# Exercise 2.3 - AWS IoT SDK

In this exercise, you will build a chat application between car1 and car2 so they can send messages to each other. There is already a piece of code written that asks for data on the command line indefinitely. You will need to implement the publishing of the messages and the subscription to the right IoT Topic.

To implement this, you will use 2 IoT Topics: lab/messaging/car1 and lab/messaging/car2. Car1 will subscribe to its IoT Topic (lab/messaging/car1) while car2 will subscribe to its own. Car1 will publish messages that it receives via command line to car2's IoT Topic while car2 will publish messages to car1's IoT Topic. They will be be able to communicate with each other this way.

The diagram below shows the resources and data flow that you will create in this exercise.

A screenshot of a computer

Description automatically generated with medium confidence

As coding is required for this exercise, you have the choice to use NodeJS or Python. The full solution is provided as option if you don't want to code and prefer to just run the code. However, if you do use the solution, it is important to read it to understand what was done to solve this use case.

The [documentation of the AWS IoT Device SDK for JavaScript](https://github.com/aws/aws-iot-device-sdk-js) is a great place for examples on how to do the tasks required. There are different examples to look at in the examples folder. That SDK is a wrapper for the [MQTT library for JavaScript](https://github.com/mqttjs/MQTT.js) which is also another great place to look for help.

If instead you prefer to use Python, the [GitHub page of the AWS IoT Device SDK for Python](https://github.com/aws/aws-iot-device-sdk-python) is a great place for examples on how to do the tasks required. There are different examples to look at in the samples folder. You can find the [documentation of the SDK linked on that page](https://s3.amazonaws.com/aws-iot-device-sdk-python-docs/html/index.html).

This exercise assumes that the resources from Exercise 1.1 haven't been deleted. If you have deleted those components, you will need to start with Exercise 1.1 again before continuing.

You have to make a choice of either doing this lab in NodeJS or Python. **Expand** one of the two sections below.

Expand me to code in NodeJS.

## 1. Download and open the code

In this section, you will download the piece of code that has already been created and from which you should continue the development.

### 1.1 Start Cloud9

Your Cloud9 environment may have shut down at this point as it's supposed to automatically shutdown after 30 minutes. To restart it, follow these steps:

1. In the AWS Management Console, click **Services**, and then click **Cloud9** to go to the Cloud9 console.
2. You should see a list of environments. If you don't, click on the hamburger menu icon (the three parallel lines) near the top left of the screen and click on **Your environments**.
3. Click the **Open IDE** button in the **IoTOnAWS** card.
4. It may take a minute for your environment to start.

### 1.2 Download and open the code

1. Download the application code in car1's folder by running the following commands in your AWS Cloud9 **terminal**:
2. cd ~/environment/car1

wget https://aws-tc-largeobjects.s3.amazonaws.com/OTP-AWS\_D5-2019/v1.0/code/exercise-2.3.js

1. In the left side of the Cloud9 environment, open the **car1/exercise-2.3.js** file by double-clicking on it.

## 2. Develop the application

In this section, you will look at the current state of the application by reading the current code. You will then develop each of the TODOs that were left inside of the code. The solution can be found in each of the sections below to validate your code. If you prefer to use the solution directly, you can read each of the sections below and go to Section 2.6 to download the solution code. **If you prefer not to code at all, skip to Section 2.6 Full solution code (optional).**

### 2.1 Current application state

You can find below the current state of the application.

* The readline library is used to read the input from the Cloud9 terminal console so you don't need to implement that piece.
* The readConsoleInput function creates an infinite async loop to read the input from the console. Each time an input is read, it sends it to the publishToIoTTopic function which you will need to implement in TODO 4.
* The host parameter to establish the connection to your AWS IoT endpoint is read from the endpoint.json file and stored in the host variable.
* The Certificate, Root CA and Private Key paths are all stored in variables ready for you to use.
* The aws-iot-device-sdk library has already been imported as the awsIoT variable. You will need to use this for TODO 1.
* The code has been built to look at the current folder's name and establish what the publish and subscribe topic is based on it. The subTopic variable contains the IoT Topic to subscribe to. The pubTopic variable contains the IoT Topic to publish to.

