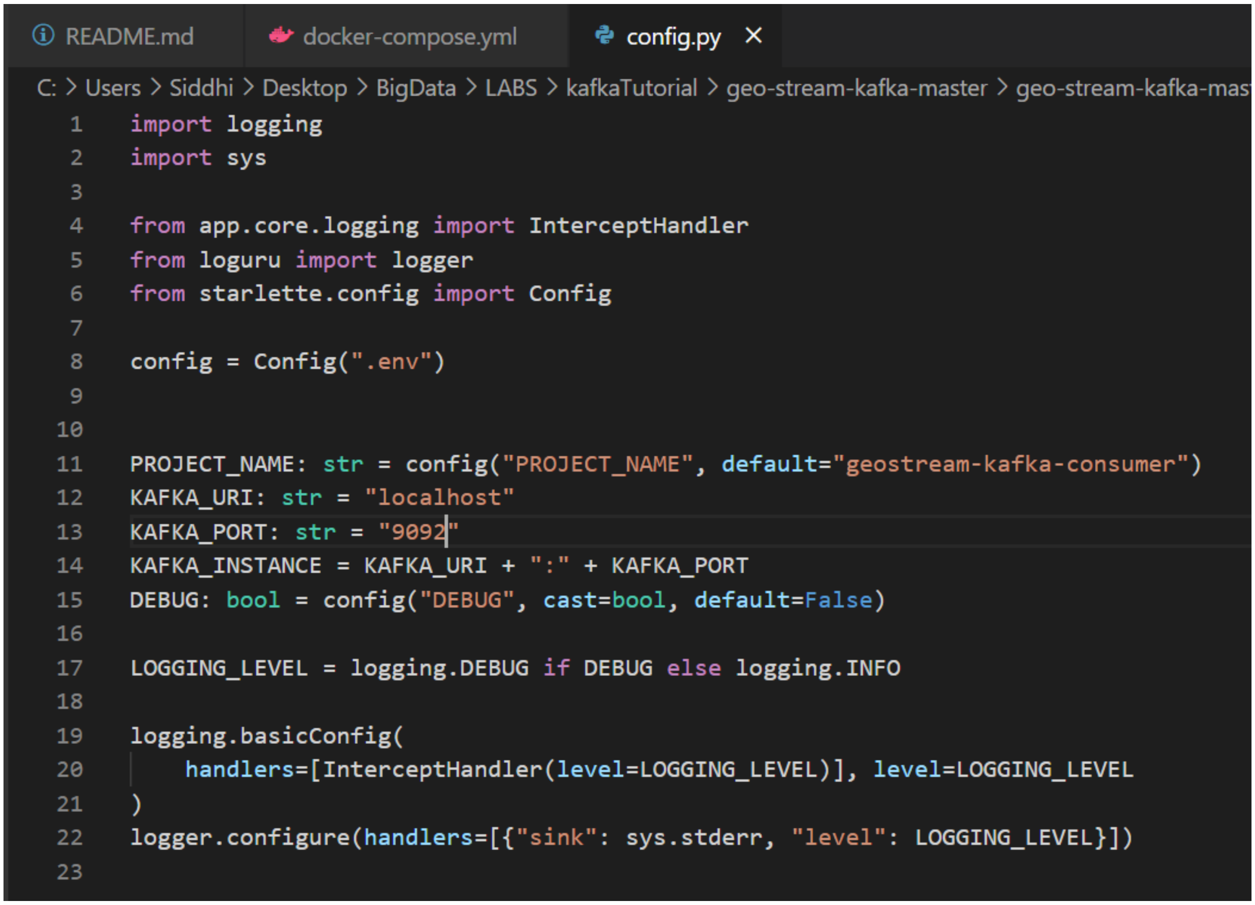
geo-stream-kafka-master_zookeeper_1 wurstmeister/zookeeper
RUNNING PORT: 2181

geo-stream-kafka-master_kafka_1 wurstmeister/kafka
RUNNING PORT: 9092

geo-stream-kafka-master_consumer_1 geostream-fastapi:latest
RUNNING PORT: 8003

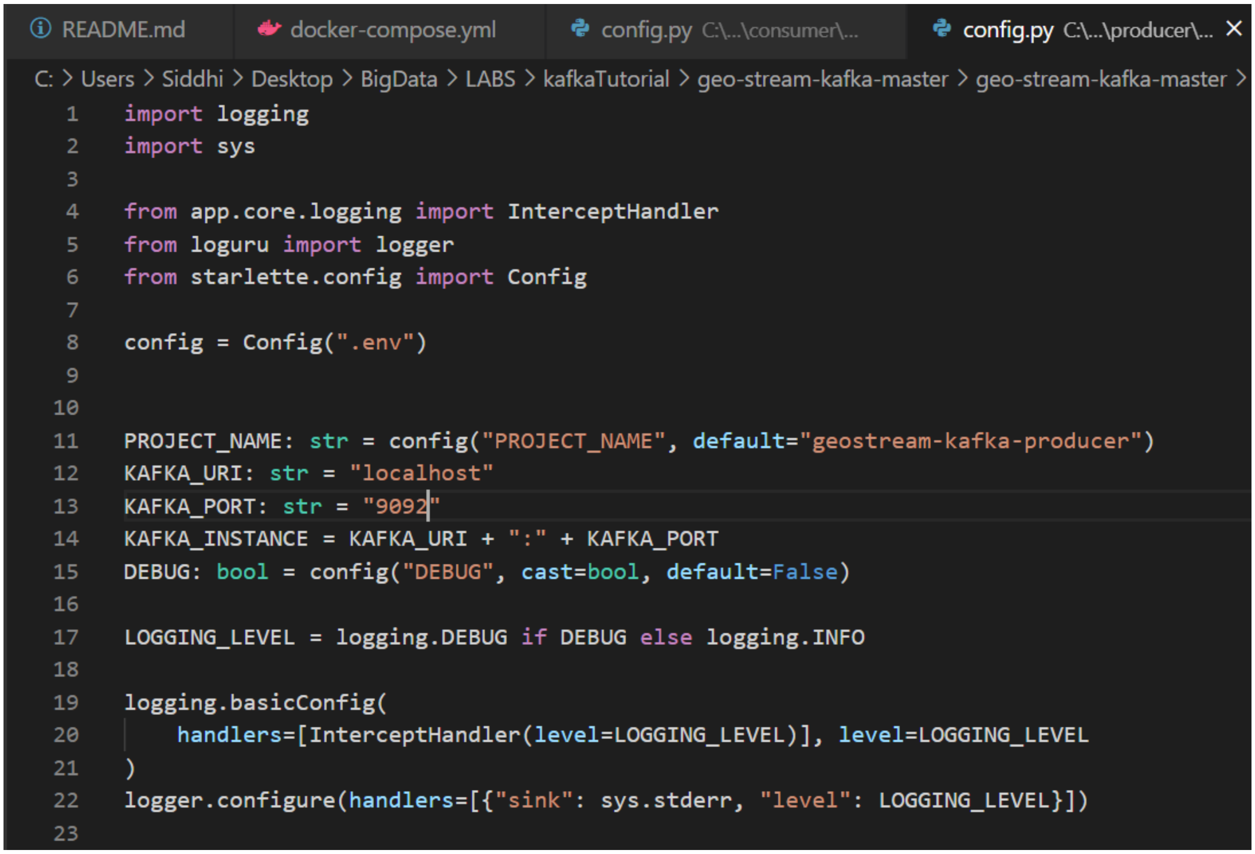
geo-stream-kafka-master_producer_1 geostream-fastapi:latest
RUNNING PORT: 8002

Change the configurations of producer and consumer



README.md
docker-compose.yml
config.py ×
C: > Users > Siddhi > Desktop > BigData > LABS > kafkaTutorial > geo-stream-kafka-master > geo-stream-kafka-mas

```python
import logging
import sys

from app.core.logging import InterceptHandler
from loguru import logger
from starlette.config import Config

config = Config(".env")


PROJECT_NAME: str = config("PROJECT_NAME", default="geostream-kafka-consumer")
KAFKA_URI: str = "localhost"
KAFKA_PORT: str = "9092"
KAFKA_INSTANCE = KAFKA_URI + ":" + KAFKA_PORT
DEBUG: bool = config("DEBUG", cast=bool, default=False)

LOGGING_LEVEL = logging.DEBUG if DEBUG else logging.INFO

logging.basicConfig(
    handlers=[InterceptHandler(level=LOGGING_LEVEL)], level=LOGGING_LEVEL
)
logger.configure(handlers=[{"sink": sys.stderr, "level": LOGGING_LEVEL}])

