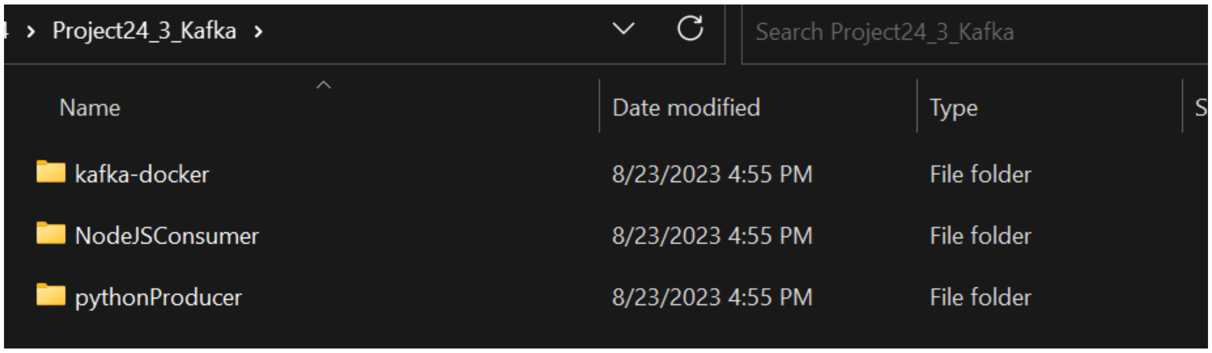
1. Download and extract the [Project24_3_Kafka]() folder.
   Provide a screenshot to show that you have downloaded and extracted the `Project 24_3_Kafka` folder.

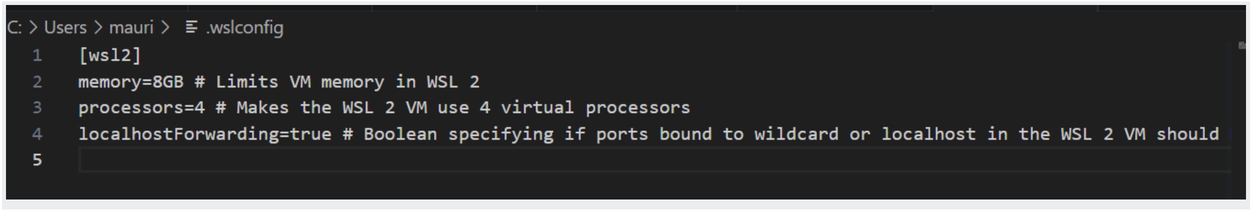


› Project24_3_Kafka ›

Name	Date modified	Type
kafka-docker	8/23/2023 4:55 PM	File folder
NodeJSConsumer	8/23/2023 4:55 PM	File folder
pythonProducer	8/23/2023 4:55 PM	File folder

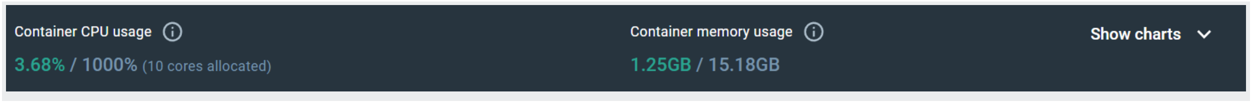
2. Open your Docker Desktop. Select the gear icon on the top right of the Docker Desktop window, then select Resources. Ensure that your CPU and memory are both set to a minimum of 6 GB.
   Provide a screenshot to show that you correctly configured your Docker Desktop and set your CPU and memory to a minimum of 6 GB.
   - Windows Users:
     Inside your user home folder: `C:\Users\"YOUR_WINDOWS_USERNAME"`, create a new file titled `.wslconfig`. Copy and and paste the following lines of code inside of the `.wslconfig` file:

```
[wsl2]
memory=8GB # Limits VM memory in WSL 2
processors=4 # Makes the WSL 2 VM use 4 virtual processors
localhostForwarding=true # Boolean specifying if ports bound to
wildcard or localhost in the WSL 2 VM should be connectable from
the host via localhost:port.
```



C: › Users › mauri › .wslconfig

```
[wsl2]
memory=8GB # Limits VM memory in WSL 2
processors=4 # Makes the WSL 2 VM use 4 virtual processors
localhostForwarding=true # Boolean specifying if ports bound to wildcard or localhost in the WSL 2 VM should

```



Container CPU usage
3.68% / 1000% (10 cores allocated)

