Lab no 07 Part 01 –AWS IoT Shadows

The purpose of this lab is to be familiar with AWS shadows in IoT. In this lab, we will interact with the Thing Shadow of a car.

Parts: -

1. Download the new code and start car1.
2. Change the status of the lights while car 1 is connected.
3. Change the status of the lights while car 1 is disconnected.
4. Delete the resources created in this exercise.

Required Resources

- 1 PC with Internet access.
- Account in AWS Management Console.

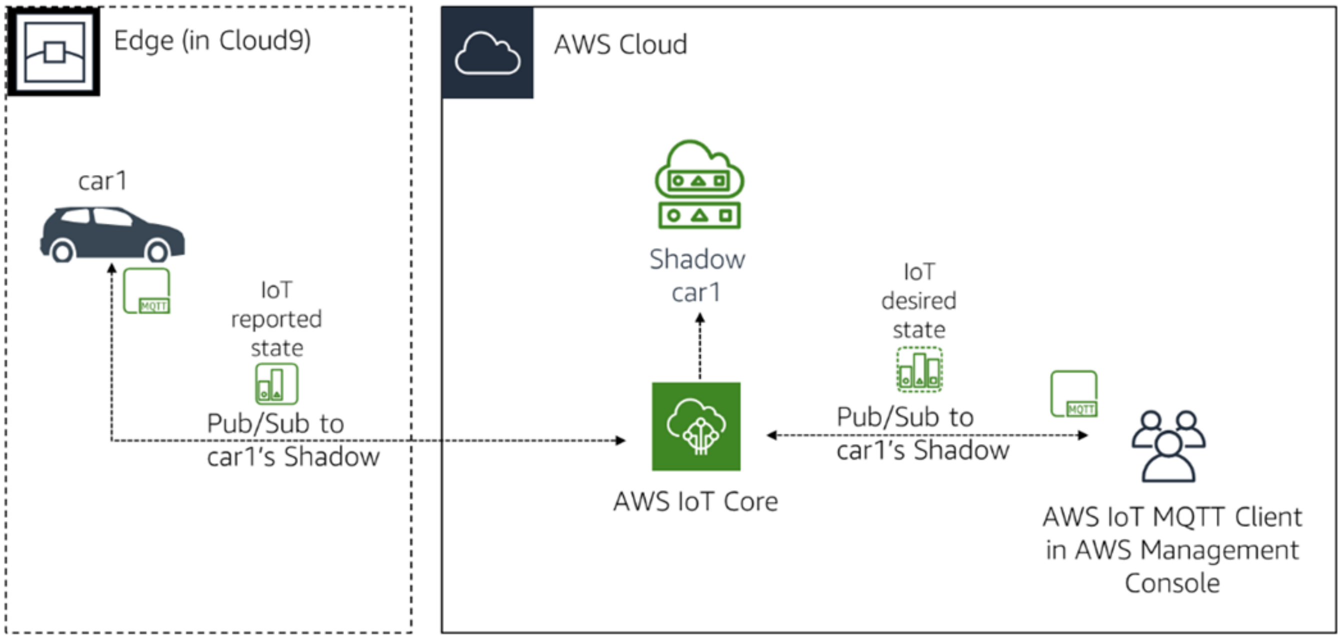
In this exercise, you will interact with the Thing Shadow of car1. The car has lights that can be turned on and off. You could directly run a server on car1 itself waiting for a REST API call, however that won't be optimized for battery. Instead, you will use AWS IoT as the channel for the interaction between the input of the lights to be turned on/off and the car.

You could use a similar IoT Topic like the one you used in lab 6 part 1, but the car would have to be connected at all times as messages in IoT are ephemeral. If the car wasn't connected at the time, you would send an input to change the light status, then the car would never know. Being connected always isn't great for a Thing that doesn't have much power or that wants to minimize power consumption.

Instead, you will use an IoT Thing Shadow. As you saw in the lecture, a Shadow represents the state of your car inside the IoT service. You can interact with the Shadow and when the car connects back to the Shadow, it will see that the state has changed and will act upon it. Thus, decoupling the front end (via a browser) and the back end (car1). It's still using IoT Topics for the interaction so any MQTT client would work to interact with that Thing Shadow.