```


README.md
docker-compose.yml
config.py C:\...\consumer\...
config.py C:\...\producer\...
C: > Users > Siddhi > Desktop > BigData > LABS > kafkaTutorial > geo-stream-kafka-master > geo-stream-kafka-master >

```python
import logging
import sys

from app.core.logging import InterceptHandler
from loguru import logger
from starlette.config import Config

config = Config(".env")


PROJECT_NAME: str = config("PROJECT_NAME", default="geostream-kafka-producer")
KAFKA_URI: str = "localhost"
KAFKA_PORT: str = "9092"
KAFKA_INSTANCE = KAFKA_URI + ":" + KAFKA_PORT
DEBUG: bool = config("DEBUG", cast=bool, default=False)

LOGGING_LEVEL = logging.DEBUG if DEBUG else logging.INFO

logging.basicConfig(
    handlers=[InterceptHandler(level=LOGGING_LEVEL)], level=LOGGING_LEVEL
)
logger.configure(handlers=[{"sink": sys.stderr, "level": LOGGING_LEVEL}])
```

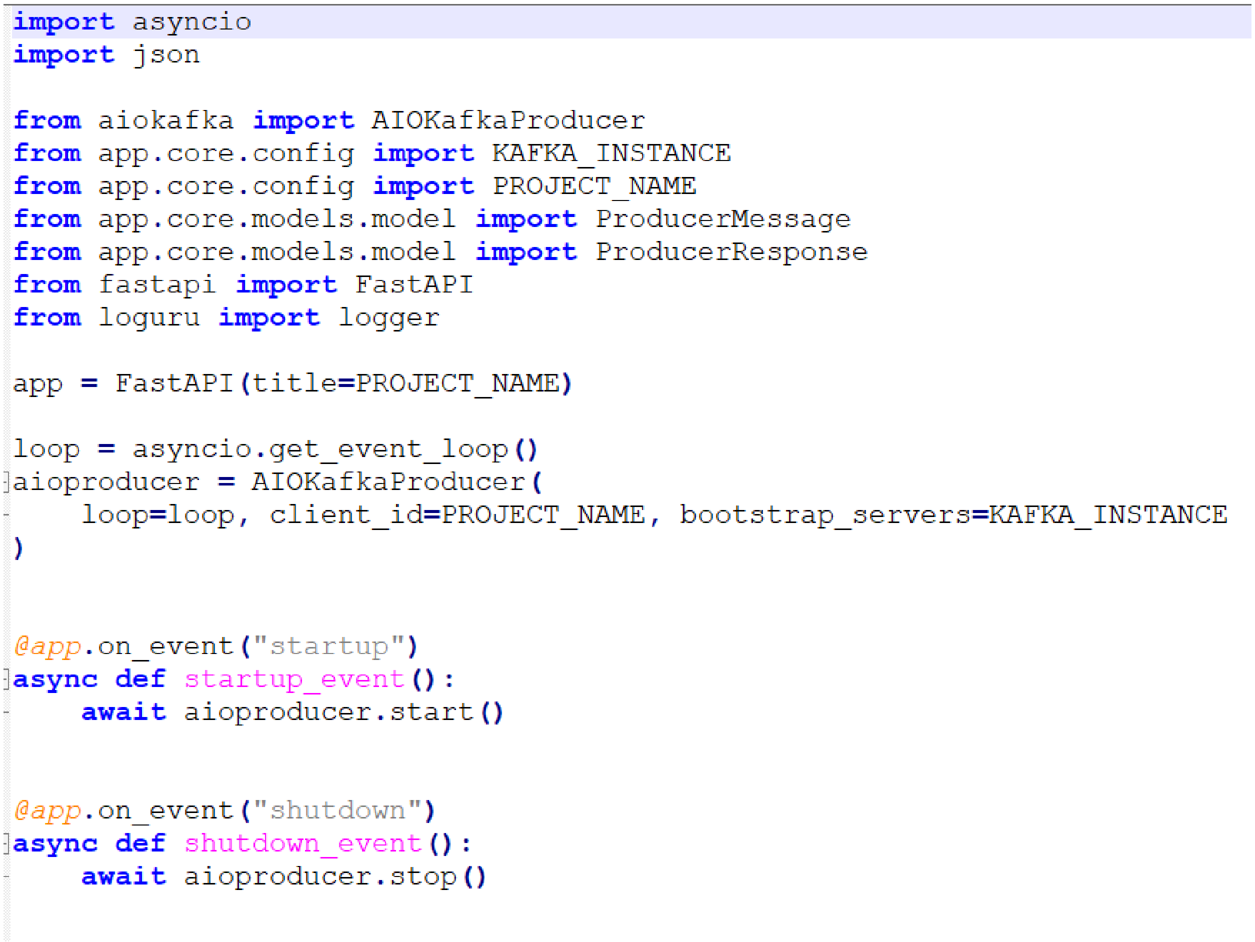
The KAFKA_URI and KAFKA_PORT need to be configured here(the above is for windows) the KAFKA_URI for Mac would be the same as KAFKA_ADVERTISED_HOST_NAME above. The KAFKA_PORT would be 9092 if you haven't made changes to the docker file.

FastAPI Apache Kafka producer

Wrap the producer into a FastAPI endpoint. This allows for more than one entity at a time to produce messages to a topic, but also enables to flexibly change topics that I want to produce messages to with FastAPI endpoint path parameters.

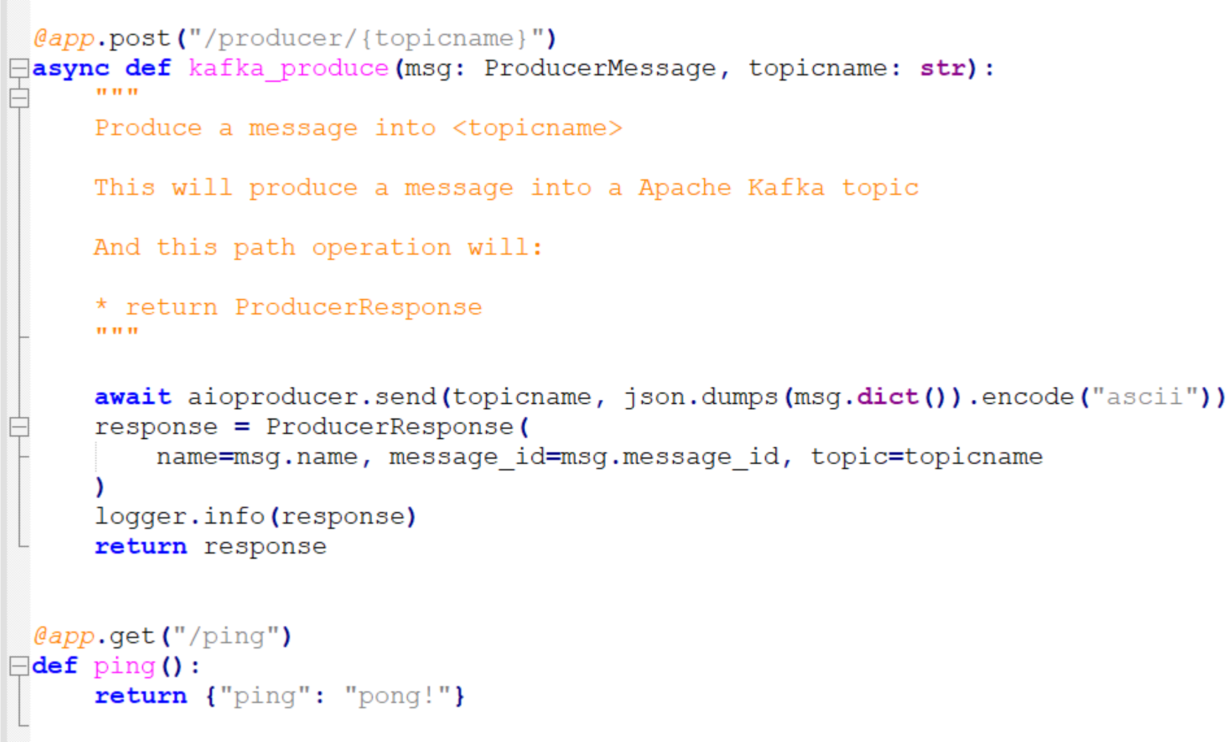
Used aiokafka to make use of FastAPIs async capabilities.

