Laboratorio

# AWS IoT Core

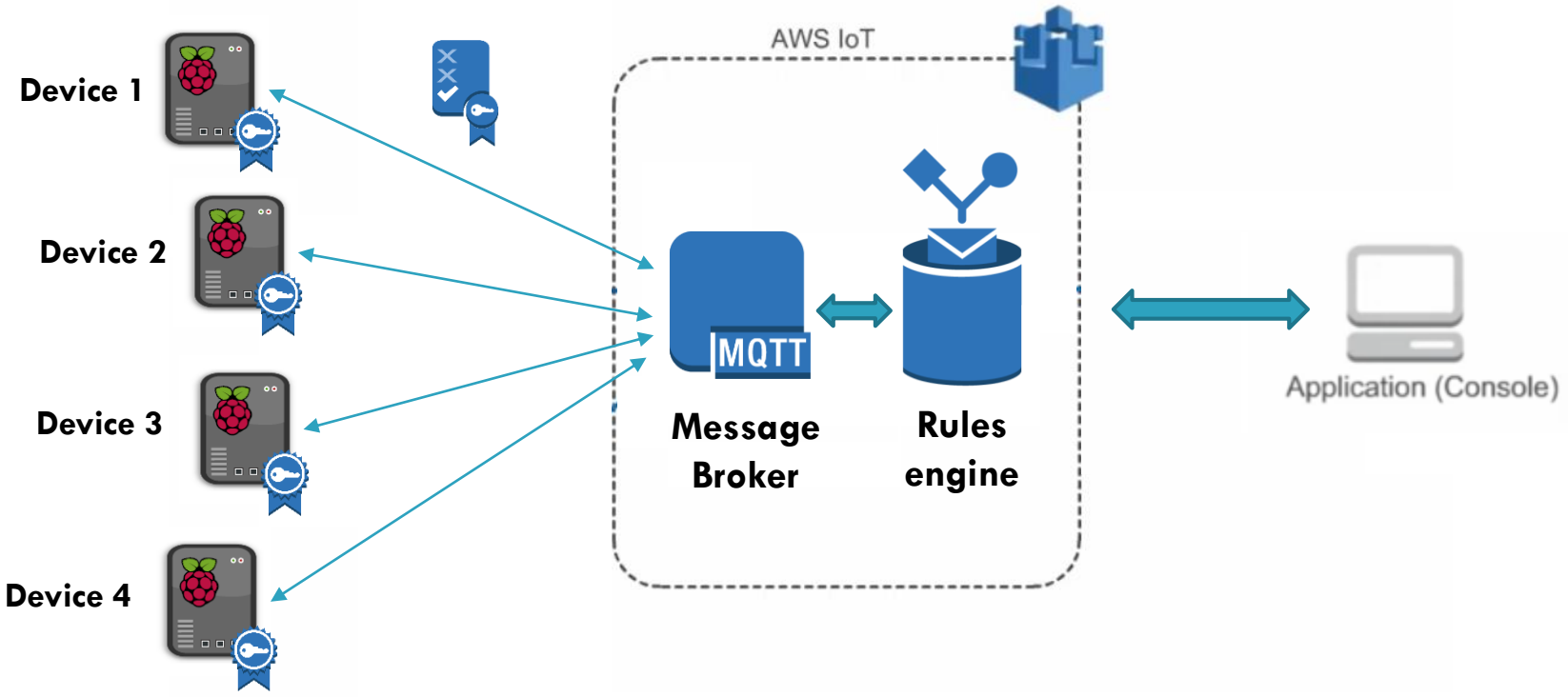
Corso di Smart City e Tecnologie Mobili

Università di Bologna

Dipartimento di Informatica — Scienza e Ingegneria

Luca Calderoni

# Architettura della soluzione cloud



# Prima di iniziare …

Scaricare il materiale di supporto fornito nel sito del corso. Caricare poi sul dispositivo l'archivio compresso (relativo alla versione studente) denominato *es1* al seguente percorso:

*home/pi/aws-iot-device-sdk-embedded-C/samples/linux/*

Estrarre il contenuto dell'archivio compresso. Devono essere presenti i seguenti file:

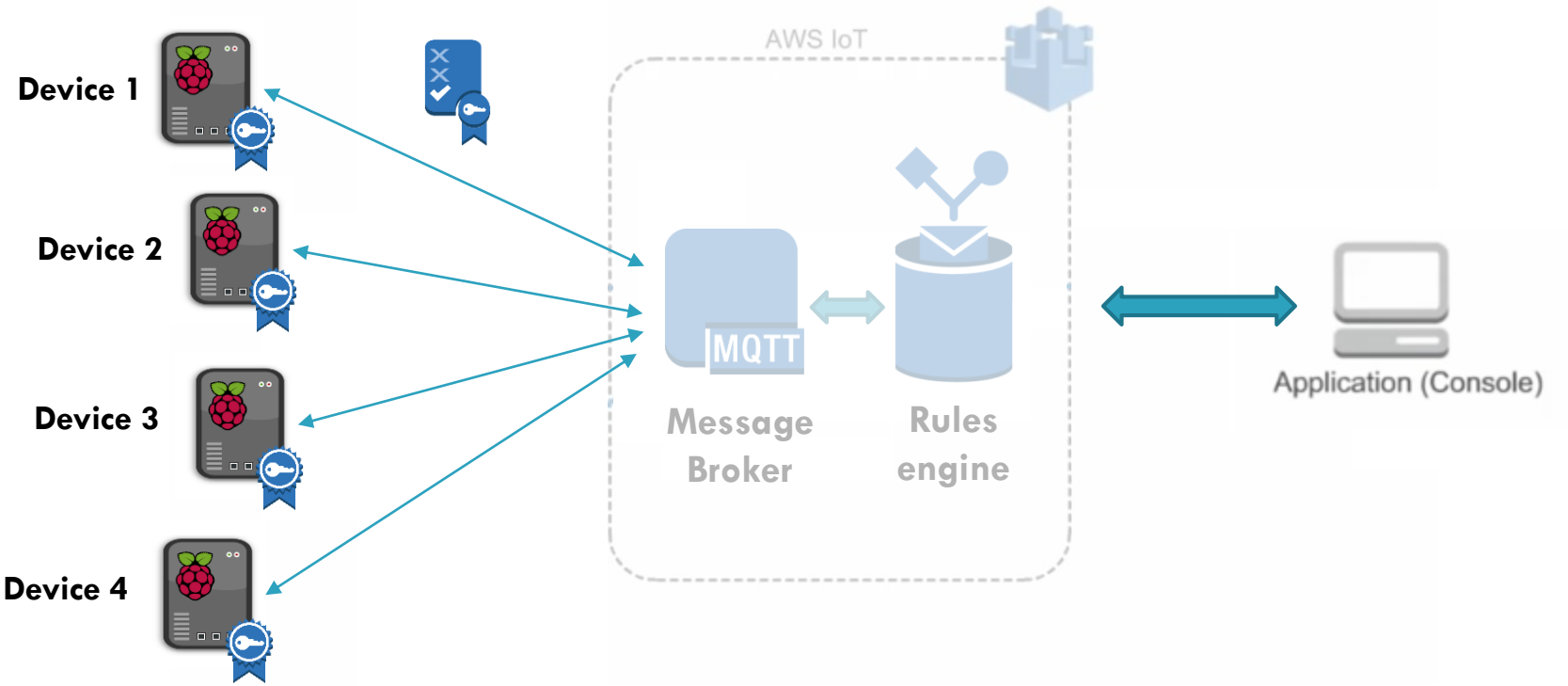
- *aws_iot_config.h*
- *aws_iot_core_lab.c*
- *Makefile*

Per generare il file eseguibile che utilizzeremo durante il corso dell'esercitazione, posizionarsi all'interno della directory e utilizzare il comando:

*make –f Makefile*

# Primo obiettivo

Realizzare la rappresentazione digitale di un device sul middleware, eventualmente assegnandolo a un tipo e a un gruppo. Connettere i device al Message Broker installando i certificati digitali e le coppie di chiavi fornite come materiale di supporto. Ad intervento terminato, i device devono comunicare con il Broker (MQTT *CONNECT*) mediante l'esecuzione del file eseguibile *aws_iot_core_lab*.



# Note per l'implementazione (1)

- Visualizzare un **gruppo**.
- Visualizzare un **tipo**.
- Visualizzare un **oggetto**.
  - Visualizzare il certificato digitale collegato all'oggetto.
  - Visualizzare la policy di comunicazione collegata al certificato.

