
Lab no 07 Part02 –Interacting with Car Shadow over the Web

The purpose of this lab is to be familiar with AWS shadows in IoT. In this lab, we will use a browser to interact with car1's Thing Shadow

Parts: -

1. Create the Cognito Identity Pool.
2. Modify the unauthenticated IAM Role.
3. Create the S3 Bucket with permission.
4. Download the webpage code and upload it to S3.
5. Start car1 and use the webpage application.
6. Delete the resources created in this exercise.

Required Resources

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

In this exercise, you will use a browser to interact with car1's Thing Shadow.

Up until now, you have only used the MQTT protocol directly to interact with your AWS IoT Endpoint. This used port 8443 and can only authenticate using X.509 client certificates. That is why you had to configure those for each of the Car Things. You can see the list of supported protocols in the documentation with the supported authentication protocol.

In a browser, although it could be possible to store the certificate and private key of your Car Thing in your JavaScript code, that wouldn't be very secure. Also, using MQTT in a browser isn't straight forward. As it's a browser, the favorite protocol is HTTPS for communication outbound. Although it is supported by the AWS IoT service, it has been proven that the HTTPS protocol doesn't have the same performance as the MQTT protocol when many messages are sent. That is due to the overhead of HTTPS as the connection needs to be re-established for every message. As opposed to MQTT where one connection can be used for all messages to be sent. Unfortunately, using MQTT directly in the browser isn't an option. Instead, MQTT over a WebSocket is a lot easier to do. Although there is an overhead of connection establishment with a WebSocket, we will gain back on the benefit of the small overhead of MQTT inside the WebSocket.

The MQTT over WebSocket protocol only supports the Signature version 4 (SigV4) authentication as you saw in the documentation. To do so, the wss://endpoint URL will need to be signed using an Access Key and a Secret Key. You could store those in your JavaScript code, but that wouldn't be very secure. Instead, you need to generate a temporary Access Key and Secret Key when the user opens the page or authenticate. The service to help with that is Cognito Identity Pool (Federated Identities).

One of the features of Cognito Identity Pools is that you can specify an IAM Role for unauthenticated users. This means that even if your user doesn't authenticate, you can still provide credentials and permissions to that user. You will be using this feature in the lab as authentication in Cognito Identity Pools is beyond the scope of the class.

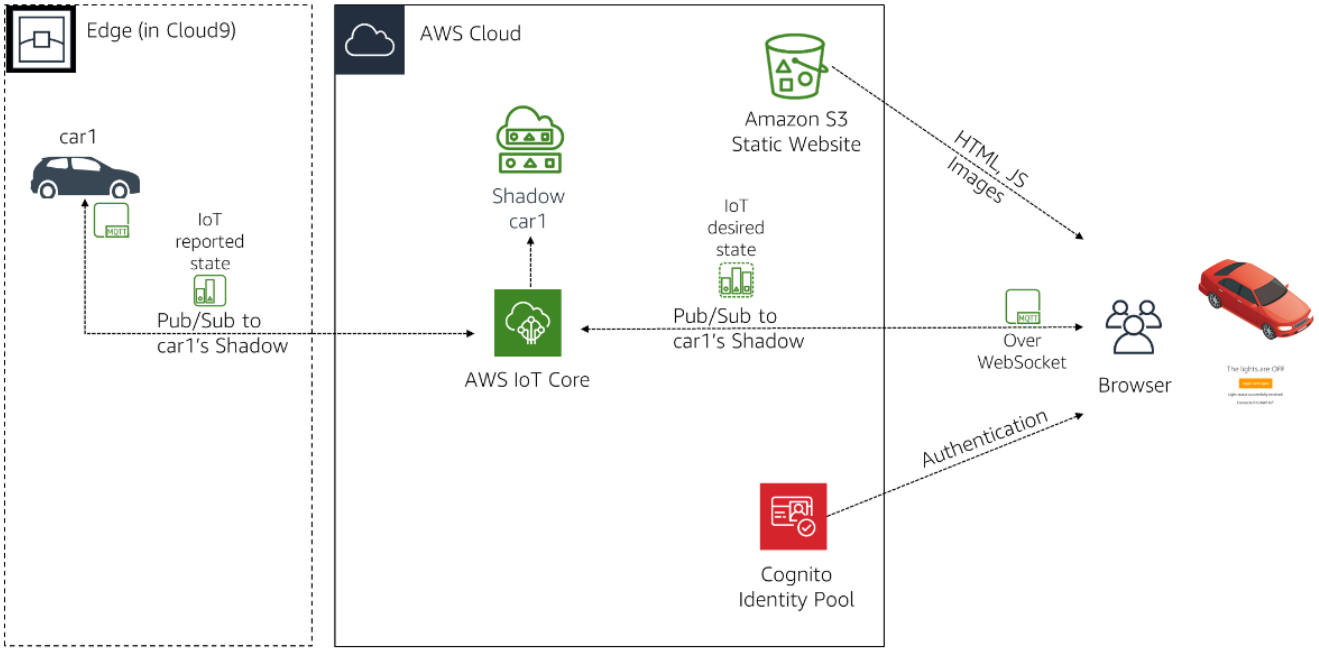
You will start with creating a Cognito Identity Pool. That will create 2 IAM Roles: authenticated and unauthenticated. You will then modify the unauthenticated role to attach an IAM Policy that will allow the Role to communicate with the Thing Shadow of car1.

