In Video 16.1, Dr. Sanchez demonstrated how to create a prototype of a web application that uses Mapbox to display the positions of buses along Route 1 for the MBTA. The longitude and latitude for the buses are hardcoded for the prototype and are incremented periodically to simulate movement.

Before working through the steps of this project, be sure to review the Submission Instructions and Project 16.1 Rubric to ensure that you collect all required screenshots as you work through the project.

For this project, you will add the following enhancements to the prototype:

1. You will create a MySQL database in a Docker *container* to store data returned by the MBTA API.
2. You will make calls to the MBTA API for Route 1 periodically (every 10 seconds). You will parse the JSON results returned and will store the data in a MySQL database for further analysis.
3. You will perform change data capture (CDC) on the MySQL database. Your application will monitor the MySQL database for changes and propagate any changes to a MongoDB database.
4. Let the *server* run for a period of 12 hours, storing data in the MySQL database. Use the [Project 16.1 Jupyter Notebook template](https://classroom.emeritus.org/courses/10605/files/3007201/download) to load and analyze the data stored in the database. Make sure you answer the following questions in your Jupyter Notebook submission:
   1. What is the average time it takes for a bus to complete Route 1?
   2. Include a *plot*-type visualization based on the data. The type of *plot* you choose to include is up to you.
   3. Give an estimate of the speed of the bus from current\_stop\_sequence = 1 to the last current\_stop\_sequence. **Note**: You can use the [haversine](https://pypi.org/project/haversine/)
   4. [Links to an external site.](https://pypi.org/project/haversine/)
   5. Python *library* to calculate the distance between two points given two longitude and latitude coordinates.

There will be two submissions for this project: The first submission is a Word document that includes the screenshots listed in the Submission instructions, each labeled for the step that the screenshot represents. The second submission is a Jupyter Notebook, using the provided template, to answer the three questions listed above.

This project is worth a total of 100 points. The screenshots in the Word document submission are worth a total of 50 points with the point distribution that is delineated in the Project 16.1 Rubric. The Jupyter Notebook is worth a total of 50 points with the point distribution that is delineated in the [Project 16.1 Jupyter Notebook template](https://classroom.emeritus.org/courses/10605/files/3007201/download) and in the Project 16.1 Rubric.

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

Your project should utilize Docker *containers* running on the same Docker network. Your project should contain the following components:

1. Create a Docker network called MBTANetwork. Associate all Docker *containers* to this network. Provide a screenshot to show that you have successfully created the MBTANetwork network.
2. Unzip and open the [mysqlDocker](https://classroom.emeritus.org/courses/10605/files/3007449/download) folder. Provide a screenshot to show that you have successfully opened the mysqlDocker folder.
   1. Using VS Code, open the MBTA.sql file. Provide a screenshot to show that you have successfully opened the MBTA.sql file.
   2. In a Jupyter Notebook, run the following code:

| mbtaURL = "https://api-v3.mbta.com/vehicles?filter[route]=1&include=trip"  import urllib.request, json with urllib.request.urlopen(mbtaURL) as url:  data = json.loads(url.read().decode())    with open('data.json', 'w') as outfile:  json.dump(data, outfile)    with open('data.txt', 'w') as outfile:  json.dump(json.dumps(data, indent=4, sort\_keys=True), outfile)    print(json.dumps(data, indent=4, sort\_keys=True)) |
| --- |