You will first download the new code for car1 and start it. Then you will use the AWS IoT MQTT Client in the AWS Management Console to publish a new Desired State for the car turning its lights on/off. Finally, you will disconnect car1 to see what happens when you modify the State of the car and it's disconnected. Does it keep its status? What if the car reconnects after you modified the Thing Shadow? You will answer these questions in this exercise.

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



1. Download the new code and start car1

In this section, you will make sure that car1's Thing Shadow is empty in case you modified it during outside of the exercises. You will then connect to your Cloud9 environment, download the new code for car1 and start it. Since this will be the first time you start car1, it will set itself to its defaults of having its lights turned off and report that state to AWS IoT.

1.1 Start Cloud9

Your Cloud9 environment may have shut down at this point as it's supposed to automatically shut down 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. If you don't see it, make sure you are in the same **Region** as the one you used in lab 6 part 1. It should be **Frankfurt, Ireland, N. Virginia, Ohio, Oregon or Tokyo**.
4. It may take a minute for your environment to start.

1.2 Set the initial state of car1

1. In the AWS Management Console, click **Services**, and then click **IoT Core** to go to the IoT console.
2. Expand **Manage** and click **Things** in the left menu.
3. Click on **car1**.
4. Click on **Shadows**.
5. Click the **3 dots** next to **Classic Shadow**, select **Remove shadow** and click **Remove shadow(s)**.

This will ensure that your Shadow is completely empty before we start using it. It should say: *No shadow found for this thing*.

1.3 Download the new code and start car1

1. Download the application code in the car1 folder by running the following commands in your AWS Cloud9 **terminal**:

```
cd ~/environment/car1
wget https://aws-tc-largeobjects.s3.amazonaws.com/OTP-AWS_D5-2019/v1.0/code/exerc
ise-2.2.js
```

2. Start car1 by executing the following command.

```
node exercise-2.2.js
```

You should see the following if you didn't modify the Thing Shadow outside of the exercises.

```
The car1 has been registered.
Sending initial get to set the light state.
Received the initial get data.
No lights state found, setting state to defaults.
My lights are off
    ____
 __/    \__
|  _     _ `-.
'-(_)---(_)--'
```

Feel free to look at the code to understand what was done. In summary, the first step is to register to the Thing Shadow topics. Once registered, which also includes the connection to AWS IoT, an initial *get* request is sent. This isn't a blocking request, and it will call the *status* event when a reply is sent back by AWS IoT. Car1's Shadow should be empty as it has never been interacted with which is what you see as reported: *No lights state found*. An initial state of having its lights off is reported to AWS IoT. It then shows beautiful ASCII art of a car with its lights off.

1. In the AWS Management Console, click **Services**, and then click **IoT Core**.
2. Expand **Manage**, click **Things** in the left menu.
3. Click on **car1**.
4. Click on **Shadows** on the left side.

5. Click on **Classic Shadow**.
6. You should see that the *Shadow state* has been reported by the car with its lights off:

```
{
    "reported": {
        "lights": false
    }
}
```

2. Change the status of the lights while car1 is connected

In this section, you will connect to AWS IoT using the AWS IoT MQTT Client and send an update to turn on the lights of the car. You will look at different IoT Shadow Topics specific to car1 to understand what is going on with the code.

1. In the AWS Management Console, click **Services**, and then click **IoT Core** to go to the IoT console.
2. Click **Test** in the left menu. It will open an AWS IoT MQTT Client where you can interact with any Topic that you have access to. This Client will automatically connect to your IoT Endpoint.
3. Under **Subscribe topic**, enter $aws/things/car1/shadow/update/documents.
4. Click **Subscribe to topic**. You have now subscribed to car1's Thing Shadow documents topic. AWS IoT publishes a state document to this topic whenever an update to the shadow is successfully performed. It is useful for understanding the different states that are changing.
5. Click the **Subscribe to a topic** blue link on the left of this page's section.
6. Under **Subscribe topic**, enter $aws/things/car1/shadow/update/delta.
7. Click **Subscribe to topic**. You have now subscribed to car1's Thing Shadow delta topic. The Device Shadow service sends messages to this topic when a difference is detected between the reported and desired states of a shadow.
8. Click the **Publish to a topic** blue link on the left of this page's section. This will not close the subscriptions you made in the last few steps, it allows you to subscribe and publish to multiple topics.
9. Under the **Publish** section, in the white text box where **Specify a topic to publish to, e.g. myTopic/1** is written, enter $aws/things/car1/shadow/update. This will set the topic to car1's Thing Shadow special update topic. This is how you need to interact with the Thing Shadow if you use the topics directly.
10. Under the text box of the previous step where there is currently a "message": "Hello from AWS IoT console", replace the entire content with the following. This says that you have a desire for the lights to be set to true by setting the state to desired and passing all the attributes that you want to set.

```
{
    "state": {
        "desired": {
            "lights": true
        }
    }
}
```