FastAPIs on_event("startup) and on_event("shutdown") make the use of a aiokafka producer easy.



```python
import asyncio
import json

from aiokafka import AIOKafkaProducer
from app.core.config import KAFKA_INSTANCE
from app.core.config import PROJECT_NAME
from app.core.models.model import ProducerMessage
from app.core.models.model import ProducerResponse
from fastapi import FastAPI
from loguru import logger

app = FastAPI(title=PROJECT_NAME)

loop = asyncio.get_event_loop()
aioproducer = AIOKafkaProducer(
    loop=loop, client_id=PROJECT_NAME, bootstrap_servers=KAFKA_INSTANCE
)


@app.on_event("startup")
async def startup_event():
    await aioproducer.start()


@app.on_event("shutdown")
async def shutdown_event():
    await aioproducer.stop()
```

Used `aioproducer` in the application



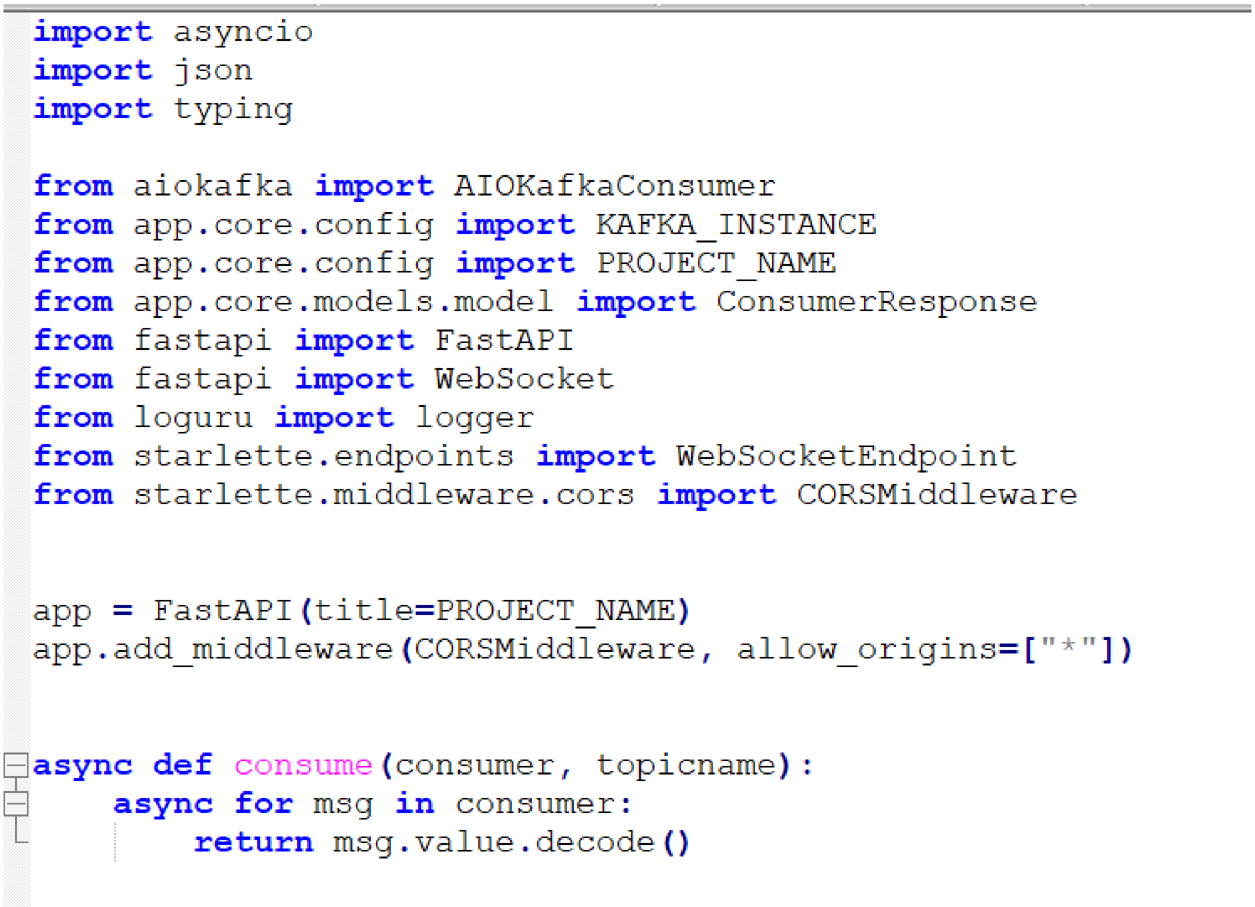
```
@app.post("/producer/{topicname}")
async def kafka_produce(msg: ProducerMessage, topicname: str):
    """
    Produce a message into <topicname>

    This will produce a message into a Apache Kafka topic

    And this path operation will:

    * return ProducerResponse
    """

    await aioproducer.send(topicname, json.dumps(msg.dict()).encode("ascii"))
    response = ProducerResponse(
        name=msg.name, message_id=msg.message_id, topic=topicname
    )
    logger.info(response)
    return response


@app.get("/ping")
def ping():
    return {"ping": "pong!"}
```

FastAPI Apache Kafka consumer

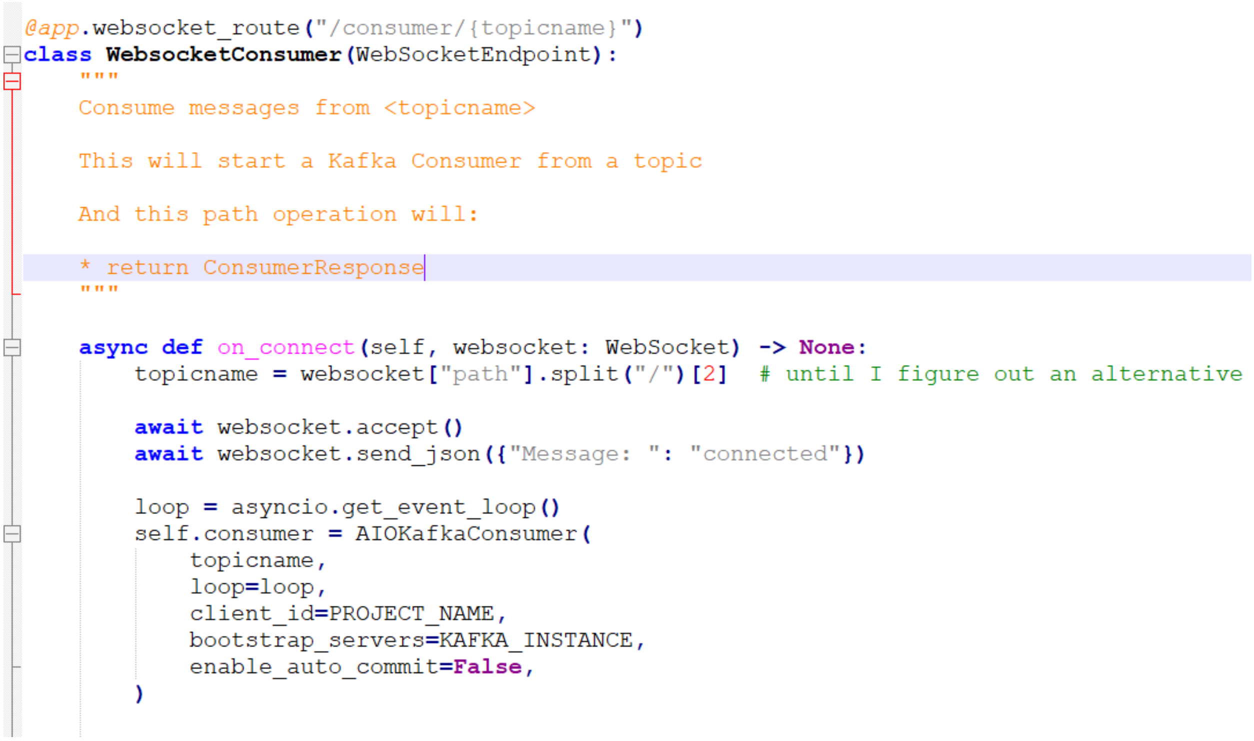
Since FastAPI is built on-top of starlette we can use class-basedd endpoints and especially the WebsocketEndpoint to handle incoming `Websocket` Sessions.

When an application starts a websocket connection with out websocket endpoint we grab the event loop, use that to build and start the aiokafka consumer, start it and start a consumer task in the loop.