You will use the Simple Storage Service (S3) to host this static website. That website will make use of JavaScript to communicate with your Cognito Identity Pool to get temporary credentials. Those credentials will then be used to authenticate the JavaScript MQTT WebSocket client to your AWS IoT Endpoint.

Once the website is ready, you will start car1 using the code from Exercise 2.2 and browse to the website you created. You will then be able to interact with car1's Thing Shadow without requiring using the AWS Management Console.

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

Lab no 07 part2 –Interacting with Car1's Shadow over the Web



This exercise assumes that the resources from lab6 and the code downloaded in lab 6 part1 haven't been deleted. If you have deleted those components, you will need to start with lab 6 part 1 again and Section 1 of lab 7 part 1 to get the code for interacting with car1's Thing Shadow.

1. Create the Cognito Identity Pool

In this section, you will create a Cognito Identity Pool which will also create 2 IAM Roles.

1. In the AWS Management Console, click **Services**, and then click **Cognito** to go to the Cognito console.
2. Make sure you are in the same **Region** as the one you used in lab 6. It should be **Frankfurt, Ireland, N. Virginia, Ohio, Oregon or Tokyo**. You can validate that by going to the Cloud9 service and looking for the IoTOnAWS environment. If you don't see it, you aren't in the right region.
3. Click **Manage Identity Pools**. It will automatically start the screen of creation of an Identity Pool as this is your first. If it's not, click the **Create new identity pool** button.
4. In the **Identity pool name** field, enter labIoTPool.
5. Click the **check mark** next to **Enable access to unauthenticated identities** as this is what you will be using to authenticate the web application.
6. Click **Create Pool**.
7. The next screen may take 30 seconds to appear. It will tell you that Cognito needs access to your resources. It needs to create the 2 IAM Roles: an authenticated Role and an unauthenticated Role.
8. Expand the **View Details** link to see the name of the roles that will be created. You can also see the IAM Policy that will be attached by clicking on the *View Policy Document*.
9. Click **Allow**.

You have now created your Cognito Identity Pool.

2. Modify the unauthenticated IAM Role

In this section, you will modify the unauthenticated IAM Role that was created as part of the Cognito Identity Pool creation to authorize, whoever authenticates with this role, the access to car1's Thing Shadow. To remove complexity, you will use an IAM Policy that already exists which is more permissive. However, in a production environment, you should use the documentation to modify the *Resource* field to be specific to car1.

1. In the AWS Management Console, click **Services**, and then click **IAM** to go to the IAM console.
2. Click **Roles**.
3. In the **Search** bar, enter Cognito_labIoTPoolUnauth_Role.
4. Click on the **Cognito_labIoTPoolUnauth_Role** link.
5. Click **Attach policies**.
6. In the **Search** bar, enter AWSIoTDataAccess.
7. Click the **check mark** next to **AWSIoTDataAccess**.
8. Click **Attach policy**.

3. Create the S3 Bucket with permissions

In this section, you will create a new S3 Bucket and allow access to upload objects in the bucket with a Public ACL. This will allow you to upload and browse to your objects as if it was a website without authenticating in a later section.

1. In the AWS Management Console, click **Services**, and then click **S3** to go to the S3 console.
2. Click the **Block public access (account settings)** link on the left menu.
3. Click the **Edit** button.
4. **Take a note** of the current settings as you will need to put them back to what they were in the cleanup section of this exercise.
5. Uncheck **Block public access to buckets and objects granted through new access control lists (ACLs)**.
6. Uncheck **Block public access to buckets and objects granted through any access control lists (ACLs)**.
7. Click **Save changes**.
8. Enter confirm in the field.
9. Click **Confirm**.