I certificati digitali e le chiavi crittografiche devono essere prelevate dal materiale didattico. Per ogni device, sono disponibili:

- Chiave pubblica
- Chiave privata
- Certificato X.509 rilasciato da AWS

Inoltre, è presente un **Certificato radice di AWS** (trust point), comune a tutti i device.

# Note per l'implementazione (2)

I certificati digitali e le chiavi crittografiche devono essere inserite nella cartella:

*home/pi/aws-iot-device-sdk-embedded-C/certs'*

Le informazioni riguardanti i certificati e gli altri parametri di connessione devono essere inserite nel file:

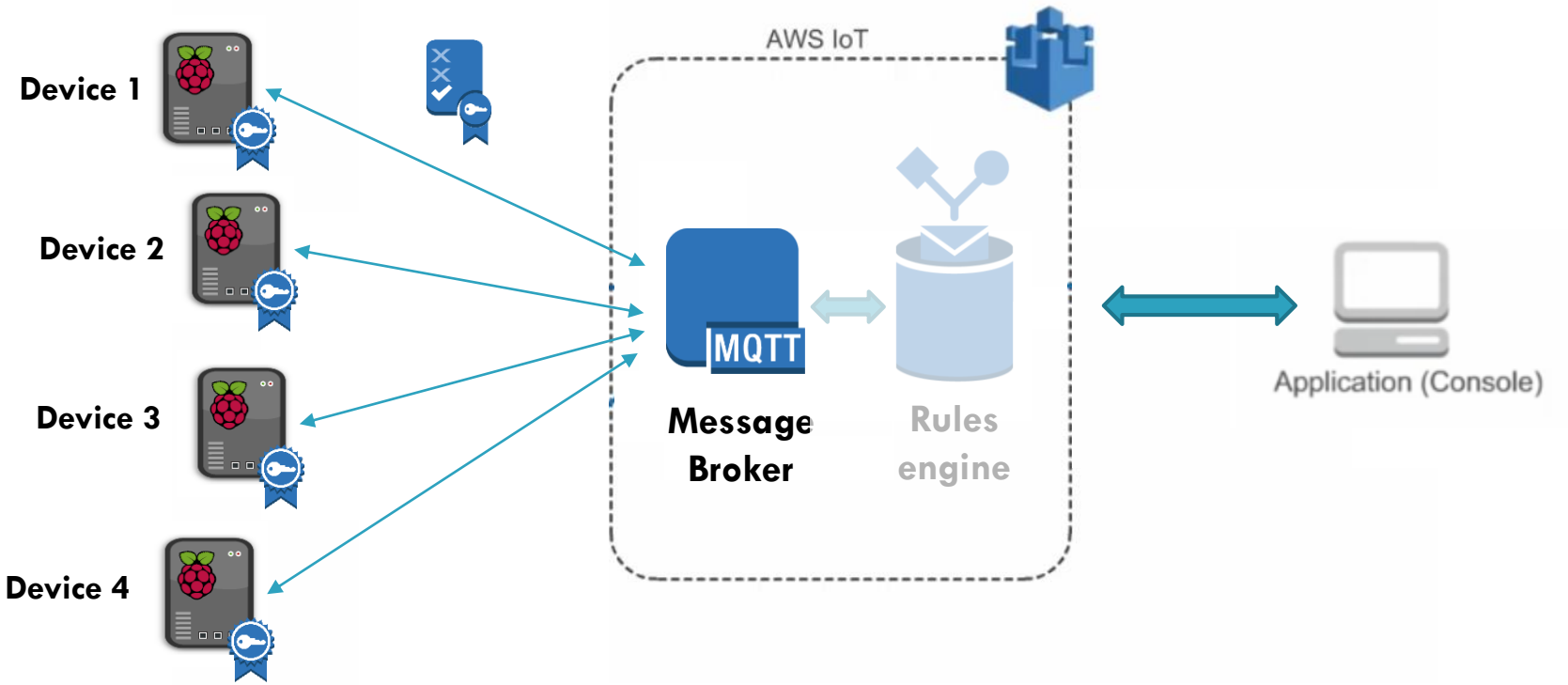
*home/pi/aws-iot-device-sdk-embedded-C/samples/linux/es1/aws_iot_config.h'*