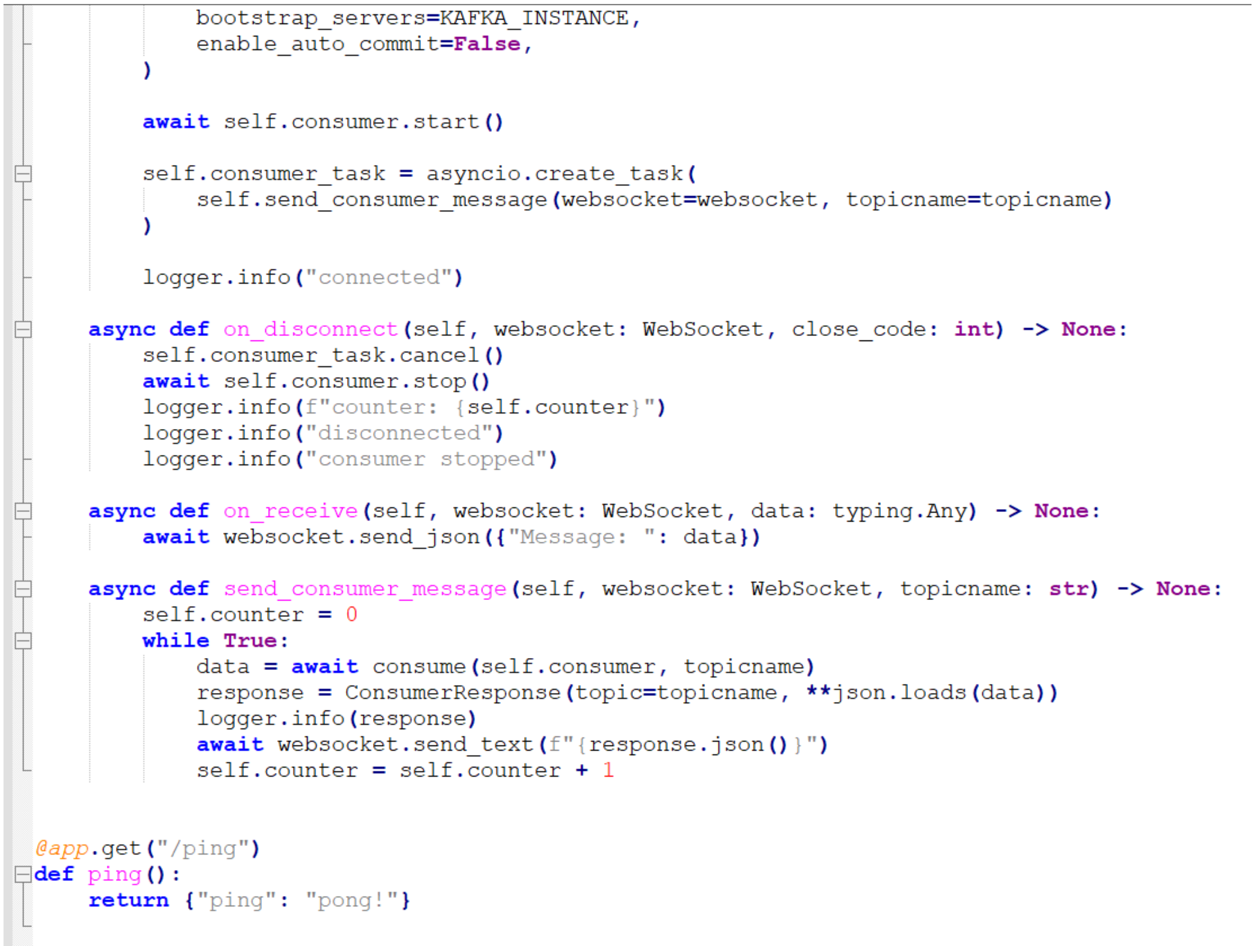
Once this is set, everytime the consumer pulls a new message it is forwarded to the application through the websocket.



```python
import asyncio
import json
import typing

from aiokafka import AIOKafkaConsumer
from app.core.config import KAFKA_INSTANCE
from app.core.config import PROJECT_NAME
from app.core.models.model import ConsumerResponse
from fastapi import FastAPI
from fastapi import WebSocket
from loguru import logger
from starlette.endpoints import WebSocketEndpoint
from starlette.middleware.cors import CORSMiddleware


app = FastAPI(title=PROJECT_NAME)
app.add_middleware(CORSMiddleware, allow_origins=["*"])


async def consume(consumer, topicname):
    async for msg in consumer:
        return msg.value.decode()
```



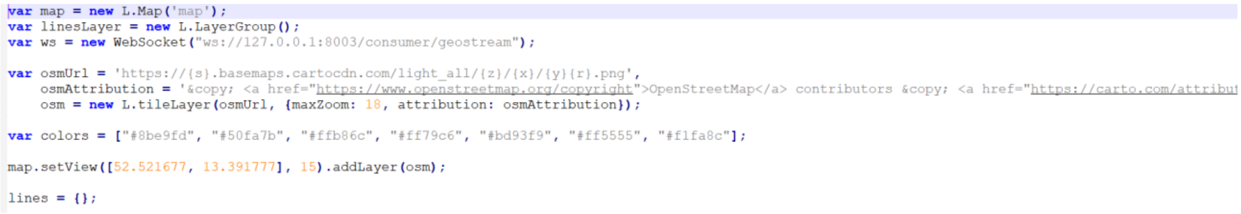
```
@app.websocket_route("/consumer/{topicname}")
class WebsocketConsumer(WebSocketEndpoint):
    """
    Consume messages from <topicname>

    This will start a Kafka Consumer from a topic

    And this path operation will:

    * return ConsumerResponse
    """

    async def on_connect(self, websocket: WebSocket) -> None:
        topicname = websocket["path"].split("/")[2]  # until I figure out an alternative

        await websocket.accept()
        await websocket.send_json({"Message: ": "connected"})

        loop = asyncio.get_event_loop()
        self.consumer = AIOKafkaConsumer(
            topicname,
            loop=loop,
            client_id=PROJECT_NAME,
            bootstrap_servers=KAFKA_INSTANCE,
            enable_auto_commit=False,
        )
```



```
            bootstrap_servers=KAFKA_INSTANCE,
            enable_auto_commit=False,
        )

        await self.consumer.start()

        self.consumer_task = asyncio.create_task(
            self.send_consumer_message(websocket=websocket, topicname=topicname)
        )

        logger.info("connected")

    async def on_disconnect(self, websocket: WebSocket, close_code: int) -> None:
        self.consumer_task.cancel()
        await self.consumer.stop()
        logger.info(f"counter: {self.counter}")
        logger.info("disconnected")
        logger.info("consumer stopped")

    async def on_receive(self, websocket: WebSocket, data: typing.Any) -> None:
        await websocket.send_json({"Message: ": data})

    async def send_consumer_message(self, websocket: WebSocket, topicname: str) -> None:
        self.counter = 0
        while True:
            data = await consume(self.consumer, topicname)
            response = ConsumerResponse(topic=topicname, **json.loads(data))
            logger.info(response)
            await websocket.send_text(f"{response.json()}")
            self.counter = self.counter + 1


@app.get("/ping")
def ping():
    return {"ping": "pong!"}
```

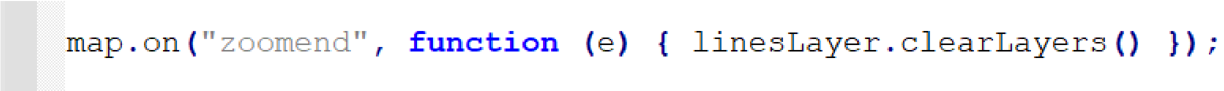
Leaflet application

Declare our map and websocket connection to the `/consumer/{topicname}` endpoint.



