Deep dive in IoT core rules: Guida pratica e casi d'uso

18 Febbraio 2022 - 8 min. read

AWS IoT Core | Internet of Things (IoT)

Introduzione

L'Internet of Things (IoT) è diventato uno dei termini più cool e ricercati degli ultimi tempi in ambito di innovazione tecnologica. Sia nel settore industriale, che retail troviamo infatti sempre più oggeti interconnessi tra loro, accessibili via Internet e costantemente controllabili.

Nel nostro precedente articolo abbiamo presentato una panoramica del mondo IoT su Amazon Web Services. Grazie ai suoi serivzi, AWS ci offre molti vantaggi per la realizzazione di applicazioni IoT sul cloud come ad esempio la possibilità di avere un punto centralizzato di gestione e monitoring dei device grazie ai servizi managed, sempre disponibili, in alta disponbilità e con scaling gestito.

In questo articolo entreremo nel dettaglio di una parte specifica dell'ecosistema IoT su AWS: il motore delle regole.

Case study: Smart cities

I servizi correlati al mondo IoT su AWS sono in continua espansione. In questo articolo analizzeremo però le componenti principali, agendo direttamente da console.

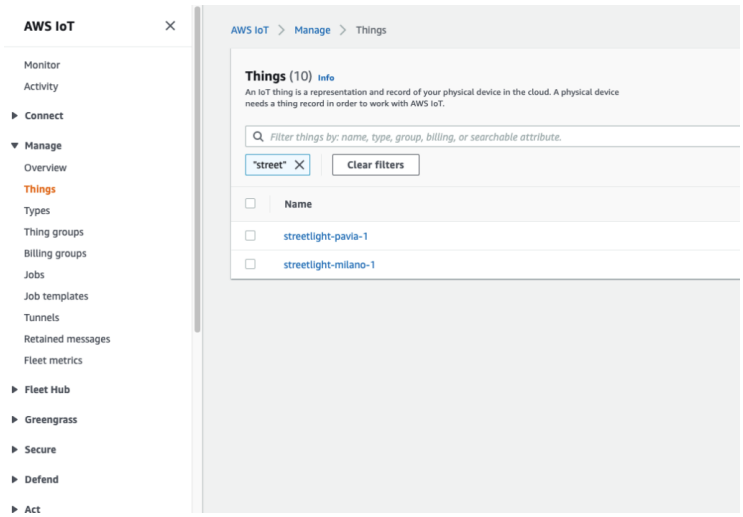
Vedremo infatti come configurare un dispositivo simulando l'invio di alcuni dati da esso, per poi innescare azioni specifiche basate su di essi. Come trattato nell'articolo precedente, questi messaggi vengono invati e ricevuti attraverso **protocollo MQTT** tramite un meccanismo di publisher/subscriber. Su AWS è supportato anche il

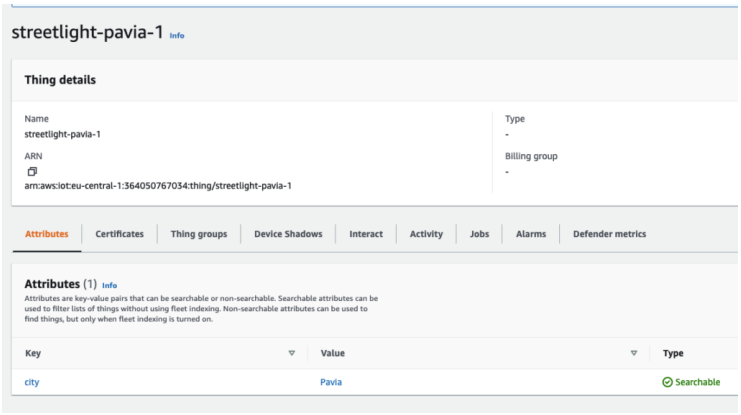
protocollo HTTP/HTTPS, ma in contesti IoT di un certo tipo è preferibile il protocollo MQTT. Non entreremo nel merito di questa scelta dato che non è lo scopo di questo articolo e in parte è stato trattato nel precedente.

Per dare contesto alla parte pratica, immaginiamo di realizzare un'applicazione IoT per un progetto di Smart Cities. Parliamo quindi di uno scenario urbano in cui sono presenti dispositivi di vario tipo, in grado di mandare continuamente (e non, in base alla connettività) i loro dati su cloud ed eseguire delle azioni specifiche se valutate opportumanente.

Things setup

Iniziamo il *deep-dive* creando degli oggetti (*Things)* su IoT Core. Per entrare nel merito dello use case, immaginiamo di dover configurare dei sensori installati su lampioni, impianti di irrigazione o semafori per la regolazione intelligente del traffico.
Entrando nella sezione *AWS IoT* sulla console, sotto la voce di menù *Manage,* sarà possibile creare degli oggetti - Things, appunto - configurandone alcune proprietà, come la loro associazione a un gruppo, ad una categoria o ad una tipologia di billing. Inoltre, è possibile assegnare dei tag specifici (attributi) che potranno essere usati nell'intero ecosistema AWS IoT. Nel nostro caso, abbiamo assegnato come attributo il nome della città di appartenenza dell'oggetto:





Nella fase di creazione degli oggetti sarà necessario scaricare i certificati associati ad essi. I certificati verranno utilizzati dal dispositivo stesso per poter inoltrare messaggi sui topic MQTT. In questo articolo non entreremo nel dettaglio delle varie modalità con cui è possibile scaricare i certificati sul dispositivo, per ora ci limitiamo a considerare il caso di dispositivi già presenti sul campo e dotati di certificati.