L'identificativo univoco di ogni dispositivo deve coincidere con il nome della rappresentazione digitale del dispositivo presente nel back-end *(Device1, Device2, …)*.

```
// ================================================================================
#define AWS_IOT_MQTT_HOST             "a30mv68jy0zfa-ats.iot.eu-west-1.amazonaws.com"
                                      ///< Customer specific MQTT HOST (endpoint).
#define AWS_IOT_MQTT_PORT             443 ///< port for MQTT/S (8883 or 443)
#define AWS_IOT_MQTT_CLIENT_ID        "" ///< MQTT client ID (unique)
#define AWS_IOT_MY_THING_NAME         "" ///< Thing Name
#define AWS_IOT_ROOT_CA_FILENAME      "aaa.crt" ///< Root CA file name
#define AWS_IOT_CERTIFICATE_FILENAME  "bbb.pem" ///< device signed certificate file name
#define AWS_IOT_PRIVATE_KEY_FILENAME  "privkey.pem" ///< device private key filename
// ================================================================================
```

# Secondo obiettivo

Modificare il programma *aws_iot_core_lab* in modo che il device invii ogni 15 secondi un messaggio *PUBLISH* al Message Broker sul topic **SCTM/AWSLAB/temperatura**. Il messaggio deve contenere i dati *deviceId* (string) e *temperatura* (int). Se inviati correttamente, i messaggi saranno visualizzati nella web interface. Si utilizzi *Quality of Service* **at most once** (QoS = 0).



# Note per l'implementazione (3)

Nel SDK per embedded C possiamo avvalerci della funzione seguente per l'invio di messaggi *PUBLISH*.

```
//Pubblica un messaggio MQTT su un determinato topic
IoT_Error_t aws_iot_mqtt_publish(

            AWS_IoT_Client *                    pClient,
            const char *                        pTopicName,
            uint16_t                            topicNameLen,
            IoT_Publish_Message_Params *        pParams

)
```

La chiamata è bloccante. In caso di QoS 0 la funzione ritorna nonappena il pacchetto viene trasferito al layer TLS. In caso di QoS 1, la funzione ritorna dopo aver ricevuto il pacchetto *PUBACK* relativo.

| | |
|---|---|
| *pClient* | Riferimento al *AWS_IoT_Client* |
| *pTopicName* | Nome del topic sul quale pubblicare |
| *topicNameLen* | Lunghezza del nome del topic |
| *pParams* | Puntatore a una struttura dati *IoT_Publish_Message_Params* |

# Note per l'implementazione (4)

La struttura pParams è già creata e parzialmente istanziata nello scheletro dell'applicazione. Prima dell'invio del pacchetto, è necessario modificare il payload (mediante *sprintf*) e specificarne la lunghezza (*paramsQOS0.payloadLen* = …).

La temperatura può essere simulata generando un intero compreso tra 0 e 39 ad ogni ciclo di invio dati.
Il payload del messaggio publish **deve essere codificato in JSON** e, se formattato correttamente, verrà ricevuto dal Message Broker come esemplificato nell'immagine sottostante.