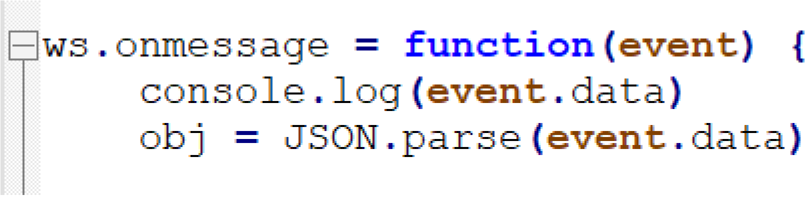
```
var map = new L.Map('map');
var linesLayer = new L.LayerGroup();
var ws = new WebSocket("ws://127.0.0.1:8003/consumer/geostream");

var osmUrl = 'https://{s}.basemaps.cartocdn.com/light_all/{z}/{x}/{y}{r}.png',
    osmAttribution = '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors &copy; <a href="https://carto.com/attribut
    osm = new L.tileLayer(osmUrl, {maxZoom: 18, attribution: osmAttribution});

var colors = ["#8be9fd", "#50fa7b", "#ffb86c", "#ff79c6", "#bd93f9", "#ff5555", "#f1fa8c"];

map.setView([52.521677, 13.391777], 15).addLayer(osm);

lines = {};
```

When you zoom on the map the `bounds` will be messed up and the events will not properly draw polylines along the trajectory of the entity. To fix this I added an `zoomend` trigger:



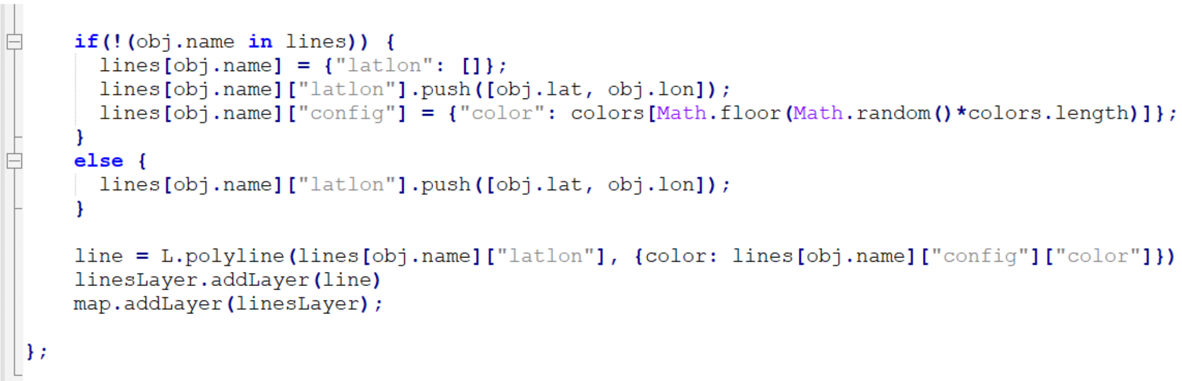
```
map.on("zoomend", function (e) { linesLayer.clearLayers() });
```

When client and backend established the silent agreement to use WebSockets, we can declare what we want to do, whenever the websocket receives a new message. Every message is an `event`. And every `event` consists of metadata and `event.data` that can be parsed with `JSON.parse()`.



```
ws.onmessage = function(event) {
    console.log(event.data)
    obj = JSON.parse(event.data)
```

One of the requirements was to display more than one entity that pushes messages through the producer, Kafka and the consumer on the map as a live-event. For the leaflet application to associate an event to an entity, I hash events by the name of the entity that is sending them. If there is a new `name` in an event, it'll be hashed into a dictionary and added as a new layer on the map. As I wanted every entity to be represented with a different color, the color will be randomly grabbed from the list of `colors` and hashed alongside the position of the event/entity.



```
    if(!(obj.name in lines)) {
      lines[obj.name] = {"latlon": []};
      lines[obj.name]["latlon"].push([obj.lat, obj.lon]);
      lines[obj.name]["config"] = {"color": colors[Math.floor(Math.random()*colors.length)]};
    }
    else {
      lines[obj.name]["latlon"].push([obj.lat, obj.lon]);
    }

    line = L.polyline(lines[obj.name]["latlon"], {color: lines[obj.name]["config"]["color"]})
    linesLayer.addLayer(line)
    map.addLayer(linesLayer);

};
```

When the leafletjs application either specifically closes the websocket or the browser is closed, we close the websocket, cancel the consumer_task and stop the consumer.

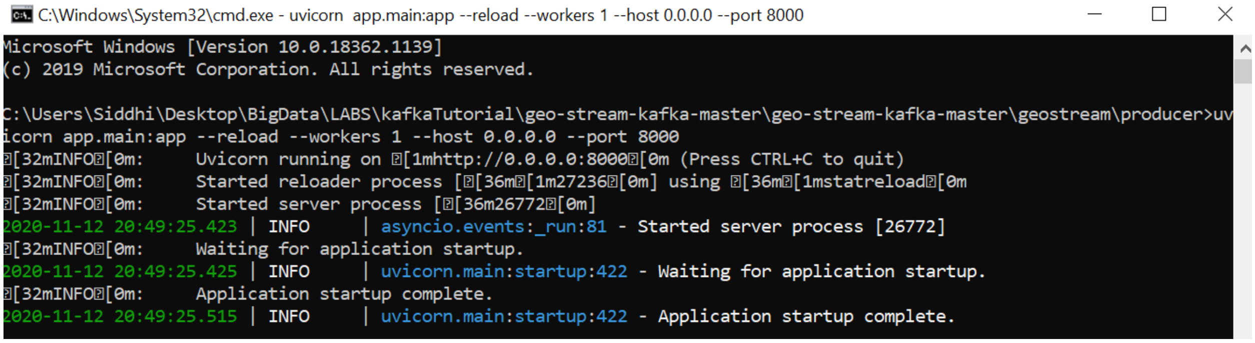
- Open to command prompts and change directory to respectively :

1. geo-stream-kafka-master\geostream\producer
2. geo-stream-kafka-master\geostream\consumer

We will use the command prompts to run the producer and consumer simultaneously,

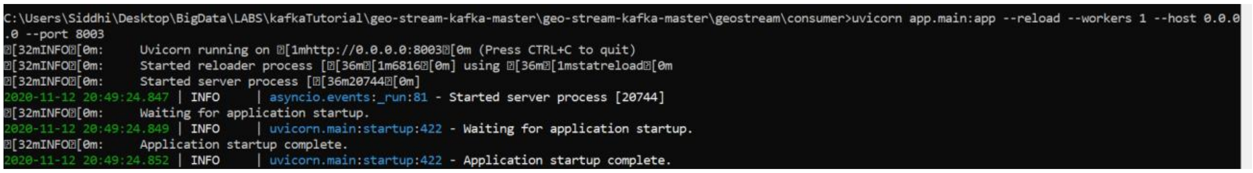
The producer should be sending messages for the consumer to receive messages hence both should be active when trying to test

Producer



```
C:\Windows\System32\cmd.exe - uvicorn  app.main:app --reload --workers 1 --host 0.0.0.0 --port 8000
Microsoft Windows [Version 10.0.18362.1139]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Siddhi\Desktop\BigData\LABS\kafkaTutorial\geo-stream-kafka-master\geo-stream-kafka-master\geostream\producer>uv
icorn app.main:app --reload --workers 1 --host 0.0.0.0 --port 8000
[32mINFO[0m:     Uvicorn running on [1mhttp://0.0.0.0:8000[0m (Press CTRL+C to quit)
[32mINFO[0m:     Started reloader process [[36m[1m27236[0m] using [36m[1mstatreload[0m
[32mINFO[0m:     Started server process [[36m26772[0m]
2020-11-12 20:49:25.423 | INFO     | asyncio.events:_run:81 - Started server process [26772]
[32mINFO[0m:     Waiting for application startup.
2020-11-12 20:49:25.425 | INFO     | uvicorn.main:startup:422 - Waiting for application startup.
[32mINFO[0m:     Application startup complete.
2020-11-12 20:49:25.515 | INFO     | uvicorn.main:startup:422 - Application startup complete.
```

