AWS IoT Core Rules in action: a practical guide and a real-(smart) world use case

18 February 2022 - 8 min. read

AWS IoT Core *Internet of Things (IoT)*

Introduction

The Internet of Things (IoT) has become one of the coolest and most sought-after terms of recent times in Information Technology. In fact, both in the industrial and retail sectors, we find more and more connected objects accessible to the Internet and constantly controllable.

Our previous article presented an overview of the IoT world focused on AWS. Thanks to its services, it offers us many advantages for developing IoT applications on the cloud, a centralized point of management and monitoring of devices thanks to its managed services, always available, in high availability, and with managed scaling.

In this article, we will detail a specific part of the IoT ecosystem on AWS: the rules engine.

Case study: Smart cities

The services related to the IoT world on AWS are constantly expanding. This article will analyze the main components acting directly from the console.

In fact, we will see how to configure a device by simulating the sending of some data from it and then triggering specific actions based on the messages produced by the devices. As discussed in the previous article, these messages are sent and received through the MQTT protocol through a publisher/subscriber mechanism. The HTTP /

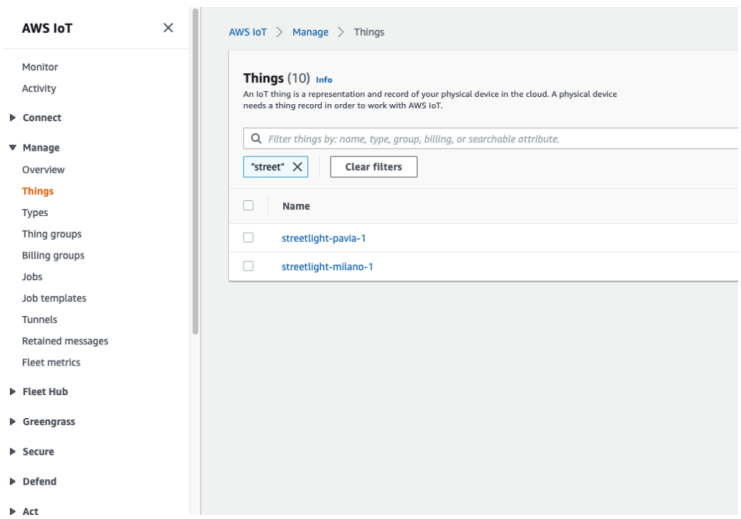
HTTPS protocol is also supported on AWS, but in IoT contexts of certain types, the MQTT protocol is preferable. We will not go into the merits of this choice since it is not the purpose of this article and was partly treated in the previous one.

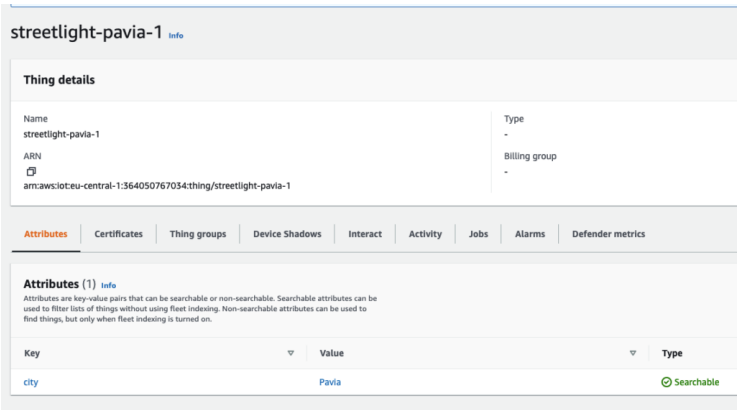
To give context to the practical exercise, let's imagine creating an IoT application for a Smart Cities project. Therefore, we are talking about an urban scenario in which various types of devices can continuously (and not, based on connectivity) send their data to the cloud and carry out specific actions if evaluated correctly.

Things setup

Let's start the deep dive by creating objects (Things) on the IoT Core. To enter into the merits of the case study, let's imagine having to configure sensors installed on street lamps, in the irrigation systems of our cities, or on traffic lights for the intelligent regulation of traffic.