* 1. Provide a screenshot to show that you have successfully run the code above in a Jupyter Notebook.  
     This code will allow you to analyze calls to the MBTA API. Decide which additional fields returned by the service should be included in your table. Remember, you should not only think about the immediate needs of your solution (i.e., ID, latitude, and longitude) but also about the future needs of your application. Add these additional fields to the mbta\_buses table inside the MBTA.sql file. Provide a screenshot to show that you have successfully added at least five additional fields to the mbta\_buses table.  
     If you want, you can also use the same Jupyter Notebook to practice parsing through the JSON results and getting to the fields you will need to insert data into the database.  
     Here are some additional links for you to familiarize yourself with the MBTA API:
     + [About the V3 MBTA API](https://www.mbta.com/developers/v3-api)
     + [Links to an external site.](https://www.mbta.com/developers/v3-api)
     + [MBTA Swagger](https://api-v3.mbta.com/docs/swagger/index.html)
     + [Links to an external site.](https://api-v3.mbta.com/docs/swagger/index.html)
  2. After modifying the MBTA.sql file and adding the additional fields you have selected, navigate from a shell prompt to the folder where you have the Dockerfile stored and run the Docker command to create a Docker *image* called mysqlmbtamasterimg. Provide a screenshot to show that you have successfully created the mysqlmbtamasterimg Docker *image*.
  3. Create a Docker *container* associated with the MBTANetwork network that you created in Step 1. Name the Docker *container* mysqlserver. Provide a screenshot to show that you have successfully created the mysqlserver Docker *container*.

1. MongoDB Docker *container*:  
   Create a MongoDB Docker *container* to be used for CDC. Be sure that the MongoDB *container* is part of the same network (MBTANetwork) as the other *containers* in your project. Name the *container* some-mongo. Provide a screenshot to show that you have successfully created the some-mongo Docker *container*.
2. Flask web *server* (Note: This will be running on your local machine.)
   1. Unzip the [Module16ProjectFlask.zip](https://classroom.emeritus.org/courses/10605/files/3007084/download) folder on your local machine and open it using VS Code. Provide a screenshot to show that you have successfully opened the Module16ProjectFlask.zip folder in VS Code.
   2. Modify the code in the mysqldb.py file to add all the columns that you defined in the mbta\_buses SQL table following the template provided in the mysqldb.py file. Provide a screenshot to show that you have successfully modified the mysqldb.py file.
   3. Modify the code in the MBTAApiClient.py file to parse all the columns that you defined in the mbta\_buses SQL table following the template provided in the MBTAApiClient.py file. Provide a screenshot to show that you have successfully modified the MBTAApiClient.py file.
   4. Modify the code in the index.html file inside the Module16ProjectFlask.zip folder and add your Mapbox access *token*. Provide a screenshot to show that you have successfully added your Mapbox access *token* in the index.html file.
   5. Modify the code in the server.py file to initialize the buses *list* by doing an API call to the MBTA database. For this part, use the callMBTAApi() *function* from the MBTAApiClient *library*. Provide a screenshot to show that you have successfully initialized the buses *list* in the server.py file.
   6. From VS Code, run the server.py file. Provide a screenshot to show that you have successfully run the server.py file in VS Code.
   7. Open a browser window and navigate to localhost:3000. Provide a screenshot to show that you have successfully navigated to localhost:3000.
3. Debezium CDC monitor *container*:  
   1. Unzip the [DebeziumCDC.zip](https://classroom.emeritus.org/courses/10605/files/3007330/download) folder on your local machine and open it using VS Code. Provide a screenshot to show that you have successfully opened the DebeziumCDC.zip folder in VS Code.
   2. From the DebeziumCDC.zip folder, create a Docker *image* called debeziummodule16. Provide a screenshot to show that you have successfully created the debeziummodule16 Docker *image*.
   3. Create the Docker *container* for Debezium and make sure you associate it with the MBTANetwork network. Provide a screenshot to show that you have successfully created the Docker *container* and associated it with the MBTANetwork network.
   4. Once the Debezium *container* is running, open a shell (from <CLI> in the Debezium Docker *container*) and go through the steps of installing the nano text editor. Provide a screenshot to show that you have successfully installed the nano text editor in your shell.
   5. From the Debezium shell, navigate to the following file:  
      /tmp/src/main/java/mit/edu/tv/listener/MongoDB.java  
      Use the nano text editor to edit the Java *class* to modify the insertRecord *method* in the MongoDB.java *class* to write data to the MongoDB database:  
      Add the following code to the insertRecord *function* to insert a *document* into the MongoDB database:

| MongoClient mongoClient = MongoClients.create(connectionString); MongoDatabase database = mongoClient.getDatabase("myDatabase"); Document document = new Document(); document.append("recordId", "CDC"); document.append("value", record); database.getCollection("myCollection").insertOne(document); |
| --- |