### 2.2 Develop TODO 1

In TODO 1, use the awsIoT library to create the [device](https://github.com/aws/aws-iot-device-sdk-js#device) object from the aws-iot-device-sdk library. The required arguments have already been assigned as variables in the code: host, clientId, certPath, keyPath and caPath. Those are the only parameters required for this exercise, but feel free to use others. The device returns a wrapper for the [mqtt.Client()](https://github.com/mqttjs/MQTT.js#client) class thus making all of its functions available for you to use. You will use a few of those in the next TODOs.

Solution for TODO 1.

### 2.3 Develop TODO 2

In TODO 2, implement a function called when a [connect](https://github.com/mqttjs/MQTT.js#event-connect) event is generated. Once a connection is established, [subscribe](https://github.com/mqttjs/MQTT.js#subscribe) to the appropriate IoT Topic stored in the subTopic variable (no need for any other parameters). Then call the readConsoleInput() function to start reading the input from the console indefinitely.

Solution for TODO 2.

### 2.4 Develop TODO 3

In TODO 3, implement a function called when a [message](https://github.com/aws/aws-iot-device-sdk-js#event-message) event is generated. When subscribed to a topic, in this case your own car's topic, the message event is emitted when a payload is received on that topic. Output the message received in the console.

Solution for TODO 3.

### 2.5 Develop TODO 4

In TODO 4, finish the implementation of the function publishToIoTTopic that takes 2 arguments: the name of the topic to use and the payload to send. Use the [publish](https://github.com/mqttjs/MQTT.js#publish) function of the device object you created in TODO 1 to publish to the other car's topic using the topic argument passed to the function. The message that was read from the console can be found in the second argument called payload.

Solution for TODO 4.

### 2.6 Full solution code (optional)

If you have developed your own code, you can use the solution code to validate what you have done or skip to the next section 2.7. If you decided not to code, then you will need to get the solution code here.

Expand to see the steps

### 2.7 Duplicate the code to car2

1. Save the file you were editing by clicking on **File > Save**.
2. As you only made modifications to car1/exercise-2.3.js, copy that file into car2's repository by executing the following commands:

cp ~/environment/car1/exercise-2.3.js ~/environment/car2

## 3. Run the code

In this section, you will run the code that you have just created in 2 different Cloud9 terminals and validate that you can pass messages between both side.

1. If you don't have **2 Cloud9 terminals**, click the **circled +** icon that is next to your current terminal and select **New Terminal**. You now have 2 different terminals.
2. In the **left terminal (car1)**, execute the following commands to start the car1 code.
3. cd ~/environment/car1

node exercise-2.3.js

You should see the following:

Enter a message on the next line to send to lab/messaging/car2:

1. In the **right terminal (car2)**, execute the following commands to start the code.
2. cd ~/environment/car2

node exercise-2.3.js

You should see the following:

Enter a message on the next line to send to lab/messaging/car1:

1. In the **left terminal (car1)**, type a message to be sent (for example: Welcome to the IoT class) and press the **enter** key.
2. In the **right terminal (car2)**, you should see the following output:

Message received on topic lab/messaging/car2: Welcome to the IoT class

1. In the **right terminal (car2)**, type a message to be sent back (for example: Thank you) and press the **enter** key.
2. In the **left terminal (car1)**, you should see the following output:

Message received on topic lab/messaging/car1: Thank you

## 4. Delete the resources created in this exercise

In this section, you will remove all the different resources created as part of this exercise that won't be required for the other exercises.

The resources from Exercise 1.1 will still be there and should remain in place.

### 4.1 Stop both Cars

Although the cars don't send any messages, they are connected to the IoT service and should be disconnected to prevent charges.

1. In the **left terminal (car1)**, press **Ctrl-c** twice and you should see the labIoTUser:~/environment/car1 $ prompt.
2. In the **right terminal (car2)**, press **Ctrl-c** twice and you should see the labIoTUser:~/environment/car2 $ prompt.

### 4.2 Stop the Cloud9 environment

The Cloud9 environment will automatically shut down after 30 minutes of inactivity. For your Cloud9 environment to be considered inactive, you need to close the browser tab. All of the settings will be saved.

1. Close the **browser tab** where your environment was running.

As the operating system is Amazon Linux, you are billed by the second during those 30 minutes of inactivity. If you are under the free tier, this would be covered. If you are no longer under the free tier, you can force a stop of the EC2 instance that runs your Cloud9 environment. This will have no effect on the future exercises.