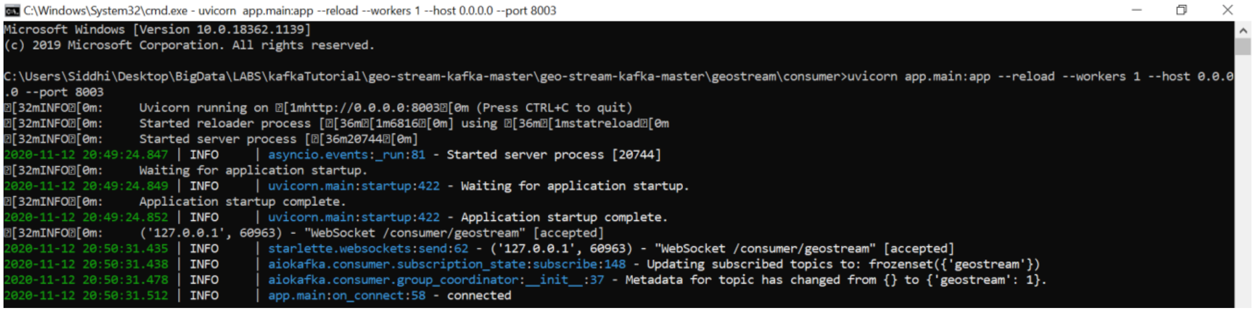
Consumer



```
C:\Users\Siddhi\Desktop\BigData\LABS\kafkaTutorial\geo-stream-kafka-master\geo-stream-kafka-master\geostream\consumer>uvicorn app.main:app --reload --workers 1 --host 0.0.0
.0 --port 8003
[32mINFO[0m:     Uvicorn running on [1mhttp://0.0.0.0:8003[0m (Press CTRL+C to quit)
[32mINFO[0m:     Started reloader process [[36m[1m6816[0m] using [36m[1mstatreload[0m
[32mINFO[0m:     Started server process [[36m20744[0m]
2020-11-12 20:49:24.847 | INFO     | asyncio.events:_run:81 - Started server process [20744]
[32mINFO[0m:     Waiting for application startup.
2020-11-12 20:49:24.849 | INFO     | uvicorn.main:startup:422 - Waiting for application startup.
[32mINFO[0m:     Application startup complete.
2020-11-12 20:49:24.852 | INFO     | uvicorn.main:startup:422 - Application startup complete.
```

- For the consumer to be connected and ready to receive messages we need to access the static webpage that is configured as the consumer for the incoming websocket messages

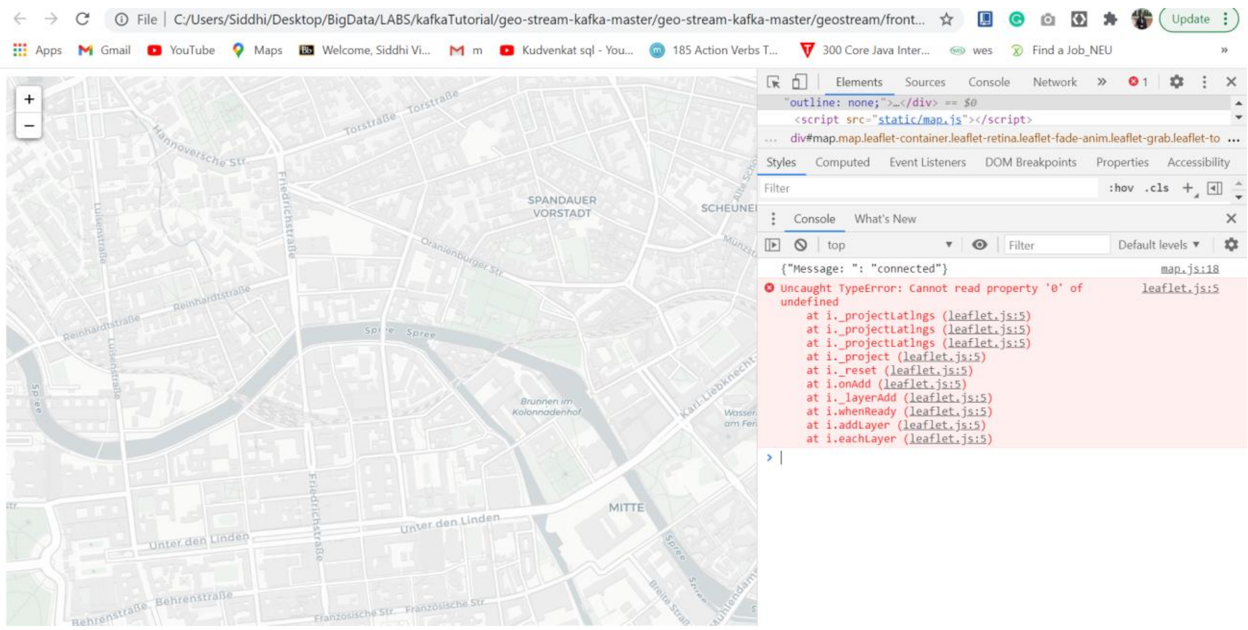
After opening index.html page



```
C:\Windows\System32\cmd.exe - uvicorn  app.main:app --reload --workers 1 --host 0.0.0.0 --port 8003
Microsoft Windows [Version 10.0.18362.1139]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Siddhi\Desktop\BigData\LABS\kafkaTutorial\geo-stream-kafka-master\geo-stream-kafka-master\geostream\consumer>uvicorn app.main:app --reload --workers 1 --host 0.0.0
.0 --port 8003
[32mINFO[0m:     Uvicorn running on [1mhttp://0.0.0.0:8003[0m (Press CTRL+C to quit)
[32mINFO[0m:     Started reloader process [[36m[1m6816[0m] using [36m[1mstatreload[0m
[32mINFO[0m:     Started server process [[36m20744[0m]
2020-11-12 20:49:24.847 | INFO     | asyncio.events:_run:81 - Started server process [20744]
[32mINFO[0m:     Waiting for application startup.
2020-11-12 20:49:24.849 | INFO     | uvicorn.main:startup:422 - Waiting for application startup.
[32mINFO[0m:     Application startup complete.
2020-11-12 20:49:24.852 | INFO     | uvicorn.main:startup:422 - Application startup complete.
[32mINFO[0m:     ('127.0.0.1', 60963) - "WebSocket /consumer/geostream" [accepted]
2020-11-12 20:50:31.435 | INFO     | starlette.websockets:send:62 - ('127.0.0.1', 60963) - "WebSocket /consumer/geostream" [accepted]
2020-11-12 20:50:31.438 | INFO     | aiokafka.consumer.subscription_state:subscribe:148 - Updating subscribed topics to: frozenset({'geostream'})
2020-11-12 20:50:31.478 | INFO     | aiokafka.consumer.group_coordinator:__init__:37 - Metadata for topic has changed from {} to {'geostream': 1}.
2020-11-12 20:50:31.512 | INFO     | app.main:on_connect:58 - connected
```

Successfully connected static webpage

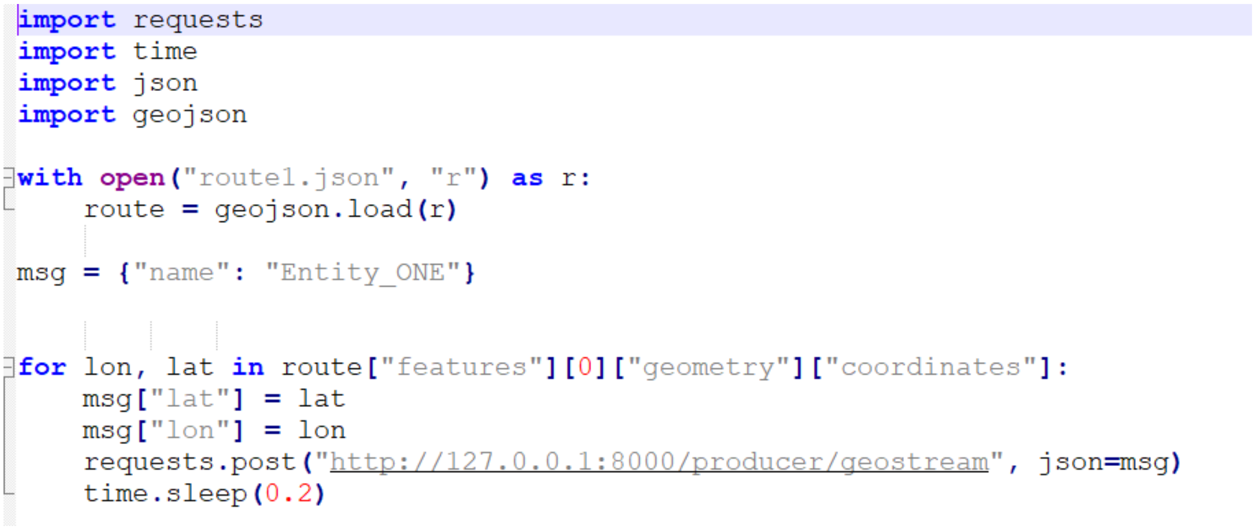
The red errors are simply an indicator that there are no incoming messages to the consumer as of now.