10. Click the **Buckets** link on the left menu.
11. Click **Create bucket**.
12. For the **Bucket name** field, enter the name of your choice. For example, this could be labiot-123456.
13. **Take in note** that **name** as you will use it again.
14. Click **Next**.
15. Click **Next** in the Configure options screen.
16. Check **enable ACL.**
17. Uncheck **Block public access to buckets and objects granted through new access control lists (ACLs)**.
18. Uncheck **Block public access to buckets and objects granted through any access control lists (ACLs)**.
19. In the warning box that just appeared above, check **I acknowledge that the current settings might result in this bucket and the objects within becoming public**.
20. Click **Next**.
21. Click **Create bucket**.

4. Download the webpage code and upload it to S3

In this section, you will download the webpage code and modify the *aws-configuration.js* configuration file to match your account. Then you will upload the code to S3.

4.1 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.
4. It may take a minute for your environment to start.

4.2 Download the webpage code

1. Create a new working folder, download the webpage code in that folder and unzip it by running the following commands in your AWS Cloud9 **terminal**:

```
mkdir ~/environment/exercise-3.1
```

```
cd ~/environment
```

```
wget https://aws-tc-largeobjects.s3.amazonaws.com/OTP-AWS_D5-2019/v1.0/code/exercise-3.1.zip
```

```
unzip exercise-3.1.zip -d ~/environment/exercise-3.1
```

4.3 Modify the configuration file for your environment

In this section, you will use different commands to edit the *exercise-3.1/aws-configuration.js* file. It's a configuration file that contains different elements that points to your resources that will be used when you load the website via your browser. The first one is the ID of the Cognito Identity Pool that you created. The second property is your AWS IoT Endpoint that you have been using in each of the exercises. The last one is the region you are operating in.

1. Insert the Identity Pool ID of your Cognito Identity Pool in the configuration file. Run the following command in the Cloud9 **terminal** to put the Identity Pool ID in a variable and replace it in the configuration file:

```
cd ~/environment/exercise-3.1
```

```
poolid=$(aws cognito-identity list-identity-pools --max-results 60 | grep -B1 labIoPool | grep -Po '"IdentityPoolId"\s*:\s*"\K([^"]*)')
```

```
sed -i "s/xpoolidx/$poolid/" aws-configuration.js
```

2. Insert your AWS IoT Endpoint in the configuration file. Run the following command in the Cloud9 **terminal** to put your AWS IoT Endpoint in a variable and replace it in the configuration file:

```
cd ~/environment/exercise-3.1
```

```
endpoint=$(aws iot describe-endpoint --endpoint-type iot:Data-ATS | grep -Po '"endpointAddress"\s*:\s*"\K([^"]*)')
```

```
sed -i "s/xendpointx/$endpoint/" aws-configuration.js
```

3. Insert the region code in the configuration file. Run the following command in the Cloud9 **terminal** to put the region code in a variable and replace it in the configuration file:

```
cd ~/environment/exercise-3.1
```

```
region=$(aws configure get region)
```

```
sed -i "s/xregionx/$region/" aws-configuration.js
```

4. Feel free to look at the configuration file. In the left side of the Cloud9 environment, open the **exercise-3.1/aws-configuration.js** file by double-clicking on it.

4.4 Upload the Website code to S3

1. In the AWS Cloud9 **terminal**, enter the following commands to upload the code to your S3 Bucket. **Replace bucketname_changeme** with your S3 Bucket name that you created in section 3. It will upload the code with the *public-read* ACL which allows anyone to access those objects if they have the URL. Since this is a development environment, it's not an issue. However, if this was your production environment, you should consider using CloudFront in front of your S3 Bucket and lock access only from CloudFront. That is beyond the scope of this exercise.

```
cd ~/environment/exercise-3.1
```

```
aws s3 cp --recursive --acl public-read . s3://bucketname_changeme
```

You should see an output like the following:

```
upload: images/car-no-lights.png to s3://bucketname_changeme/images/car-no-lights.png
```

```
upload: images/car-with-lights.png to s3://bucketname_changeme/images/car-with-lights.png
```

```
upload: ./aws-configuration.js to s3://bucketname_changeme/aws-configuration.
js

upload: ./index.html to s3://bucketname_changeme/index.html

upload: ./index.js to s3://bucketname_changeme/index.js

upload: ./aws-iot-sdk-browser-bundle-min.js to s3://bucketname_changeme/aws-
iot-sdk-browser-bundle-min.js
```

5. Start car1 and use the webpage application

In this section, you will start car1 using the same code as Exercise 2.2. You will then browse to your web application and interact with car1's Thing Shadow.