By entering the AWS IoT section on the console, under the Manage menu item, it will be possible to create objects, precisely Things, by configuring some properties, such as associating them with a group, a category, or a type of billing. In addition, it is possible to assign specific tags (attributes) that can be used throughout the AWS IoT ecosystem. In our case, we have given the name of the city to which the object belongs as an attribute.





During the object creation phase, it will be necessary to download the associated certificates. The certificates will be used by the device itself to forward messages on MQTT topics. In this article, we will not go into the details of how it is possible to download the certificates on the device; for now, we will limit ourselves to considering the case of devices already present in the field with certificates in the field. This part will be better described in the following article (Device management)

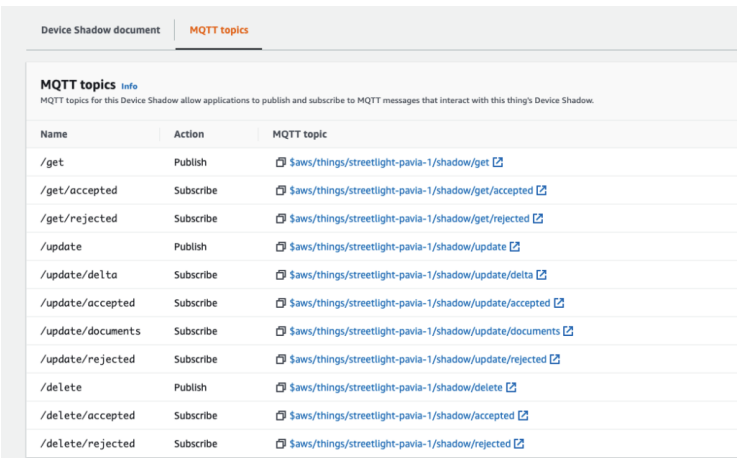
However, we recommend that you save the certificates to a private S3 bucket, so they can be available just in case. For completeness, the certificates associated with a device can be more than one and it is possible to retrieve them if necessary.

Device shadow

When configuring our devices, we have chosen to use the Classic Shadow.

What is the shadow of a device? The shadow is the device's virtual representation, usable by other applications, regardless of their connection status. The same device, web application, and numerous other services can create, update, delete shadows using various channels, including MQTT topics. Shadows are saved on the AWS cloud and, therefore, always available.

By default, some MQTT topics are available through here you can manage the shadow:



We can obviously create additional MQTT topics for our IoT application, but they will not be able to manage the device shadow.

How is device data represented in the shadow? It is nothing more than a document in JSON format precisely containing the state of the device, divided into the following parts:

- **Desired**: The state desired by applications interacting with the device
- **Reported**: The current status reported by the device
- **Delta**: Difference between desired and reported. This part of the shadow is managed automatically by AWS IoT.

Let's see an example of shadow for the street lamps of our smart city:

```
{
  "state": {
    "reported": {
      "light_sensor": 80,
      "light_actuator": "on",
      "last_clean_in_days": 67
    },
    "desired": {
      "light_sensor": 80,
      "light_actuator": "off",
      "last_clean_in_days": 67
    },
    "delta": {
      "light_actuator": "off"
    }
  }
}
```

Rule engine

Finally, let's talk about the Rules! The rules give our devices the ability to interact with AWS services. In a fully event-driven approach, the rules are analyzed every time

events on MQTT topics arrive.

But what can we do with the rules? The possible actions are many; most of them concern native integrations with other AWS services. Let's see some examples:

- We can filter or enrich the data received from the devices before forwarding it to other services
- Write data to various data sources, such as S3, DynamoDB, and Timestream
- Submit data to Cloudwatch to trigger alarms, update metrics or save logs
- Invoke Lambda functions or Step functions

A simplified SQL statement, created ad hoc for IoT Core, defines our rule. It is evaluated every time a device forwards a message on an MQTT topic; if there is a matching between the message and the rule, one or more actions are activated.