[*Spoiler: questa parte verrà descritta nel dettaglio nel prossimo articolo della nostra serie dedicato al Device management!]*

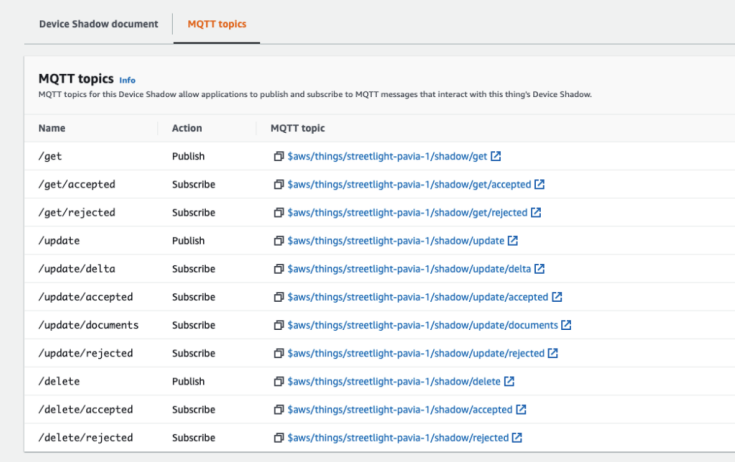
Ad ogni modo, consigliamo di salvare i certificati su un bucket S3 privato, in modo che possano essere disponibili per ogni evenienza. Per completezza, i certificati associati ad un dispositivo possono essere più di uno ed è possibile recuperarli, se necessario.

Device shadow

In fase di configurazione dei nostri dispositivi abbiamo scelto di utilizzare il ***Classic Shadow***.

Ma cos'è lo shadow di un device? Lo shadow è la rappresentazione virtuale del dispositivo, fruibile da altre applicazione, indipendentemente dal loro stato di connessione. Lo stesso device, le web application e numerosi altri servizi possono creare, aggiornare, cancellare i shadows usando vari canali, tra cui i topic MQTT. Gli shadows sono salvati sul cloud di AWS e quindi sempre disponibili.

Di default, sono disponibili alcuni topic MQTT tramite qui è possibile gestire lo shadow:



Per la nostra applicazione IoT possiamo ovviamente creare ulteriori topic MQTT, ma questi non potranno gestire lo shadow del device.

I dati del dispositivo sono rappresentati nello shadow sottoforma di documento in formato JSON conentente appunto lo stato del device, diviso nelle seguenti parti:

- **Desiderd**: Lo stato desiderato dalle applicazioni che interagiscono con il dispositivo
- **Reported**: Lo stato corrente riportato dal dispositivo
- **Delta**: Differenza tra *desired* e *reported*. Questa parte dello shadow è gestita in automatico da AWS IoT.

Ecco un esempio di shadow per i lampioni della nostra 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

Eccoci finalente alle *Rules!* Le regole conferiscono l'abiità ai nostri device di interagire con i servizi AWS. In un approccio completamente **event-driven**, le regole vengono analizzate ogni volta che arrivano eventi sui topic MQTT.

Ma cosa possiamo fare con le regole? Le *action* possibili sono molteplici, la maggior parte di essere riguardano integrazioni native con altri servizi AWS.

Vediamo alcuni esempi:

- Filtrare o arricchire i dati ricevuti dai dispositivi, prima di inoltrarli ad altri servizi
- Scrivere i dati su vari data sources, come S3, DynamoDB e Timestream
- Inoltrare i dati a Cloudwatch per far scattare allarni, aggiornare metriche o salvare dei log
- Invocare funzioni Lambda o Step functions

Uno statement SQL semplificato, creato ad hoc per IoT Core, definisce la nostra regola. Essa viene valutata ogni volta che un device inoltra un messaggio su un topic MQTT. Se esiste un matching tra il messaggio e la regola, allora verranno attivate una o più azioni.

Lo statement SQL viene definito come di seguito:

- **SELECT**: Estrae informazioni dal payload di un evento in ingresso. E' possibile trasformarlo utilizzando anche funzioni rese già disponibili da AWS.
- **FROM:** Qui viene ideinfiticato il topic MQTT su cui la regola è in ascolto.
- **WHERE (opzionale):** Fase in cui possiamo aggiungere ultriori condizioni che determinano quando al regola dev'essere valutata positivamente.

Possiamo riassumere gli step effettuati dalle regole come di seguito:

- Una regola viene valutata se è in arrivo un messaggio sul topic MQTT selezionato.