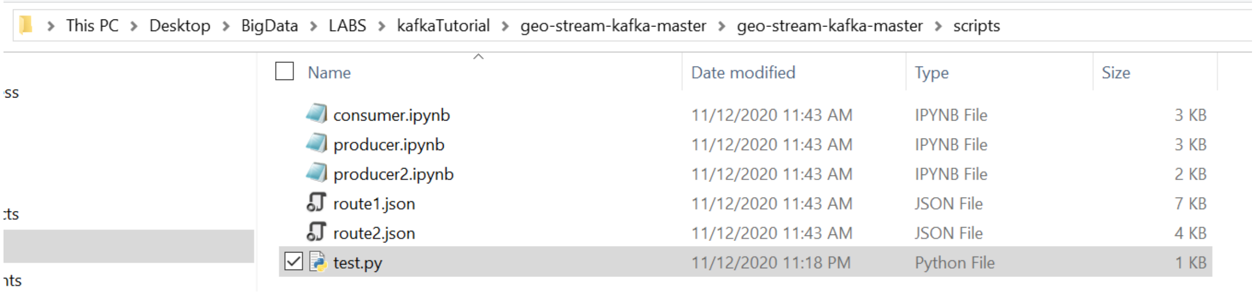
```
"outline: none;">…</div> == $0
  <script src="static/map.js"></script>
... div#map.map.leaflet-container.leaflet-retina.leaflet-fade-anim.leaflet-grab.leaflet-to ...
```

```
{"Message: ": "connected"}                                    map.js:18
Uncaught TypeError: Cannot read property '0' of               leaflet.js:5
undefined
    at i._projectLatlngs (leaflet.js:5)
    at i._projectLatlngs (leaflet.js:5)
    at i._projectLatlngs (leaflet.js:5)
    at i._project (leaflet.js:5)
    at i._reset (leaflet.js:5)
    at i.onAdd (leaflet.js:5)
    at i._layerAdd (leaflet.js:5)
    at i.whenReady (leaflet.js:5)
    at i.addLayer (leaflet.js:5)
    at i.eachLayer (leaflet.js:5)
>
```

- To stream data from producer to consumer configure a python file test.py as below in folder: geo-stream-kafka-master\scripts



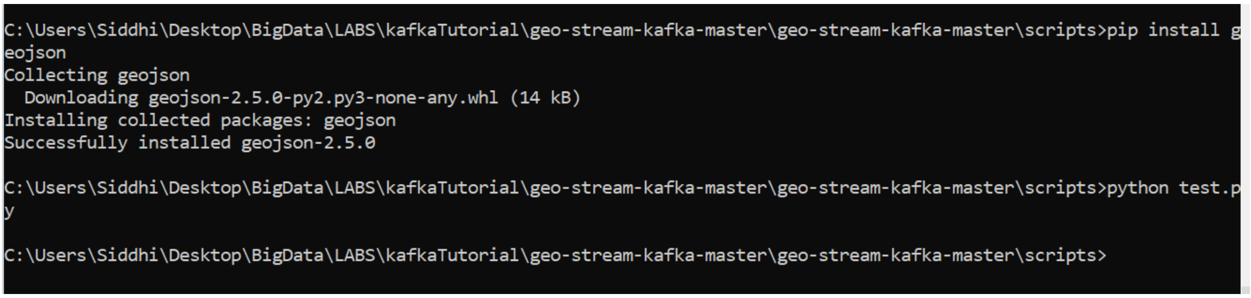
```python
import requests
import time
import json
import geojson

with open("route1.json", "r") as r:
    route = geojson.load(r)

msg = {"name": "Entity_ONE"}


for lon, lat in route["features"][0]["geometry"]["coordinates"]:
    msg["lat"] = lat
    msg["lon"] = lon
    requests.post("http://127.0.0.1:8000/producer/geostream", json=msg)
    time.sleep(0.2)
```



This PC > Desktop > BigData > LABS > kafkaTutorial > geo-stream-kafka-master > geo-stream-kafka-master > scripts

□ Name	Date modified	Type	Size
consumer.ipynb	11/12/2020 11:43 AM	IPYNB File	3 KB
producer.ipynb	11/12/2020 11:43 AM	IPYNB File	3 KB
producer2.ipynb	11/12/2020 11:43 AM	IPYNB File	2 KB
route1.json	11/12/2020 11:43 AM	JSON File	7 KB
route2.json	11/12/2020 11:43 AM	JSON File	4 KB
☑ test.py	11/12/2020 11:18 PM	Python File	1 KB

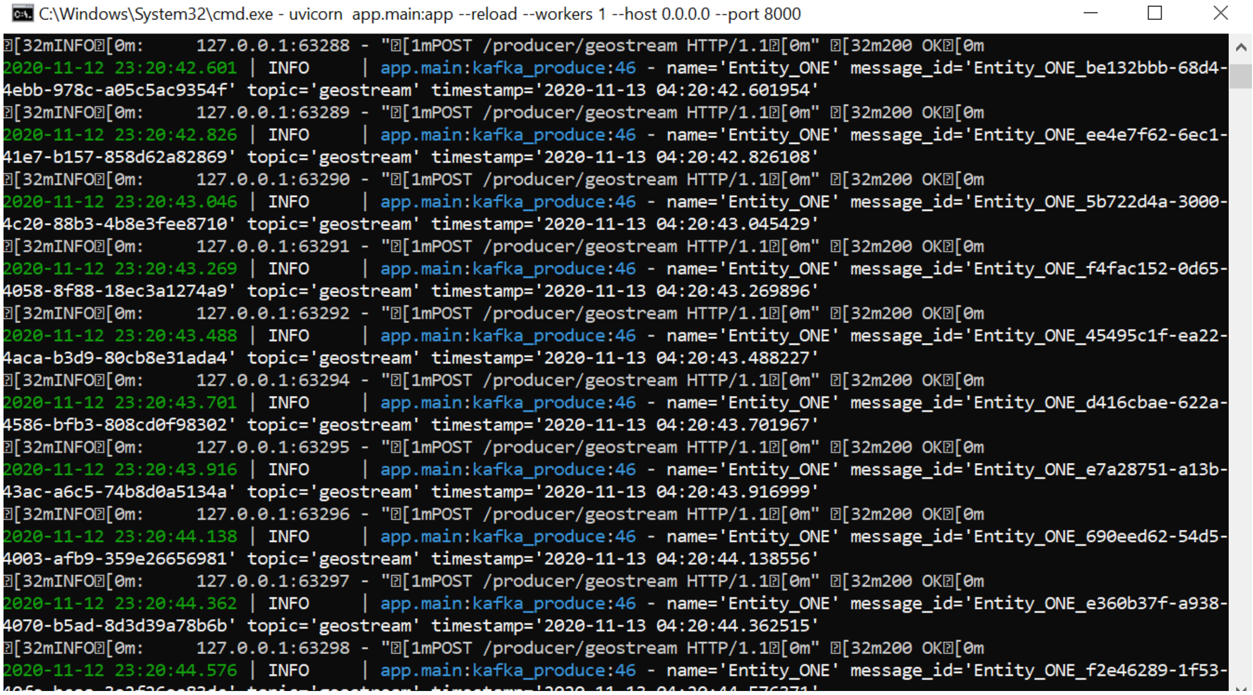
Install geojson package



```
C:\Users\Siddhi\Desktop\BigData\LABS\kafkaTutorial\geo-stream-kafka-master\geo-stream-kafka-master\scripts>pip install g
eojson
Collecting geojson
  Downloading geojson-2.5.0-py2.py3-none-any.whl (14 kB)
Installing collected packages: geojson
Successfully installed geojson-2.5.0

C:\Users\Siddhi\Desktop\BigData\LABS\kafkaTutorial\geo-stream-kafka-master\geo-stream-kafka-master\scripts>python test.p
y

C:\Users\Siddhi\Desktop\BigData\LABS\kafkaTutorial\geo-stream-kafka-master\geo-stream-kafka-master\scripts>
```