11. Click Publish to Topic. What happens is that the desired state is sent to AWS IoT which then generates a delta state between the originally reported state from the previous section and the new desired state that you just sent.

12. Go back to the Cloud9 terminal and you should see a new beautiful ASCII art with lights turned on like the following:

```
My lights are on

    ____
 __/    \__
|  _     _ `-.///
'-(_)---(_)--'\\\
Reporting my new state.
```

What just happened is that the code is subscribed to the *delta* topic of car1's Thing Shadow. The code receives that *delta* using the *on('delta')* event. AWS IoT will only place the lights status in the delta if they are different than what was previously reported. If the code detects that the lights were modified, it will change its lights status to what is requested in the *delta*. Finally, the code will report back to AWS IoT using the *reported* status that it just turned on its lights. Feel free to look at the code to get a deeper understanding.

13. Back in the AWS IoT MQTT Client in the AWS Management Console, you will see that the **$aws/things/car1/shadow/update/documents** topic received an update as there is a green dot next to its name (it's the first one in the list). **Click** on that **topic**.

14. You will see that 2 messages have been received. Let's look at the oldest message at the bottom of the list. This message was generated when you published your desired state to the topic. It should look like the following:

```
{
    "previous": {
        "state": {
            "reported": {
                "lights": false
            }
        },
        "metadata": {
            "reported": {
                "lights": {
                    "timestamp": 1552943011
                }
            }
        },
        "version": 2
    },
    "current": {
        "state": {
            "desired": {
                "lights": true
            },
            "reported": {
                "lights": false
            }
        },
        "metadata": {
            "desired": {
                "lights": {
                    "timestamp": 1552943018
                }
            },
            "reported": {
                "lights": {
                    "timestamp": 1552943011
                }
            }
        },
        "version": 3
    },
    "timestamp": 1552943018
}
```

The JSON document will contain two primary nodes: previous and current. The *previous* node contains the content of the full shadow document before the update was performed while *current* contains the full shadow document after the update is successfully applied. The *previous* state shows that the previously *reported* state has the lights set to false. The *current* state represents the time you sent an update with a new *desired* state. You can see that the *desired* and *reported* state aren't the same. A delta state has been generated, but you can't see it here. You will look at it next.

15. Click on the **second topic** that you subscribed to: **$aws/things/car1/shadow/update/delta**. If you put your cursor on top of the topic's name, you will see its full name or you could click on the topic and it will be mentioned as the title.

16. You can see that a message was published to that *delta* topic. However, you never did that and neither did the code. That is done automatically by AWS IoT. It should look like the following:

```
{
    "version": 3,
    "timestamp": 1552943869,
    "state": {
        "lights": true
    },
    "metadata": {
        "lights": {
            "timestamp": 1552943869
        }
    }
}
```

You can see that the *state* is set directly to what is different between the *desired* and the *reported* states. In this case, *lights* is set to true. The code, which is subscribed to the *delta* topic, updated its status based on that delta and published a new *reported* state. You will see that next.

17. Click on the **first topic** that you subscribed to: **$aws/things/car1/shadow/update/documents**. This time take a look at the other message: the newest at the top of the list. It should look like the following:

```
{
    "previous": {
        "state": {
            "desired": {
                "lights":  true
            },
            "reported":  {
                "lights":  false
            }
        },
        "metadata":  {
            "desired":  {
                "lights":  {
                    "timestamp":  1552943018
                }
            },
            "reported":  {
                "lights":  {
                    "timestamp":  1552943011
                }
            }
        },
        "version":  3
    },
    "current":  {
        "state":  {
            "reported":  {
                "lights":  true
            }
        },
        "metadata":  {
            "reported":  {
                "lights":  {
                    "timestamp":  1552943018
                }
            }
        },
        "version":  4
    },
    "timestamp":  1552943018,
    "clientToken":  "car1-1"
}
```

The *previous* node shows the same state as the *current* node of the previous document which is expected with the differences between
the *desired* and *reported* states. The *current* node has now been updated with
a *reported* state set to what the previously *desired* state was at: lights set to true. That is normal as that is what you said you wanted by publishing that *desired* state in the beginning. The car1 code reported that new state once it received the *delta*.