* 1. **Note:** The code provided requires that your MongoDB *container* is titled some-mongo. If you decide to use another name, be sure to modify the code accordingly.  
     Provide a screenshot to show that you have successfully modified the MongoDB.java *class*.
  2. From the Debezium shell, navigate to the following file:  
     /tmp/src/main/java/mit/edu/tv/listener/DebeziumListener.java  
     Inside the handleChangeEvent *method*, use the insertRecord *method* of the MongoDB *class* to insert a record inside the MongoDB database. Pass the following argument to the insertRecord *method*:  
     sourceRecord.value().toString()  
     Provide a screenshot to show that you have successfully modified the handleChangeEvent *method*.
  3. From the Debezium shell prompt, run the Maven SpringBoot application using the following command:  
     mvn spring-boot:run  
     Provide a screenshot to show that you have successfully run the Maven SpringBoot application.

1. Verify that the MongoDB database is being populated:
   1. Following the steps in [Mini-Lesson 16.4](https://classroom.emeritus.org/courses/10605/pages/mini-lesson-16-dot-4-performing-crud-operations-on-a-mongodb-database-using-java-30-00), create a *container* called javamaven to *query* the MongoDB database. Follow the steps to download the Java MongoDB *classes* and copy the files to the *container*. Provide a screenshot of your Docker desktop to show the javamaven *container* running.
   2. After you have installed the nano text editor, navigate to the following folder from the javamaven *container* bash prompt:  
      /java-quick-start/src/main/java/com/mongodb/quickstart  
      List the files in the *directory*.  
      Provide a screenshot to show that you successfully navigated to the *directory* and listed the files.
   3. Using the nano text editor, create a file called ReadCDC.java in the current *directory* (/java-quick-start/src/main/java/com/mongodb/quickstart) and copy the following code into the file:

| package com.mongodb.quickstart;  import com.mongodb.client.\*; import org.bson.Document;  import java.util.ArrayList; import java.util.List; import java.util.function.Consumer;  import static com.mongodb.client.model.Filters.\*; import static com.mongodb.client.model.Projections.\*; import static com.mongodb.client.model.Sorts.descending;  public class ReadCDC {  public static void main(String[] args) {  try (MongoClient mongoClient = MongoClients.create(System.getProperty("mongodb.uri"))) {  MongoDatabase sampleTrainingDB = mongoClient.getDatabase("myDatabase");  MongoCollection<Document> myCDCCollection = sampleTrainingDB.getCollection("myCollection");  Document cdcDocument = myCDCCollection.find(new Document("recordId", "CDC")).first();  System.out.println("CDC Record: " + cdcDocument.toJson());  }  } } |
| --- |

* 1. Provide a screenshot to show that you successfully created the ReadCDC.java file and copied the code.
  2. From the bash command prompt, make sure you are in the /java-quick-start folder and run the following command to execute the ReadCDC.java *class*:

| mvn compile exec:java -Dexec.mainClass="com.mongodb.quickstart.ReadCDC" -Dmongodb.uri="mongodb://some-mongo:27017" |
| --- |

* 1. Provide a screenshot to show the results of the bash command to execute the ReadCDC.java *class*.  
     This is the final step of creating screenshots for your Word document submission file. In the last step of this project, you will work on creating the second submission, which utilizes the Jupyter Notebook template provided below.