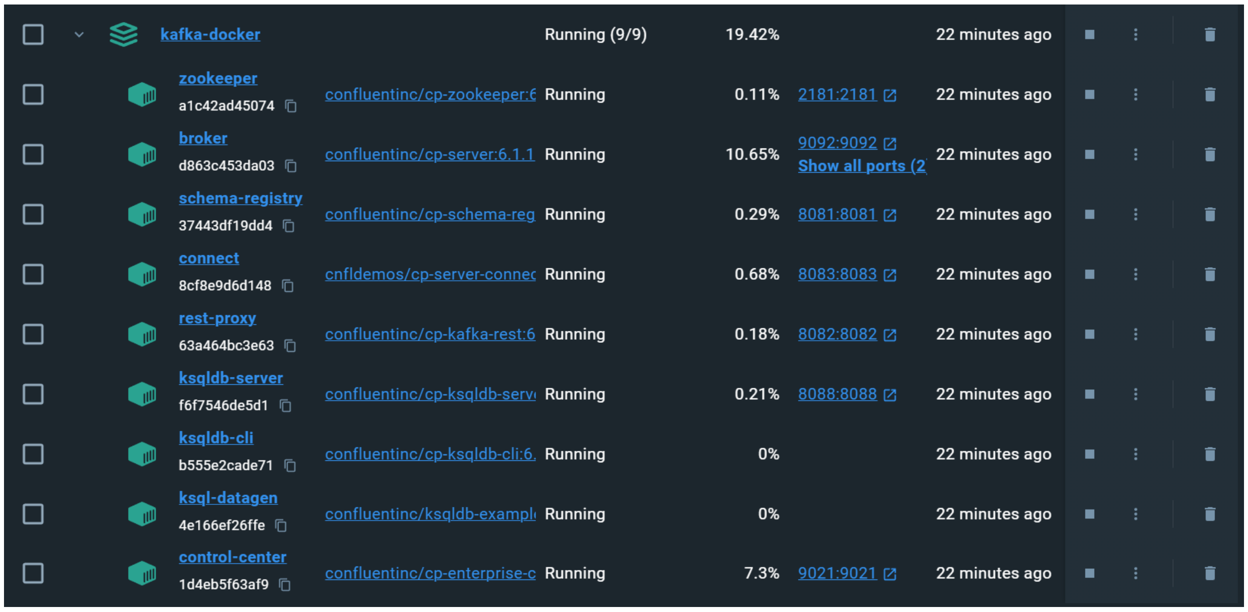
Container memory usage
1.25GB / 15.18GB

3. Next, you will create the Kafka Docker *container* components. Navigate to the `kafka-docker` folder inside the `Project24_3_Kafka` folder. In a Terminal window, run the following command to initialize your Kafka *container*:

```
docker-compose up
```

Open the Docker Desktop and confirm that all of the required Kafka *container* components are now up and running. The nine required *container* components are: `zookeeper`, `broker`, `schema-registry`, `rest-proxy`, `connect`, `ksqlbd-server`, `ksql-datagen`, `ksqlbd-cli`, and `control-center`.

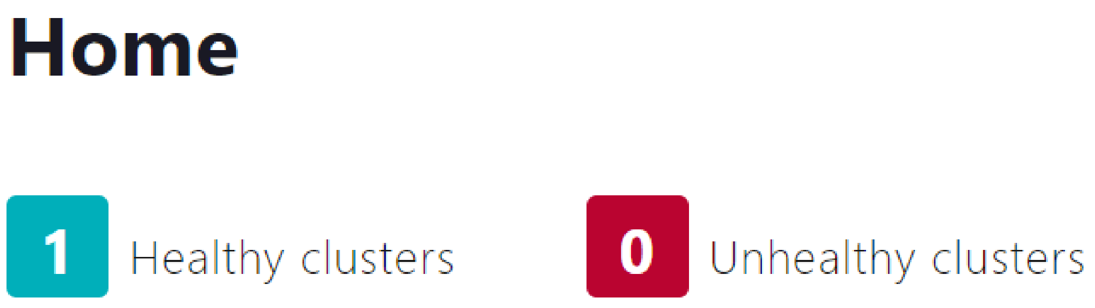
Provide a screenshot to show that you successfully executed the `docker-compose` command and created all of the required nine Kafka *container* components: `zookeeper`, `broker`, `schema-registry`, `rest-proxy`, `connect`, `ksqlbd-server`, `ksql-datagen`, `ksqlbd-cli`, and `control-center`.



kafka-docker		Running (9/9)	19.42%		22 minutes ago
zookeeper a1c42ad45074	confluentinc/cp-zookeeper:6	Running	0.11%	2181:2181	22 minutes ago
broker d863c453da03	confluentinc/cp-server:6.1.1	Running	10.65%	9092:9092 Show all ports (2	22 minutes ago
schema-registry 37443df19dd4	confluentinc/cp-schema-reg	Running	0.29%	8081:8081	22 minutes ago
connect 8cf8e9d6d148	cnfldemos/cp-server-connec	Running	0.68%	8083:8083	22 minutes ago
rest-proxy 63a464bc3e63	confluentinc/cp-kafka-rest:6	Running	0.18%	8082:8082	22 minutes ago
ksqldb-server f6f7546de5d1	confluentinc/cp-ksqldb-serv	Running	0.21%	8088:8088	22 minutes ago
ksqldb-cli b555e2cade71	confluentinc/cp-ksqldb-cli:6.	Running	0%		22 minutes ago
ksql-datagen 4e166ef26ffe	confluentinc/ksqldb-exampl	Running	0%		22 minutes ago
control-center 1d4eb5f63af9	confluentinc/cp-enterprise-c	Running	7.3%	9021:9021	22 minutes ago

4. Open your browser and navigate to the following URL: http://localhost:9021. This is the Confluent Control Center, and it will allow you to check the status of the Kafka *cluster* that you spun up.

Provide a screenshot of the Confluent Control Center web interface to show that the *cluster* status is healthy.