1. In the AWS Management Console, click **Services**, and then click **EC2** to open the EC2 console.
2. Click **Instances** in the left menu.
3. Select the EC2 Instance that has a name that starts with **aws-cloud9**.
4. Click **Actions > Instance State > Stop instance**

Congratulations! You have successfully completed this exercise. No need to look at the Python code unless you want to do the same exercise again in Python. You can now move on to the next unit.

Expand me to code in Python.

## 1. Install the AWS IoT Device SDK for Python, download and open the code

In this section, you will install the AWS IoT Device SDK for Python. You will then download the piece of code that has already been created and from which you should continue the development.

### 1.1 Start Cloud9

Your Cloud9 environment may have shut down at this point as it's supposed to automatically shutdown after 30 minutes. To restart it, follow these steps:

1. In the AWS Management Console, click **Services**, and then click **Cloud9** to go to the Cloud9 console.
2. You should see a list of environments. If you don't, click on the hamburger menu icon (the three parallel lines) near the top left of the screen and click on **Your environments**.
3. Click the **Open IDE** button in the **IoTOnAWS** card.
4. It may take a minute for your environment to start.

### 1.2 Install the AWS IoT Device SDK for Python

1. In your AWS Cloud9 **terminal**, run the following command.

sudo pip install AWSIoTPythonSDK

### 1.3 Download and open the code

1. Download the application code in car1's folder by running the following commands in your AWS Cloud9 **terminal**:
2. cd ~/environment/car1

wget https://aws-tc-largeobjects.s3.amazonaws.com/OTP-AWS\_D5-2019/v1.0/code/exercise-2.3.py

1. In the left side of the Cloud9 environment, open the **car1/exercise-2.3.py** file by double-clicking on it.

## 2. Develop the application

In this section, you will look at the current state of the application by reading the current code. You will then develop each of the TODOs that were left inside of the code. The solution can be found in each of the sections below to validate your code. If you prefer to use the solution directly, you can read each of the sections below and go to Section 2.6 to download the solution code. **If you prefer not to code at all, skip to Section 2.6 Full solution code (optional).**

### 2.1 Current application state

* raw\_input is used to read the input from the Cloud9 terminal console already in an infinite loop so you don't need to implement that piece. Each time an input is read, it sends it to the publishToIoTTopic function which you will need to implement in TODO 4.
* AWSIoTMQTTClient has already been imported. You will need to use it for TODO 1.
* The host parameter to establish the connection to your AWS IoT endpoint is read from the endpoint.json file and stored in the host variable.
* The code has been built to look at the current folder's name and establish what the publish and subscribe topic is based on it. The subTopic variable contains the IoT Topic to subscribe to. The pubTopic variable contains the IoT Topic to publish to.

### 2.2 Develop TODO 1

1. In TODO 1, use AWSIoTMQTTClient to create a client supplying the clientId using TLSv1.2 Mutual Authentication. See the documentation [here](https://s3.amazonaws.com/aws-iot-device-sdk-python-docs/html/index.html#AWSIoTPythonSDK.MQTTLib.AWSIoTMQTTClient).
2. Use the configureEndpoint function to set the host and port. See the documentation [here](https://s3.amazonaws.com/aws-iot-device-sdk-python-docs/html/index.html#AWSIoTPythonSDK.MQTTLib.AWSIoTMQTTClient.configureEndpoint).
3. Use the configureCredentials function to set your credentials using the caPath, certPath and keyPath. See the documentation [here](https://s3.amazonaws.com/aws-iot-device-sdk-python-docs/html/index.html#AWSIoTPythonSDK.MQTTLib.AWSIoTMQTTClient.configureCredentials).
4. Connect to the AWS IoT service using the connect function. See the documentation [here](https://s3.amazonaws.com/aws-iot-device-sdk-python-docs/html/index.html#AWSIoTPythonSDK.MQTTLib.AWSIoTMQTTClient.connect).

Solution for TODO 1.

### 2.3 Develop TODO 2

In TODO 2, implement callback function that will be called when a new message is received on the subTopic taking client, userdata, message as parameters. This function will be used in TODO 3. You can look at the documentation for the [subscribe](https://s3.amazonaws.com/aws-iot-device-sdk-python-docs/html/index.html#AWSIoTPythonSDK.MQTTLib.AWSIoTMQTTClient.subscribe) function for some help.