The SQL statement is defined as follows:

- **SELECT:** Extracts information from the payload of an incoming event. It is possible to transform it using functions already made available by AWS.
- **FROM:** The MQTT topic on which the rule is listening.
- **WHERE (optional):** Phase in which we can add other conditions that determine when the rule must be positively evaluated.

We can summarize the steps performed by the rules as follows:

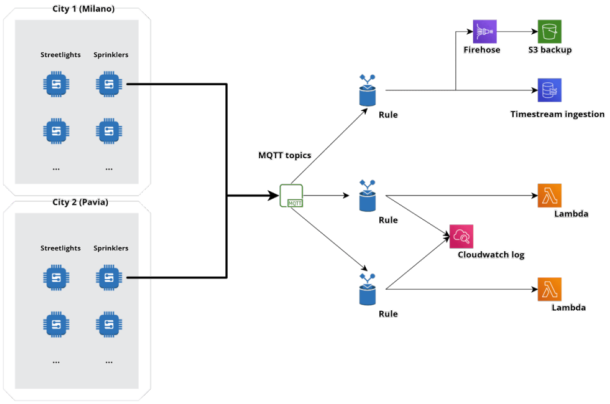
- A rule is evaluated if there is an incoming message on the selected MQTT topic.
- The rule is checked according to the clauses after the WHERE
- If the rule is checked, actions are triggered.

Don't worry, in the next chapter we will see some examples of SQL statements.

Hands-on

Before going into the details of the rules engine, let's go back to our use case: Smart cities.

Simplifying, we can imagine the ingestion part of events with an infrastructural architecture of this type:

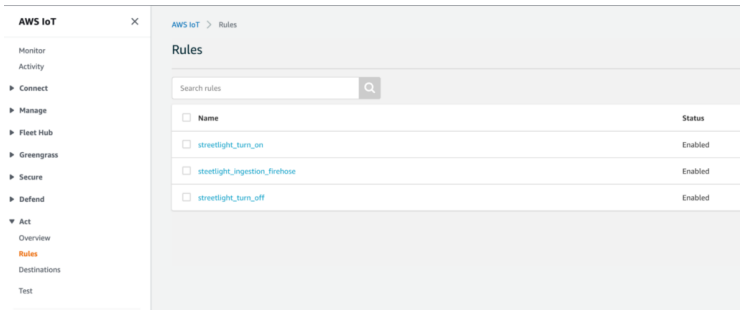


Let's assume we want to make multiple cities "intelligent". On the field we will have a series of sensors physically located in different geographical points, each with difficulty even in having access to connectivity.

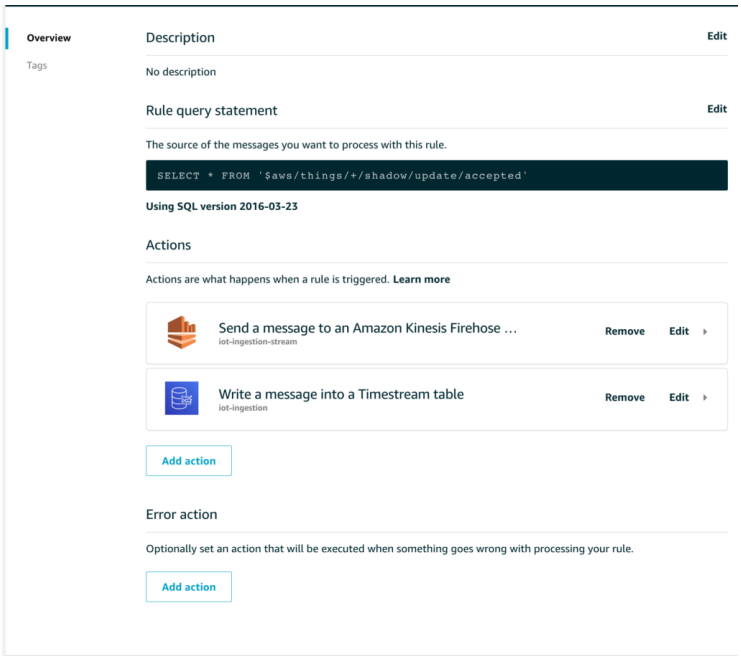
The devices will forward their data on various MQTT topics; for our infrastructure, we have assumed 3 rules:

- I**ngestion rule**: All data arriving on an MQTT topic are forwarded to Firehose / S3 and Timestram for historical storage.
- **Rules of action**: The other 2 rules are evaluated positively only in some specific cases, that is when the system detects that we need to turn on or off the street lamps or sprinklers of our Smart City.