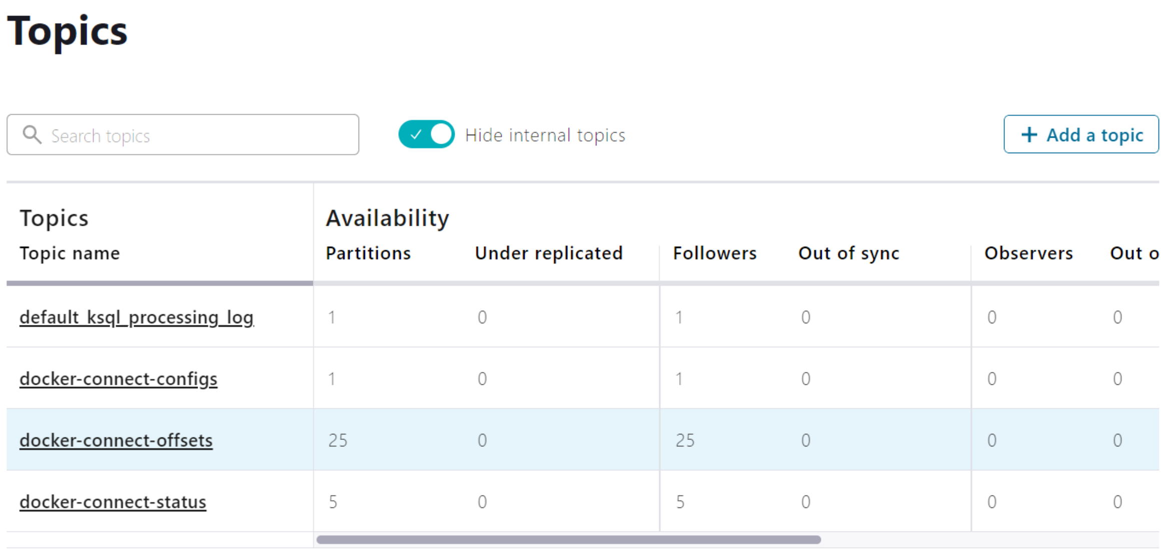

Home

1 Healthy clusters

0 Unhealthy clusters

5. Navigate to the *Topics* list on the Confluent Control Center by selecting "*Topics*" in the controlcenter.cluster in the Confluent Control Center.

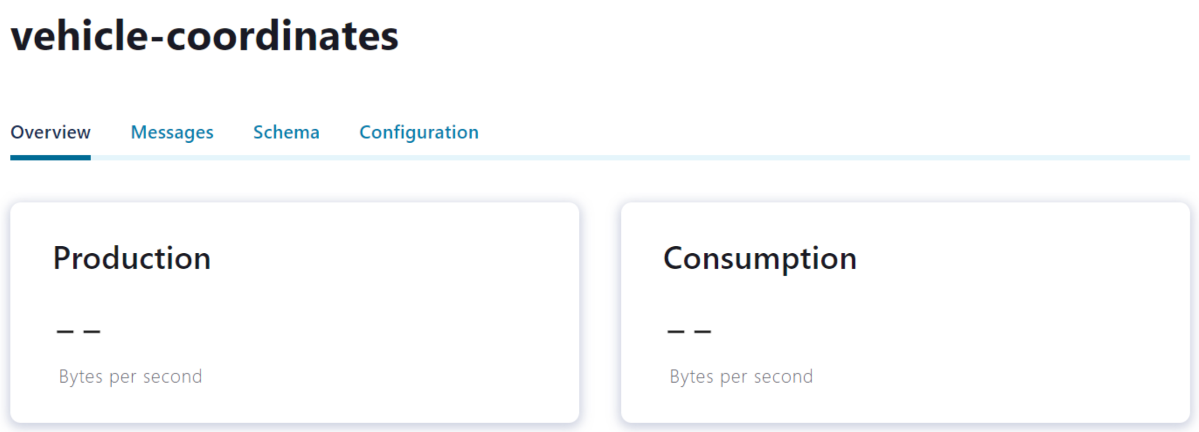
Provide a screenshot to show the existing default *topics* in the Confluent Control Center.


Topics

Topics	Availability					
Topic name	Partitions	Under replicated	Followers	Out of sync	Observers	Out o
default_ksql_processing_log	1	0	1	0	0	0
docker-connect-configs	1	0	1	0	0	0
docker-connect-offsets	25	0	25	0	0	0
docker-connect-status	5	0	5	0	0	0

6. Next, you will be writing Python code to simulate an IoT device that *publishes* the longitude and latitude coordinates of vehicles' locations to Kafka. From http://localhost:9021, select *Topics* and then select "Add a *topic*". Title the new *topic* `vehicle-coordinates`, set the number of partitions to 1, and select "Create with default settings".

Provide a screenshot to show that you have successfully added the `vehicle-coordinates` *topic* with the correct settings.


vehicle-coordinates

Overview Messages Schema Configuration

Production

– –

Bytes per second

Consumption

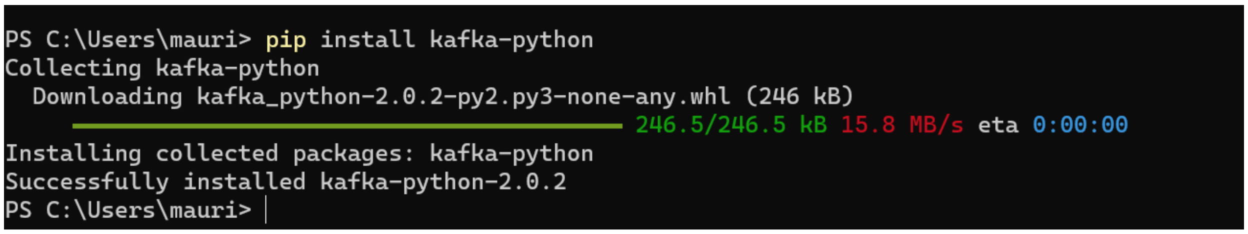
– –

Bytes per second

7. From the command prompt, run the following command to install the Kafka Python *client*:

```
pip install kafka-python
```

Provide a screenshot to show that you have successfully run the `pip install kafka-python` command.