1. In the AWS Cloud9 **terminal**, enter the following commands to start car1.

```
cd ~/environment/car1

node exercise-2.2.js
```

You should see the following. You may see a different status for the lights, but that isn't an issue. You can continue with the next step.

```
The car1 has been registered.
Sending initial get to set the light state.
Received the initial get data.
Found a previously reported state, setting my lights to that
My lights are off
    ____
 __/    \__
|  _____  `-.
'-(_)---(_)--'
```

   Leave car1 running.

2. In the AWS Management Console, click **Services**, and then click **S3** to go to the S3 console.
3. Click the **name of the S3 Bucket** that you created. You should see a list of objects.
4. Click the **check mark** next to **index.html**. An overlay window will appear from the right.
5. Click on the **Object URL** link. This will open the web application.

6. You can see the same lights status as you saw earlier when you started car1. You should see a car image with its lights turned off (or on depending on what your lights state was).
7. Click **Toggle car1's lights**.
    - You will see that the lights of the car have changed.
    - You can also see in the Cloud9 terminal that the lights have changed as well.
    - You shouldn't see any delay between the time that you pressed the button and that the lights changed in the web page. You may think that it's normal as the button just changed the car image, but what happened is a bit more complex. What just happened is that a *desired* state was published to the AWS IoT service by the web application (JavaScript in your browser). A *delta* was created by the AWS IoT service and the car1 application running in Cloud9 received the *delta*, changed its lights status and published a *reported* state. The web application running in your browser received the *reported* state and changed the text and car image. All of that within a split second.

The web application is running as JavaScript in your browser. Feel free to review the code of the application. You can view it in AWS Cloud9 under *index.js*. That code has not been packaged (webpack, browserify, uglify or others) on purpose so that you can read it. In summary, the code first gets new credentials from your Cognito Identity Pool using the AWS SDK. It then uses the AWS IoT Device SDK to create a Thing Shadow very similar to the code in lab 7 part 1. Once it registers to the Thing Shadow topics, it sends a *get* request to receive the currently *reported* state and shows it in the browser. When the *Toggle car1's lights* button is pressed, it uses the *update* function to send a *desired* state to AWS IoT. The *delta* is generated by AWS IoT, sent to car1 which changes the state of its lights and publishes a new *reported* state. This state generates a *foreignStateChange* event that triggers the function to change the car image and text.

You have now successfully interacted with a Thing Shadow from a browser using the MQTT over WebSocket protocol authenticating via the SigV4 protocol by using temporary credentials received from your Cognito Identity Pool.

6. 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.

6.1 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.

6.2 Delete the S3 Bucket

1. In the AWS Management Console, click **Services**, and then click **S3** to go to the S3 console.
2. Click the **check mark** next to the name of your bucket.
3. Click **Delete**.
4. In the field, enter the name of your bucket.
5. Click **Confirm**.

6.3 Undo the changes to the public access settings for S3

If you didn't have to change any settings for the **Block public access (account settings)** in section 3, you can skip this step. If you did have to change those settings, edit the settings below in accordance to what they were.

1. In the AWS Management Console, click **Services**, and then click **S3** to go to the S3 console.
2. Click the **Block public access (account settings)** link on the left menu.
3. Click the **Edit** button.
4. Click the check mark next to **Block public access to buckets and objects granted through new access control lists (ACLs)** if that setting was checked before.

5. Click the check mark next to **Block public access to buckets and objects granted through any access control lists (ACLs)** if that setting was checked before.
6. Click **Save**.
7. Enter confirm in the field.
8. Click **Confirm**.

6.4 Delete the Cognito Identity Pool

1. In the AWS Management Console, click **Services**, and then click **Cognito** to go to the Cognito console.
2. Click **Manage Identity Pools**.
3. Click the **labIoTPool** link.
4. Click the **Edit identity pool** link at the top right of the page.
5. Expand **Delete identity pool**.
6. Click the **Delete identity pool** button.
7. Click **Delete pool**.

6.5 Delete the IAM Roles

1. In the AWS Management Console, click **Services**, and then click **IAM** to go to the IAM console.
2. Click **Roles** in the left menu.
3. In the **Search** field, enter Cognito_labIoTPool.
4. Click the **check mark** next to **Cognito_labIoTPoolAuth_Role**
5. Click the **check mark** next to **Cognito_labIoTPoolUnauth_Role**
6. Click **Delete role**.
7. Click **Yes, delete**.

6.6 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 labs.

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**

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