Run the python file:-

Producer producing messages after executing python script on producer’s port

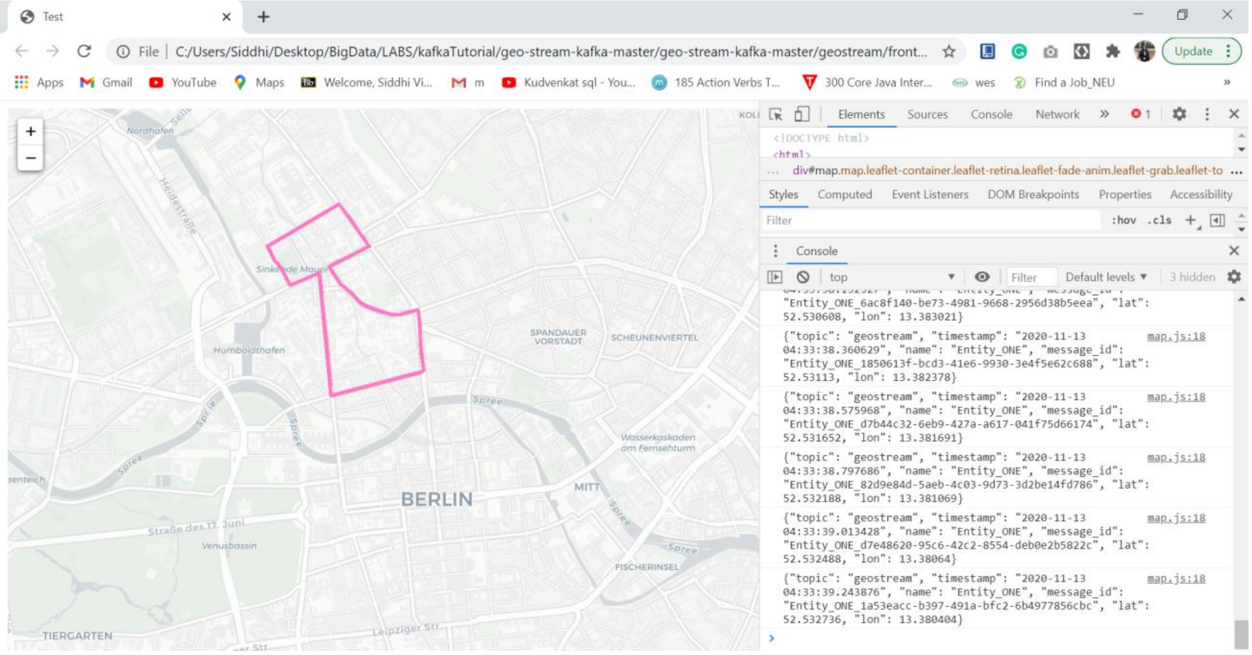


C:\Windows\System32\cmd.exe - uvicorn app.main:app --reload --workers 1 --host 0.0.0.0 --port 8000

```
⍰[32mINFO⍰[0m:     127.0.0.1:63288 - "⍰[1mPOST /producer/geostream HTTP/1.1⍰[0m" ⍰[32m200 OK⍰[0m
2020-11-12 23:20:42.601 | INFO     | app.main:kafka_produce:46 - name='Entity_ONE' message_id='Entity_ONE_be132bbb-68d4-
4ebb-978c-a05c5ac9354f' topic='geostream' timestamp='2020-11-13 04:20:42.601954'
⍰[32mINFO⍰[0m:     127.0.0.1:63289 - "⍰[1mPOST /producer/geostream HTTP/1.1⍰[0m" ⍰[32m200 OK⍰[0m
2020-11-12 23:20:42.826 | INFO     | app.main:kafka_produce:46 - name='Entity_ONE' message_id='Entity_ONE_ee4e7f62-6ec1-
41e7-b157-858d62a82869' topic='geostream' timestamp='2020-11-13 04:20:42.826108'
⍰[32mINFO⍰[0m:     127.0.0.1:63290 - "⍰[1mPOST /producer/geostream HTTP/1.1⍰[0m" ⍰[32m200 OK⍰[0m
2020-11-12 23:20:43.046 | INFO     | app.main:kafka_produce:46 - name='Entity_ONE' message_id='Entity_ONE_5b722d4a-3000-
4c20-88b3-4b8e3fee8710' topic='geostream' timestamp='2020-11-13 04:20:43.045429'
⍰[32mINFO⍰[0m:     127.0.0.1:63291 - "⍰[1mPOST /producer/geostream HTTP/1.1⍰[0m" ⍰[32m200 OK⍰[0m
2020-11-12 23:20:43.269 | INFO     | app.main:kafka_produce:46 - name='Entity_ONE' message_id='Entity_ONE_f4fac152-0d65-
4058-8f88-18ec3a1274a9' topic='geostream' timestamp='2020-11-13 04:20:43.269896'
⍰[32mINFO⍰[0m:     127.0.0.1:63292 - "⍰[1mPOST /producer/geostream HTTP/1.1⍰[0m" ⍰[32m200 OK⍰[0m
2020-11-12 23:20:43.488 | INFO     | app.main:kafka_produce:46 - name='Entity_ONE' message_id='Entity_ONE_45495c1f-ea22-
4aca-b3d9-80cb8e31ada4' topic='geostream' timestamp='2020-11-13 04:20:43.488227'
⍰[32mINFO⍰[0m:     127.0.0.1:63294 - "⍰[1mPOST /producer/geostream HTTP/1.1⍰[0m" ⍰[32m200 OK⍰[0m
2020-11-12 23:20:43.701 | INFO     | app.main:kafka_produce:46 - name='Entity_ONE' message_id='Entity_ONE_d416cbae-622a-
4586-bfb3-808cd0f98302' topic='geostream' timestamp='2020-11-13 04:20:43.701967'
⍰[32mINFO⍰[0m:     127.0.0.1:63295 - "⍰[1mPOST /producer/geostream HTTP/1.1⍰[0m" ⍰[32m200 OK⍰[0m
2020-11-12 23:20:43.916 | INFO     | app.main:kafka_produce:46 - name='Entity_ONE' message_id='Entity_ONE_e7a28751-a13b-
43ac-a6c5-74b8d0a5134a' topic='geostream' timestamp='2020-11-13 04:20:43.916999'
⍰[32mINFO⍰[0m:     127.0.0.1:63296 - "⍰[1mPOST /producer/geostream HTTP/1.1⍰[0m" ⍰[32m200 OK⍰[0m
2020-11-12 23:20:44.138 | INFO     | app.main:kafka_produce:46 - name='Entity_ONE' message_id='Entity_ONE_690eed62-54d5-
4003-afb9-359e26656981' topic='geostream' timestamp='2020-11-13 04:20:44.138556'
⍰[32mINFO⍰[0m:     127.0.0.1:63297 - "⍰[1mPOST /producer/geostream HTTP/1.1⍰[0m" ⍰[32m200 OK⍰[0m
2020-11-12 23:20:44.362 | INFO     | app.main:kafka_produce:46 - name='Entity_ONE' message_id='Entity_ONE_e360b37f-a938-
4070-b5ad-8d3d39a78b6b' topic='geostream' timestamp='2020-11-13 04:20:44.362515'
⍰[32mINFO⍰[0m:     127.0.0.1:63298 - "⍰[1mPOST /producer/geostream HTTP/1.1⍰[0m" ⍰[32m200 OK⍰[0m
2020-11-12 23:20:44.576 | INFO     | app.main:kafka_produce:46 - name='Entity_ONE' message_id='Entity_ONE_f2e46289-1f53-
```

Consumer is consuming messages

We can see the path being traced in real-time when the script is ran.



```
"Entity_ONE_6ac8f140-be73-4981-9668-2956d38b5eea", "lat":
52.530608, "lon": 13.383021}
{"topic": "geostream", "timestamp": "2020-11-13         map.js:18
04:33:38.360629", "name": "Entity_ONE", "message_id":
"Entity_ONE_1850613f-bcd3-41e6-9930-3e4f5e62c688", "lat":
52.53113, "lon": 13.382378}
{"topic": "geostream", "timestamp": "2020-11-13         map.js:18
04:33:38.575968", "name": "Entity_ONE", "message_id":
"Entity_ONE_d7b44c32-6eb9-427a-a617-041f75d66174", "lat":
52.531652, "lon": 13.381691}
{"topic": "geostream", "timestamp": "2020-11-13         map.js:18
04:33:38.797686", "name": "Entity_ONE", "message_id":
"Entity_ONE_82d9e84d-5aeb-4c03-9d73-3d2be14fd786", "lat":
52.532188, "lon": 13.381069}
{"topic": "geostream", "timestamp": "2020-11-13         map.js:18
04:33:39.013428", "name": "Entity_ONE", "message_id":
"Entity_ONE_d7e48620-95c6-42c2-8554-deb0e2b5822c", "lat":
52.532488, "lon": 13.38064}
{"topic": "geostream", "timestamp": "2020-11-13         map.js:18
04:33:39.243876", "name": "Entity_ONE", "message_id":
"Entity_ONE_1a53eacc-b397-491a-bfc2-6b4977856cbc", "lat":
52.532736, "lon": 13.380404}
>
```