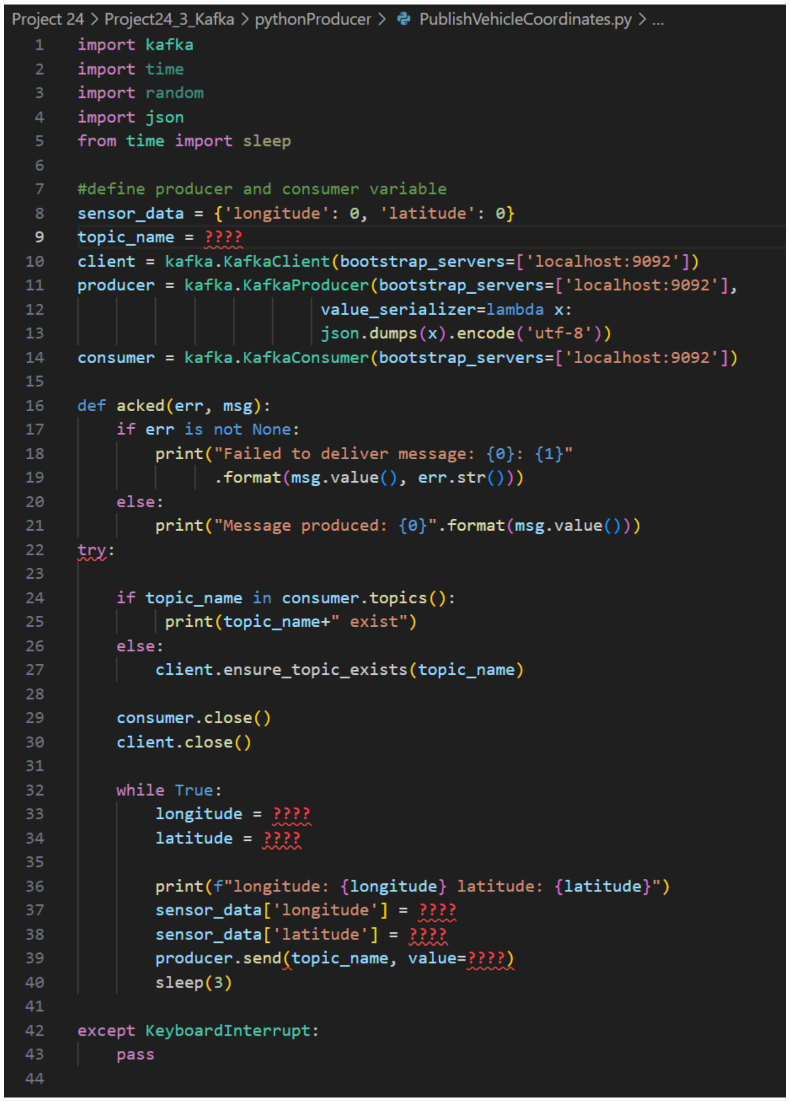
```
PS C:\Users\mauri> pip install kafka-python
Collecting kafka-python
  Downloading kafka_python-2.0.2-py2.py3-none-any.whl (246 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 246.5/246.5 kB 15.8 MB/s eta 0:00:00
Installing collected packages: kafka-python
Successfully installed kafka-python-2.0.2
PS C:\Users\mauri>
```

8. In VS Code, open the `PublishVehicleCoordinates.py` file inside the `pythonProducer` folder. Paste the code below into the file to create the template to define the *producer*:

```
import kafka
import time
import random
import json
from time import sleep

#define producer and consumer variable
sensor_data = {'longitude': 0, 'latitude': 0}
topic_name = ????
client =
kafka.KafkaClient(bootstrap_servers=['localhost:9092'])
producer =
kafka.KafkaProducer(bootstrap_servers=['localhost:9092'],
                         value_serializer=lambda x:
                         json.dumps(x).encode('utf-8'))
consumer =
kafka.KafkaConsumer(bootstrap_servers=['localhost:9092'])

def acked(err, msg):
    if err is not None:
        print("Failed to deliver message: {0}: {1}"
              .format(msg.value(), err.str()))
    else:
        print("Message produced: {0}".format(msg.value()))

try:


    if topic_name in consumer.topics():
         print(topic_name+" exist")
    else:
        client.ensure_topic_exists(topic_name)

    consumer.close()
    client.close()

    while True:
        longitude = ????
        latitude = ????


        print(f"longitude: {longitude} latitude:
{latitude}")
        sensor_data['longitude'] = ????
        sensor_data['latitude'] = ????
```

```
            producer.send(topic_name, value=????)
            sleep(3)

    except KeyboardInterrupt:
        pass
```

Provide a screenshot to show that you have successfully opened the `PublishVehicleCoordinates.py` file to create the template to define the *producer*.