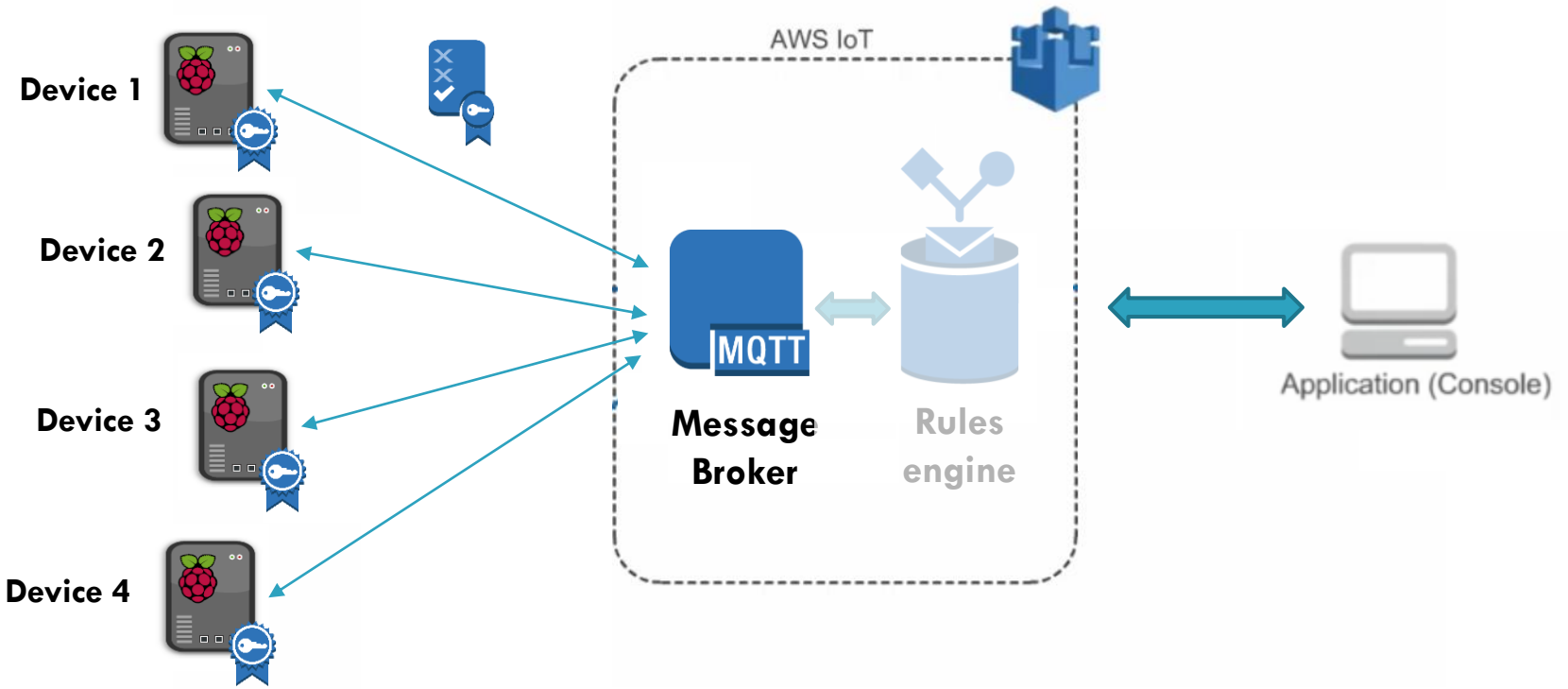
SCTM/AWSLAB/temperatura

```
{
  "deviceId": "Device1",
  "temperatura": 23
}
```

Nota: la costruzione del payload all'interno delle apposite funzioni C deve essere fatta eseguendo l'escape dei doppi apici.

# Terzo obiettivo

Modificare il programma *aws_iot_core_lab* in modo che il device si registri, mediante messaggio *SUBSCRIBE*, al medesimo topic sul quale vengono pubblicate le temperature (**SCTM/AWSLAB/temperatura**). Gestire inoltre la *callback function* che sarà invocata ogni volta che il device riceve un messaggio su uno dei topic al quale si è registrato. La callback deve prevedere la stampa a video dei messaggi ricevuti.



# Note per l'implementazione (5)

Nel SDK per embedded C possiamo avvalerci della funzione seguente per l'invio di messaggi *SUBSCRIBE*.

```
//Effettua la sottoscrizione ad un determinato topic
IoT_Error_t aws_iot_mqtt_subscribe(

            AWS_IoT_Client *        pClient,
            const char *            pTopicName,
            uint16_t                topicNameLen,
            QoS                     qos,
            pApplicationHandler_t   pApplicationHandler,
            void *                  pApplicationHandlerData

)
```

La chiamata è bloccante e ritorna dopo aver ricevuto il pacchetto *SUBACK* relativo.

| | |
|---|---|
| *pClient* | Riferimento al *AWS_IoT_Client* |
| *pTopicName* | Nome del topic al quale sottoscriversi |
| *topicNameLen* | Lunghezza del nome del topic |
| *pApplicationHandler_t* | Riferimento alla *callback function* da eseguire a seguito della ricezione di un pacchetto MQTT sul topic in oggetto |

# Note per l'implementazione (6)

La struttura *pClient* è già creata e istanziata nello scheletro dell'applicazione.
La *callback function* per gestire la ricezione dei pacchetti sul topic al quale ci si è sottoscritti va invece implementata, benché lo scheletro sia già presente nella versione del codice rilasciata.