Here they are reported in the console:



Let's see how one of these rules is composed, for example, the ingestion one. The SQL syntax of this rule is straightforward:



In the Rule query statement section you can see how all the input parameters on the topic $ aws / things / + / shadow / update / accepted are selected.

This rule is evaluated positively every time any device (thanks to the wildcard "+" present in the FROM clause) forwards a correctly formatted update message.

As actions we have configured the forwarding of data on S3, passing through Kinesis Firehose, and on Timestream, the serverless database managed by AWS for time series.

It is possible to select up to 10 actions per rule, and optionally you can also configure a rule to manage errors during the processing of the rule itself.

Let's now analyze the other 2 rules used to control our smart city components.

Taking for example the power-up rule, the SQL statement will be formatted as follows:

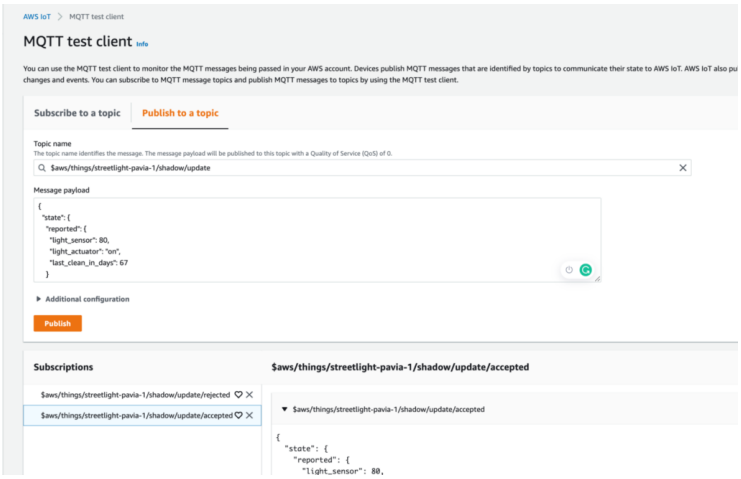
```
SELECT topic(3) as device_id,* FROM '$aws/things/+/shadow/update/acce
pted' where state.reported.light_sensor < 60 and state.reported.light
_actuator = 'off'
```

With this rule, we filter for specific fields in the device shadow reported by the device. Furthermore, in the select clause, we see how to retrieve the device identifier, a fundamental aspect since the rule is evaluated for each device, thanks to the '+' wildcard.

The payload reported in the SELECT will then be forwarded to a Lambda function that will effectively manage the business logic and communicate with the devices.

How can this communication take place? Thanks to MQTT topics, like IoT Rules are listening on MQTT topics, devices can also listen to these topics, reacting to new events.

How can we test everything? There is the Test menu item in the IoT Core panel through which you can find an MQTT client. We can subscribe and publish messages on MQTT topics at our convenience without having a device really in the field.



To conclude

In this article we have seen how it is possible to approach the IoT world with the services offered by AWS, potentially offering a relatively low time to market.

A question could arise spontaneously: why should I design such a complex system to leave all the management to the devices?

The answer is simple: Centralization of management.

Indeed, by exploiting the cloud, we can centralize all the control, delegating the mere functionality of sensors to our IoT devices, and then manage all the logic of our applications in a single point. IT systems are constantly changing and updating. Imagine having to release a software update, every time, on all devices (among other things, this is a feature available with IoT Greengrass)

I hope you enjoyed this first deep dive into the IoT ecosystem on AWS, in the next we will talk about **device management**!

Subscribe to our newsletter to be notified!

Alessandro Bertini

DevOps Engineer @ beSharp. I deal with Cloud-Native software development, strongly oriented to the serverless paradigm!Passionate about board games and video games (as the best geeks do!)