Project 24 > Project24_3_Kafka > pythonProducer > PublishVehicleCoordinates.py > ...

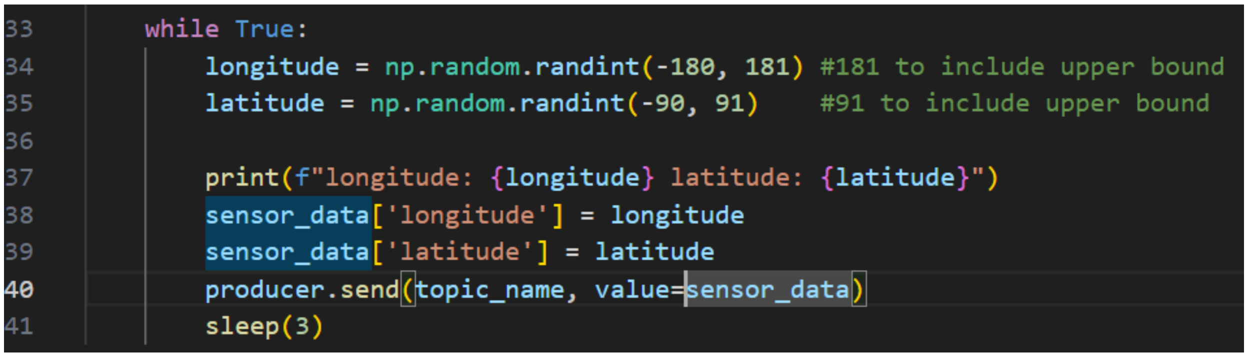
```
import kafka
import time
import random
import json
from time import sleep

#define producer and consumer variable
sensor_data = {'longitude': 0, 'latitude': 0}
topic_name = ????
client = kafka.KafkaClient(bootstrap_servers=['localhost:9092'])
producer = kafka.KafkaProducer(bootstrap_servers=['localhost:9092'],
                                value_serializer=lambda x:
                                json.dumps(x).encode('utf-8'))
consumer = kafka.KafkaConsumer(bootstrap_servers=['localhost:9092'])

def acked(err, msg):
    if err is not None:
        print("Failed to deliver message: {0}: {1}"
              .format(msg.value(), err.str()))
    else:
        print("Message produced: {0}".format(msg.value()))
try:

    if topic_name in consumer.topics():
         print(topic_name+" exist")
    else:
        client.ensure_topic_exists(topic_name)

    consumer.close()
    client.close()

    while True:
        longitude = ????
        latitude = ????

        print(f"longitude: {longitude} latitude: {latitude}")
        sensor_data['longitude'] = ????
        sensor_data['latitude'] = ????
        producer.send(topic_name, value=????)
        sleep(3)

except KeyboardInterrupt:
    pass
```

9. Modify the code in the `PublishVehicleCoordinates.py` file as instructed below:

a. Set the `topic_name` variable equal to a *string* with the `vehicle-coordinates` value.

b. In the `while` *loop*, use the correct NumPy *function* to simulate random *integer* values for the `longitude` and the `latitude`. The values for `longitude` should be random *integers* between −180 and 180. The values for `latitude` should be random *integers* between −90 and 90.

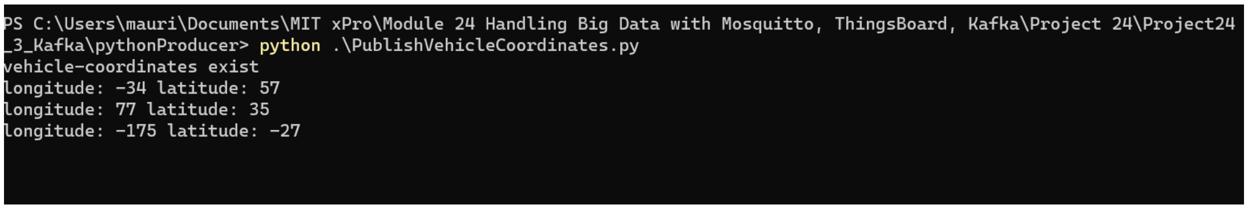
Provide a screenshot to show that you have correctly modified the code in the `PublishVehicleCoordinates.py` file.



```
while True:
    longitude = np.random.randint(-180, 181) #181 to include upper bound
    latitude = np.random.randint(-90, 91)    #91 to include upper bound

    print(f"longitude: {longitude} latitude: {latitude}")
    sensor_data['longitude'] = longitude
    sensor_data['latitude'] = latitude
    producer.send(topic_name, value=sensor_data)
    sleep(3)
```

10. From the Terminal window, run the code in the `PublishVehicleCoordinates.py` file to produce the longitude and latitude data.

Provide a screenshot to show that you successfully ran the Python code from the command prompt and that the longitude and the latitude data is being produced.

