n this project, you will work with Kafka, one of the most successful applications for handling *streaming* data at scale. You will use a Docker *image* created by Confluent that installs all of the necessary Kafka components including, among others, the *broker* and ZooKeeper. Kafka allows you to store messages in *topics*, which are distributed across multiple *brokers* by splitting *topics* into data partitions that save you from unfortunate *broker* failures.

To get started with this project, you will begin with a simple Kafka implementation. With this implementation, you will create a Python application that *publishes* vehicle location longitude-latitude data to a Kafka *topic*. Next, you will use Node.js to start a web *server* that acts as a *consumer* for the messages received from the Kafka application.

**To complete this project, follow these steps:**

1. Download and extract the [Project24\_3\_Kafka](https://classroom.emeritus.org/courses/10605/files/3006889/download) folder.  
   Provide a screenshot to show that you have downloaded and extracted the Project 24\_3\_Kafka 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.

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.

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* you spun up.

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

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 showing the existing default *topics* in the Confluent Control Center.

6. Next, you will write 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.

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.

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*.

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

1. Set the topic\_name variable equal to a *string* with the vehicle-coordinates value.
2. 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.

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.

11. Install Node.js as shown in [Mini-Lesson 24.6](https://classroom.emeritus.org/courses/10605/pages/mini-lesson-24-dot-6-introduction-to-node-dot-js-30-00). 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.

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.

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*.

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.

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

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.

Congratulations on completing the project. 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.

**Submission Instructions:**

Your submission for this project should be a Word document that includes the following screenshots, each labeled for the step that the screenshot represents:

1. Provide a screenshot to show that you have downloaded and extracted the Project 24\_3\_Kafka folder.
2. Provide a screenshot to show that you correctly configured your Docker Desktop and set your CPU and memory to a minimum of 6 GB.
3. 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.
4. Provide a screenshot of the Confluent Control Center web interface to show that the *cluster* status is healthy.
5. Provide a screenshot to show the existing default *topics* in the Confluent Control Center.
6. Provide a screenshot to show that you have successfully added the vehicle-coordinates *topic* with the correct settings.
7. Provide a screenshot to show that you have successfully run the pip install kafka-python command.
8. Provide a screenshot to show that you have successfully opened the PublishVehicleCoordinates.py file to create the template to define the *producer*.
9. Provide a screenshot to show that you have correctly modified the code in the PublishVehicleCoordinates.py file.
10. 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.
11. Provide a screenshot to show that you successfully ran the command to return the Node.js version number.
12. 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.
13. Provide a screenshot to show that you successfully installed the node-rdkafka *library*.
14. Provide a screenshot to show that you successfully ran the node server.js command and displayed the web page.
15. 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”.
16. Provide a screenshot of the console to show the data being consumed without errors.