1. Be sure to leave the server.py file running for a period of 12 hours. Use the [Project 16.1 Jupyter Notebook template](https://classroom.emeritus.org/courses/10605/files/3007201/download) to load and analyze the data stored in the database. Make sure you answer the following questions in your Jupyter Notebook submission:
   1. What is the average time it takes for a bus to complete Route 1?
   2. Include a *plot*-type visualization based on the data. The type of *plot* you choose to include is up to you.
   3. Give an estimate of the speed of the bus from current\_stop\_sequence = 1 to the last current\_stop\_sequence. **Note:** You can use the [haversine](https://pypi.org/project/haversine/)
   4. [Links to an external site.](https://pypi.org/project/haversine/)
   5. Python *library* to calculate the distance between two points given two longitude and latitude coordinates.

(**Note:** You may have to stop the process that is performing CDC for the 12-hour period and only run the *server* that calls the MBTA API and stores the data in the MySQL database because the CDC process is memory-intensive, and your machine could run out of memory.)

**Submission Instructions:**

There will be two submissions for this project. The first submission is a Word document that includes the screenshots listed below, each labeled for the step that the screenshot represents. The second submission is a Jupyter Notebook, using the provided template, to answer the final three questions in the submission instructions. You can also check the Project 16.1 Rubric below for specific grading details.

Your first 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 successfully created the MBTANetwork network.
2. Provide a screenshot to show that you have successfully opened the mysqlDocker folder.
   1. Provide a screenshot to show that you have successfully opened the MBTA.sql file.
   2. For this step, you will provide two screenshots. The first screenshot should show that you have successfully run the provided code in a Jupyter Notebook. The second screenshot should show that you have successfully added at least five additional fields to the mbta\_buses table.
   3. Provide a screenshot to show that you have successfully created the mysqlmbtamasterimg Docker *image*.
   4. Provide a screenshot to show that you have successfully created the mysqlserver Docker *container*.
3. Provide a screenshot to show that you have successfully created the some-mongo Docker *container*.
4. 1. Provide a screenshot to show that you have successfully opened the Module16ProjectFlask.zip folder in VS Code.
   2. Provide a screenshot to show that you have successfully modified the mysqldb.py file.
   3. Provide a screenshot to show that you have successfully modified the MBTAApiClient.py file.
   4. Provide a screenshot to show that you have successfully added your Mapbox access *token* in the index.html file.
   5. Provide a screenshot to show that you have successfully initialized the buses *list* in the server.py file.
   6. Provide a screenshot to show that you have successfully run the server.py file in VS Code.
   7. Provide a screenshot to show that you have successfully navigated to localhost:3000.
5. 1. Provide a screenshot to show that you have successfully opened the DebeziumCDC.zip folder in VS Code.
   2. Provide a screenshot to show that you have successfully created the debeziummodule16 Docker *image*.
   3. Provide a screenshot to show that you have successfully created the Docker *container* and associated it with the MBTANetwork network.
   4. Provide a screenshot to show that you have successfully installed the nano text editor in your shell.
   5. Provide a screenshot to show that you have successfully modified the MongoDB.java *class*.
   6. Provide a screenshot to show that you have successfully modified the handleChangeEvent *method*.
   7. Provide a screenshot to show that you have successfully run the Maven SpringBoot application.
6. 1. Provide a screenshot of your Docker desktop to show the javamaven *container* running.
   2. Provide a screenshot to show that you successfully navigated to the *directory* and listed the files.
   3. Provide a screenshot to show that you successfully created the ReadCDC.java file and copied the code.
   4. Provide a screenshot to show the results of the bash command to execute the ReadCDC.java *class*.

Your second submission will be the [Project 16.1 Jupyter Notebook template](https://classroom.emeritus.org/courses/10605/files/3007201/download) with your answers to the following three questions included:

1. Answer the following questions in your Jupyter Notebook submission:
   1. What is the average time it takes for a bus to complete Route 1?
   2. Include a *plot*-type visualization based on the data. The type of *plot* you choose to include is up to you.
   3. Give an estimate of the speed of the bus from current\_stop\_sequence = 1 to the last current\_stop\_sequence. **Note**: You can use the [haversine](https://pypi.org/project/haversine/)
   4. [Links to an external site.](https://pypi.org/project/haversine/)
   5. Python *library* to calculate the distance between two points given two longitude and latitude coordinates.