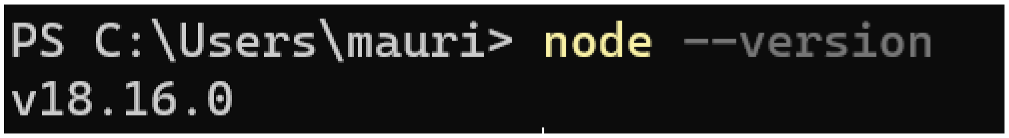


```
PS C:\Users\mauri\Documents\MIT xPro\Module 24 Handling Big Data with Mosquitto, ThingsBoard, Kafka\Project 24\Project24
_3_Kafka\pythonProducer> python .\PublishVehicleCoordinates.py
vehicle-coordinates exist
longitude: -34 latitude: 57
longitude: 77 latitude: 35
longitude: -175 latitude: -27
```

11. Install Node.js as shown in Mini-Lesson 24.6. From the command prompt, run the command below to verify that Node.js is installed:

```
node --version
```

Provide a screenshot to show that you successfully ran the command to return the Node.js version number.

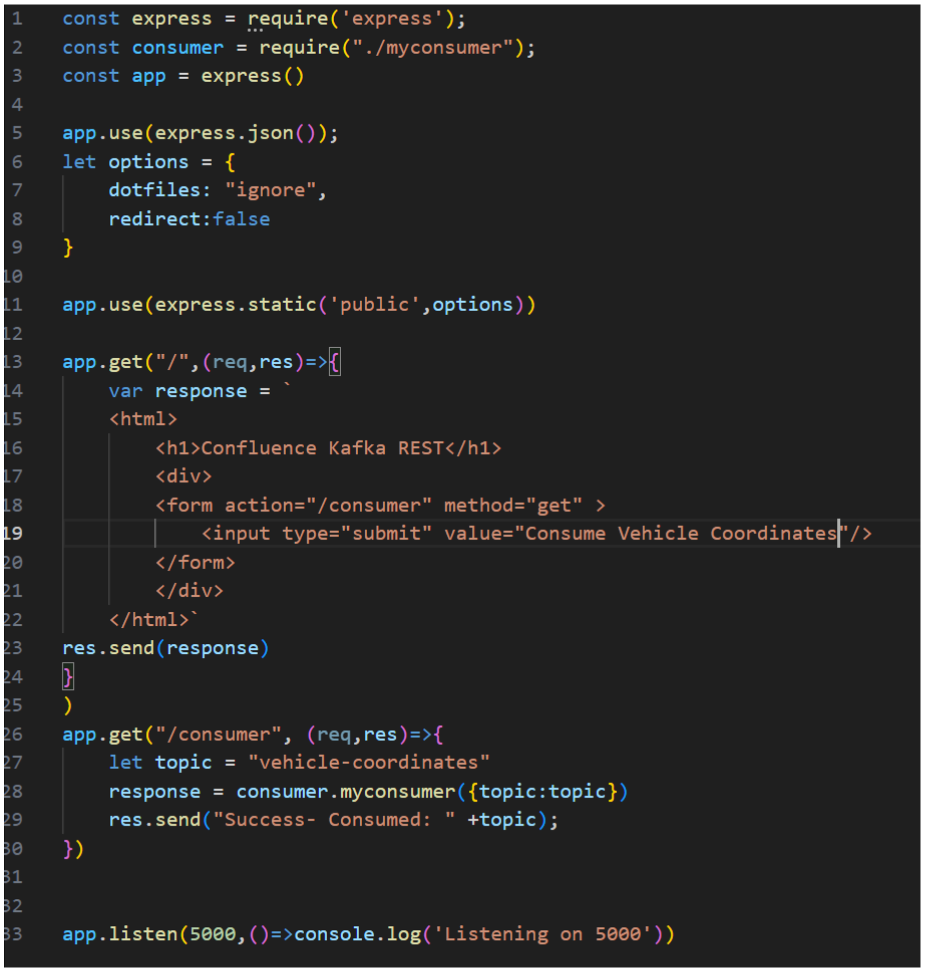


```
PS C:\Users\mauri> node --version
v18.16.0
```

12. Now you will write a Node.js web page to consume the location data produced by the `PublishVehicleCoordinates.py` code. Move the `PublishVehicleCoordinates.py` file inside the `NodeJSConsumer` folder. Modify the `server.js` file inside the `NodeJSConsumer` folder as follows:

- Change the *topic* to `vehicle-coordinates`.
- Change the HTML code on line 19 to display the `Consume Vehicle Coordinates` value.
- Save the file.

Provide a screenshot to show that you changed the `server.js` file to consume the `vehicle-coordinates` *topic* and changed the button value to display the `Consume Vehicle Coordinates` value.