3. Change the status of the lights while car1 is disconnected

In this section, you will update the light status of car1 via the AWS IoT MQTT Client while car1 is offline. To do this, you will stop car1, update the light status and start car1 again to see what happens.

1. In the Cloud9 **terminal**, press **Ctrl-c** to stop car1.
2. Back to the AWS IoT MQTT Client console, click the **Publish to a topic** blue link.
3. Under the **Publish** section, in the white text box $aws/things/car1/shadow/update should still be there. If not, enter it.
4. Under the previous step's text box, you should see a JSON payload, replace the **entire content** with the following. This says that you have a desire for the lights to set to false by setting the state to *desired* and passing all the attributes that you want to set.

```
{
    "state": {
        "desired": {
            "lights": false
        }
    }
}
```

5. Click **Publish to Topic**. What happens is that the desired state is sent to AWS IoT which then generates a *delta* state between the originally *reported* state from the previous section and the new *desired* state that you just sent.
6. You can look at both *documents* and *delta* topics that you subscribed to in the previous section where you will see a similar pattern. However, there will only be 1 new message in the *documents* Topic. The reason is because the device is currently offline. So, it can't act on the *delta* that was just published.
7. Open a **new** AWS Management Console tab as you want to conserve the MQTT Client you are currently in. You can do that by copying the URL of this tab, opening a new tab, and pasting that URL in. You won't have to login again.

8. Click **Services**, and then click **IoT Core** to go to the IoT console.
9. Expand **Manage** and click **Things** in the left menu.
10. Click on **car1**.
11. Click on **Shadows**.
12. Click on **Classic Shadow**.
13. You should see a **Shadow state** like the following:

```
{
    "desired": {
        "lights": false
    },
    "reported": {
        "lights": true
    },
    "delta": {
        "lights": false
    }
}
```

As you can see, the Shadow has a *desired* state set to what you just published. It has a *reported* state set to what the last reported state was. And it has a *delta* state showing the difference between those two states.

14. You will now start car1 again. The first step that the car1's code takes when it wakes up is to issue a *get* request. On the response to that request, it looks for a *delta* state. If there is a *delta*, it acts on it by setting its lights to that state. Thus, it doesn't need to go back for every single message that were sent. In fact, those messages aren't even there anymore. We know there was only one, but what if you modified the lights 5 more times while it was offline? It only needs to look at the *delta*.
15. In the Cloud9 **terminal**, start car1 again by entering the following command:

```
node exercise-2.2.js
```

You should see the following output:

```
The car1 has been registered.
Sending initial get to set the light state.
Received the initial get data.
Delta found on initial get setting lights to that state and reporting.
My lights are off
    ____
 __/    \__
|  _     _ `-.
'-(_)---(_)--'
```

It looks very similar to the first time you started car1, however this time the line *Delta found on initial get setting lights to that state and reporting* is present. This indicates that while looking at the initial get data, it found a *delta* state and acted on it.

16. Feel free to look at the code and to play with the status of the lights by publishing more updates on the *update* Shadow Topic. You can also do the same with the car2.

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 lab 6 part 1 will still be there and should remain in place. If you would like to remove the resources from lab 6 part 1, refer to that exercise.

4.1 Stop car1

Although car1 doesn't send any messages, it is connected to the IoT service and should be disconnected to prevent charges.

1. **Press Ctrl-c** in the Cloud9 **terminal** to stop car1 from interacting with AWS IoT.

4.2 Stop the MQTT Client

1. **Navigate away** from the **AWS IoT MQTT Client** page to disconnect from the client.

4.3 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 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** on the left menu.
3. Select the EC2 Instance that has a name that starts with **aws-cloud9**.
4. Click **Actions > Instance State > Stop instance**

Note:
Labs are from Course: AWS IoT: Developing and Deploying an Internet of Things
https://www.edx.org/course/aws-iot-developing-and-deploying-an-internet-of-th