In that function, print in the console the topic (message.topic) and payload (message.payload). Note that the message.payload is in bytes, so you will need to decode it before printing it.

Solution for TODO 2.

### 2.4 Develop TODO 3

In TODO 3, use the subscribe function to subscribe to the subTopic IoT Topic with a **qos** of **1** and using the function you created in TODO 2 as the callback. See the documentation [here](https://s3.amazonaws.com/aws-iot-device-sdk-python-docs/html/index.html#AWSIoTPythonSDK.MQTTLib.AWSIoTMQTTClient.subscribe).

Solution for TODO 3.

### 2.5 Develop TODO 4

In TODO 4, finish the implementation of the function publishToIoTTopic that takes 2 arguments. The name of the topic to use and the payload to send. Use the [publish](https://s3.amazonaws.com/aws-iot-device-sdk-python-docs/html/index.html#AWSIoTPythonSDK.MQTTLib.AWSIoTMQTTClient.publish) function to publish to the other car's topic using the topic argument passed to the function. The message that was read from the console can be found in the second argument called payload. Use a **qos** of **1**.

Solution for TODO 4.

### 2.6 Full solution code (optional)

If you have developed your own code, you can use the solution code to validate what you have done or skip to the next section 2.7. If you decided not to code, then you will need to get the solution code here.

Expand to see the steps

### 2.7 Duplicate the code to car2

1. Save the file you were editing by clicking on **File > Save**.
2. As you only made modifications to car1/exercise-2.3.py, copy that file into car2's repository by executing the following commands:

cp ~/environment/car1/exercise-2.3.py ~/environment/car2

## 3. Run the code

In this section, you will run the code that you have just created in 2 different Cloud9 terminals and validate that you can pass messages between both side.

1. If you don't have **2 Cloud9 terminals**, click the **circled +** icon that is next to your current terminal and select **New Terminal**. You now have 2 different terminals.
2. In the **left terminal (car1)**, execute the following commands to start the car1 code.
3. cd ~/environment/car1

python exercise-2.3.py

You should see the following:

Enter a message on the next line to send to lab/messaging/car2:

1. In the **right terminal (car2)**, execute the following commands to start the code.
2. cd ~/environment/car2

python exercise-2.3.py

You should see the following:

Enter a message on the next line to send to lab/messaging/car1:

1. In the **left terminal (car1)**, type a message to be sent (for example: Welcome to the IoT class) and press the **enter** key.
2. In the **right terminal (car2)**, you should see the following output:

Message received on topic lab/messaging/car2: Welcome to the IoT class

1. In the **right terminal (car2)**, type a message to be sent back (for example: Thank you) and press the **enter** key.
2. In the **left terminal (car1)**, you should see the following output:

Message received on topic lab/messaging/car1: Thank you

## 4. Delete the resources created in this exercise

In this section, you will remove all the different resources created as part of this exercise that won't be required for the other exercises.

The resources from Exercise 1.1 will still be there and should remain in place. If you would like to remove the resources from Exercise 1.1, refer to that exercise.

### 4.1 Stop both Cars

Although the cars don't send any messages, they are connected to the IoT service and should be disconnected to prevent charges.

1. In the **left terminal (car1)**, press **Ctrl-c** and you should see the labIoTUser:~/environment/car1 $ prompt.
2. In the **right terminal (car2)**, press **Ctrl-c** and you should see the labIoTUser:~/environment/car2 $ prompt.

### 4.2 Stop the Cloud9 environment

The Cloud9 environment will automatically shut down after 30 minutes of inactivity. For your Cloud9 environment to be considered inactive, you need to close the browser tab. All of the settings will be saved.

1. Close the **browser tab** where your environment was running.

As the operating system is Amazon Linux, you are billed by the second during those 30 minutes of inactivity. If you are under the free tier, this would be covered. If you are no longer under the free tier, you can force a stop of the EC2 instance that runs your Cloud9 environment. This will have no effect on the future exercises.

1. In the AWS Management Console, click **Services**, and then click **EC2** to open the EC2 console.
2. Click **Instances** in the left menu.
3. Select the EC2 Instance that has a name that starts with **aws-cloud9**.
4. Click **Actions > Instance State > Stop instance**

Congratulations! You have successfully completed this exercise. No need to look at the NodeJS code unless you want to do the same exercise again in NodeJS. You can now move on to the next unit.