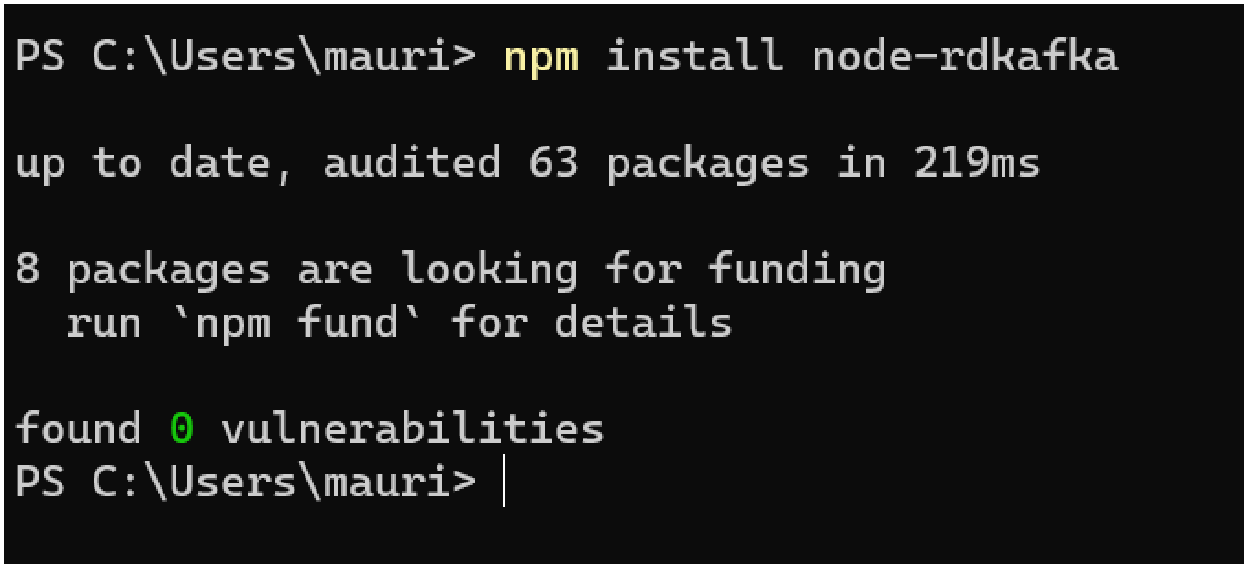


```
1    const express = require('express');
2    const consumer = require("./myconsumer");
3    const app = express()
4
5    app.use(express.json());
6    let options = {
7        dotfiles: "ignore",
8        redirect:false
9    }
10
11   app.use(express.static('public',options))
12
13   app.get("/",(req,res)=>{
14       var response = `
15       <html>
16           <h1>Confluence Kafka REST</h1>
17           <div>
18           <form action="/consumer" method="get" >
19               <input type="submit" value="Consume Vehicle Coordinates"/>
20           </form>
21           </div>
22       </html>`
23   res.send(response)
24   }
25   )
26   app.get("/consumer", (req,res)=>{
27       let topic = "vehicle-coordinates"
28       response = consumer.myconsumer({topic:topic})
29       res.send("Success- Consumed: " +topic);
30   })
31
32
33   app.listen(5000,()=>console.log('Listening on 5000'))
```

13. In a Terminal window, run the following command to install the Kafka JavaScript *library*. This will be used to consume Kafka messages in your Node.js web application.

```
npm install node-rdkafka
```

Provide a screenshot to show that you successfully installed the `node-rdkafka` *library*.

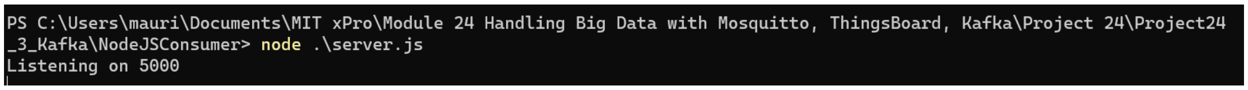


```
PS C:\Users\mauri> npm install node-rdkafka

up to date, audited 63 packages in 219ms

8 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\mauri> 
```

14. In a Terminal window, navigate to the `Project24_3` folder. Start the web page by running the following command:

```
node server.js
```

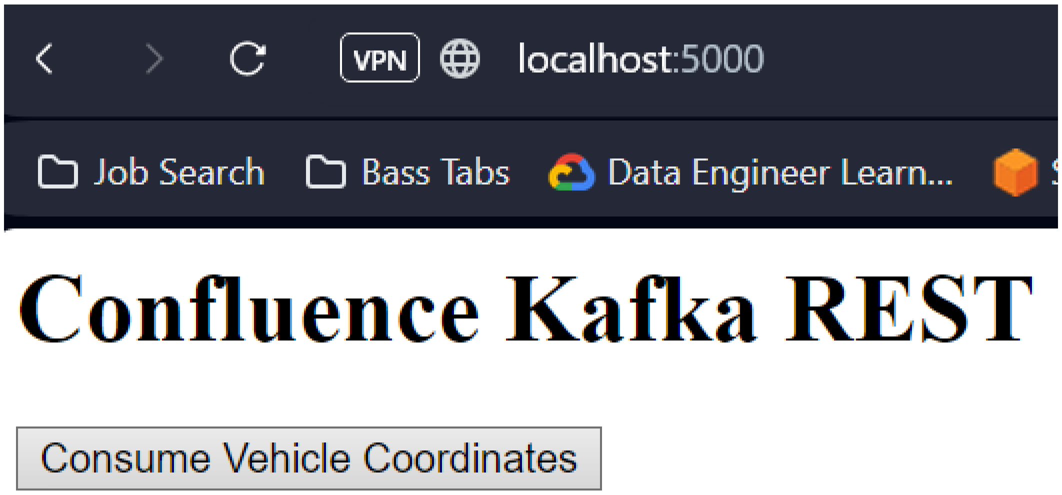
Provide a screenshot to show that you successfully ran the `node server.js` command and displayed the web page.



```
PS C:\Users\mauri\Documents\MIT xPro\Module 24 Handling Big Data with Mosquitto, ThingsBoard, Kafka\Project 24\Project24
_3_Kafka\NodeJSConsumer> node .\server.js
Listening on 5000
```

15. Navigate to http://localhost:5000. Your web page should be titled “Confluence Kafka REST” and have a button labeled “Consume Vehicle Coordinates”.