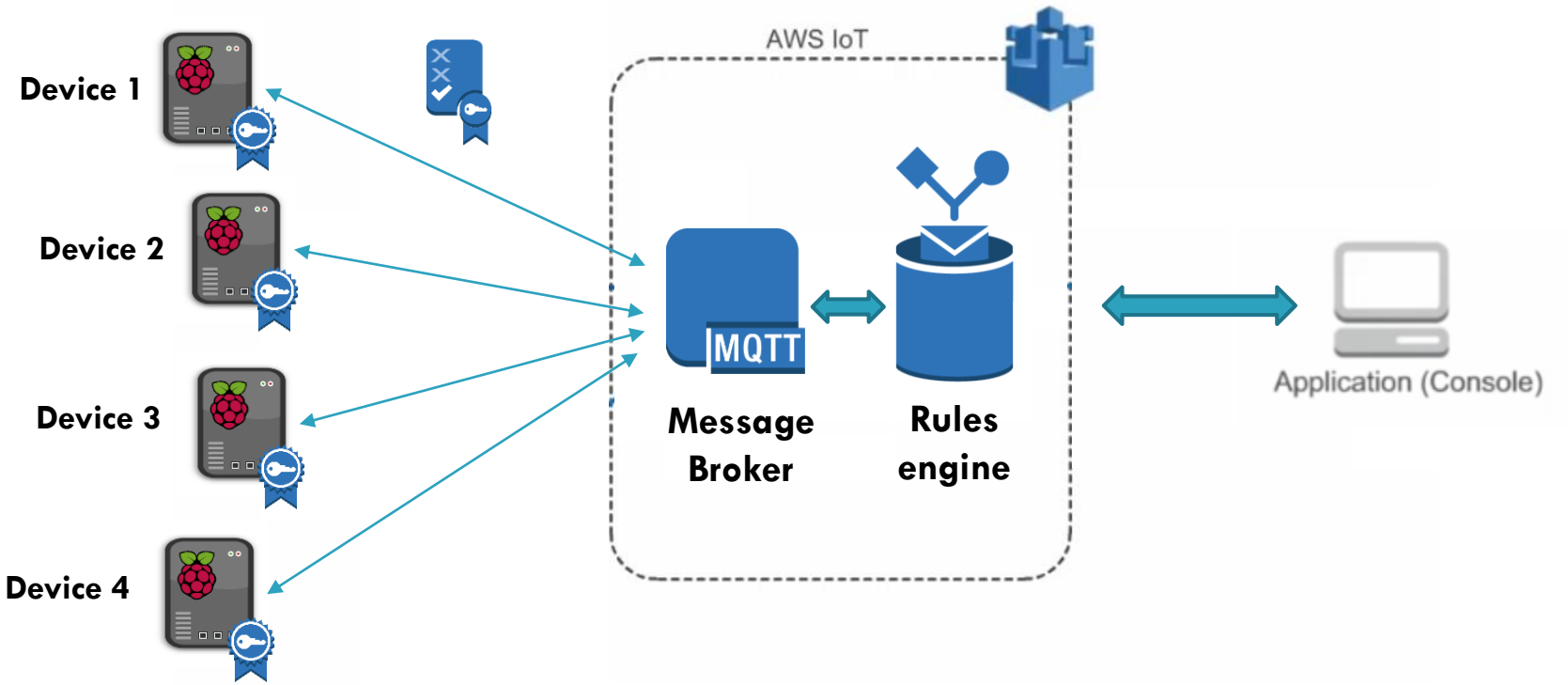
```
//Callback function scatenata alla ricezione di un messaggio sul topic sottoscritto
void iot_subscribe_callback_handler_temperatura(

            AWS_IoT_Client *                    pClient,
            char *                              pTopicName,
            uint16_t                            topicNameLen,
            IoT_Publish_Message_Params *        params,
            void *                              pData

)
```

In particolare, i dati da stampare a viedo possono essere letti dalla struttura dati *params*, che contiene il payload ricevuto.
Per accedere al payload si possono utilizzare le chiamate:

- *params->payloadLen*
- *params->payload*

# Quarto obiettivo

Modificare il programma *aws_iot_core_lab* in modo che il device si registri al topic **SCTM/AWSLAB/alarm**. Grazie ad una opportuna regola impostata su *Rules Engine*, su questo topic saranno inoltrati i soli messaggi che nel campo *temperatura* presentano un valore maggiore di 30. Gestire inoltre la *callback function* in modo che, dopo aver ricevuto un allarme da ciascuno degli altri tre device, il dispositivo si spenga.



# Note per l'implementazione (7)

Per propagare gli allarmi, occorre creare una regola su **Rules Engine.** Per la sottoscrizione al topic e relativa callback si segua quanto visto per il precedente obiettivo.