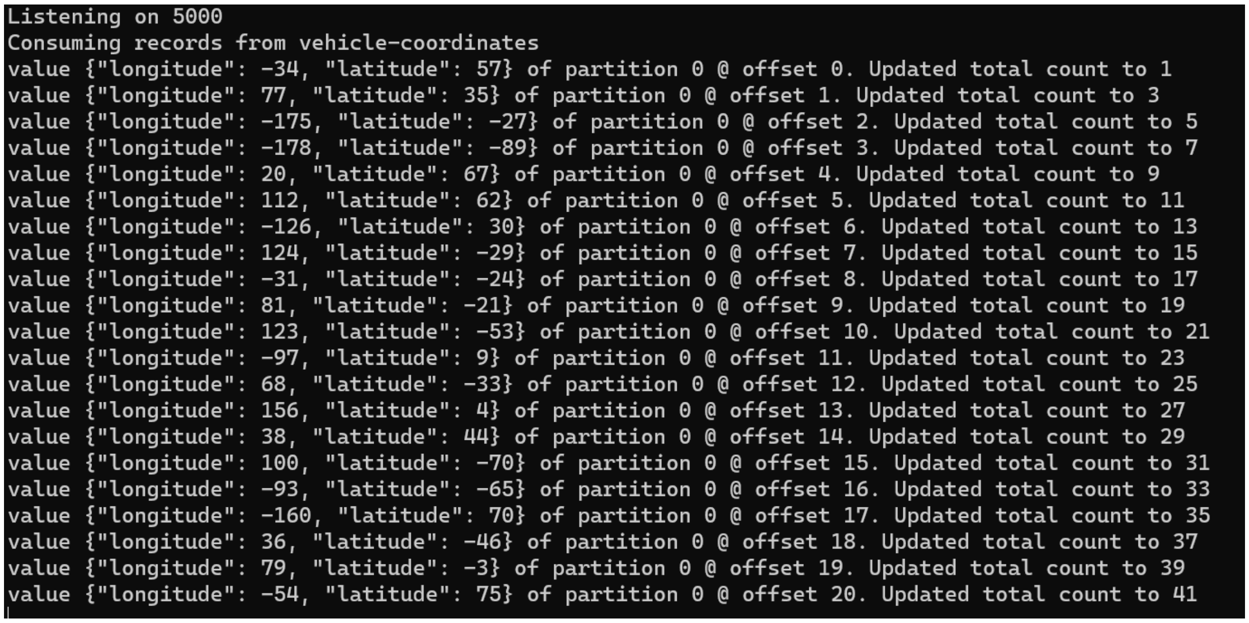
Provide a screenshot to show that the web page loaded correctly with a title of “Confluence Kafka REST” and a button labeled “Consume Vehicle Coordinates”.



Confluence Kafka REST

16. Select the “Consume Vehicle Coordinates” button to verify that your data is being consumed and that there are no errors.

Provide a screenshot of the console to show the data being consumed without errors.



```
Listening on 5000
Consuming records from vehicle-coordinates
value {"longitude": -34, "latitude": 57} of partition 0 @ offset 0. Updated total count to 1
value {"longitude": 77, "latitude": 35} of partition 0 @ offset 1. Updated total count to 3
value {"longitude": -175, "latitude": -27} of partition 0 @ offset 2. Updated total count to 5
value {"longitude": -178, "latitude": -89} of partition 0 @ offset 3. Updated total count to 7
value {"longitude": 20, "latitude": 67} of partition 0 @ offset 4. Updated total count to 9
value {"longitude": 112, "latitude": 62} of partition 0 @ offset 5. Updated total count to 11
value {"longitude": -126, "latitude": 30} of partition 0 @ offset 6. Updated total count to 13
value {"longitude": 124, "latitude": -29} of partition 0 @ offset 7. Updated total count to 15
value {"longitude": -31, "latitude": -24} of partition 0 @ offset 8. Updated total count to 17
value {"longitude": 81, "latitude": -21} of partition 0 @ offset 9. Updated total count to 19
value {"longitude": 123, "latitude": -53} of partition 0 @ offset 10. Updated total count to 21
value {"longitude": -97, "latitude": 9} of partition 0 @ offset 11. Updated total count to 23
value {"longitude": 68, "latitude": -33} of partition 0 @ offset 12. Updated total count to 25
value {"longitude": 156, "latitude": 4} of partition 0 @ offset 13. Updated total count to 27
value {"longitude": 38, "latitude": 44} of partition 0 @ offset 14. Updated total count to 29
value {"longitude": 100, "latitude": -70} of partition 0 @ offset 15. Updated total count to 31
value {"longitude": -93, "latitude": -65} of partition 0 @ offset 16. Updated total count to 33
value {"longitude": -160, "latitude": 70} of partition 0 @ offset 17. Updated total count to 35
value {"longitude": 36, "latitude": -46} of partition 0 @ offset 18. Updated total count to 37
value {"longitude": 79, "latitude": -3} of partition 0 @ offset 19. Updated total count to 39
value {"longitude": -54, "latitude": 75} of partition 0 @ offset 20. Updated total count to 41
```

You have created a *producer/consumer* pair for handling IoT data. Kafka can handle large volumes of data that could potentially be generated in an environment like this. It is also fault tolerant. That is, it would continue to operate properly in the event of the failure of one or more of its components.