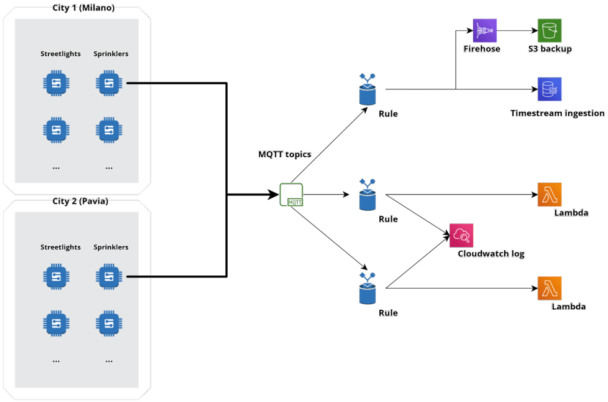
- La regola viene verificata secondo le clausole riportate dopo il WHERE
- Se la regola viene verificata, delle azioni vengono triggerate.

Non preoccupatevi, nel prossimo capitolo vedremo alcuni esempi di SQL statement.

Hands-on!

Prima di entrare nel dettaglio del motore delle regole, ritorniamo al nostro caso d'uso: le Smart cities.

Semplificando, possiamo immaginare la parte di ingestion degli eventi con un'architettura infrastrutturale di questo tipo:

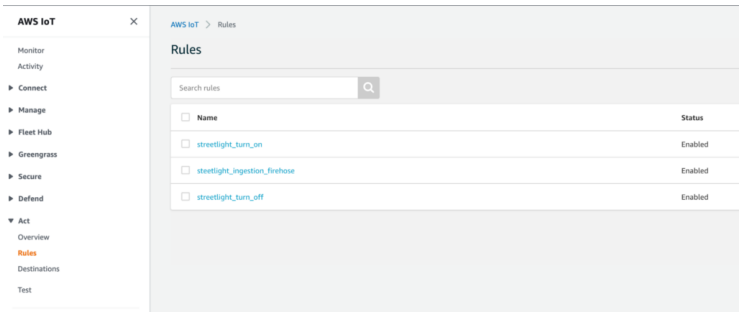


Ipotizziamo di voler rendere "intelligenti" molteplici città. Sul campo, avremo una serie di sensori dislocati fisicamente in punti geografici diversi, ognuno con difficoltà di accesso alla connettività.

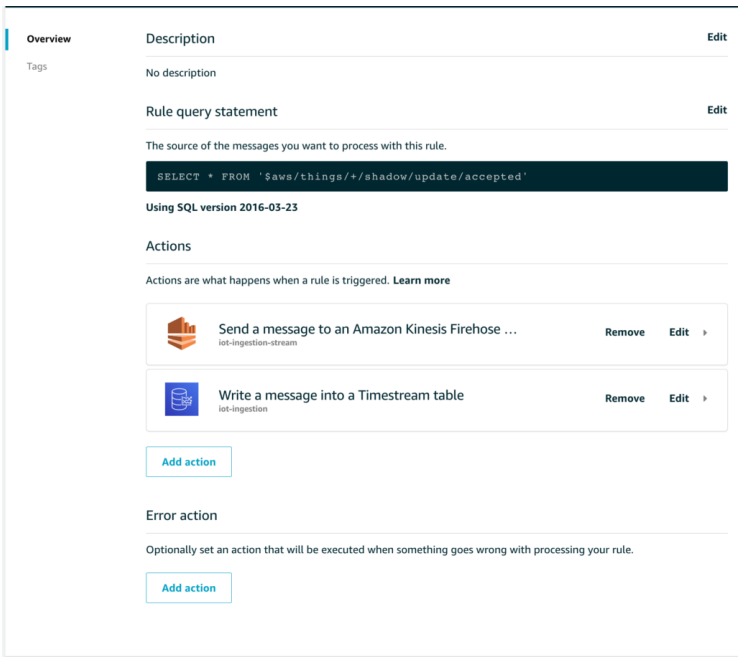
I device inoltreranno i propri dati su vari topic MQTT. Per la nostra infrastruttura abbiamo ipotizzato 3 regole:

- **Regola di Ingestion**: Tutti i dati in arrivo su un topic MQTT, vengono inoltrati a Firehose/S3 e Timestram per memorizzazione storica.
- **Regole di action**: Le altre 2 regole vengono valutate positivamente solo in alcuni casi specifici, ovvero quando il sistema rileva che bisogna accendere o spegnere i lampioni o gli irrigatori della nostra Smart City.

Eccole riportate in console:



Vediamo nel dettaglio come è composta una di queste regole e prendiamo ad esempio quella di ingestion. La sintassi SQL di questa regola risulta molto semplice:



Nella sezione *Rule query statement* si può notare come vengano selezionati tutti i parametri in ingresso sul topic $aws/things/+/shadow/update/accepted.

Questa regola viene valutata positivamente ogni volta che un quaslsiasi dispositivo (grazie alla wildcard “+” presente nella clausola FROM) inoltra un messaggio di update correttamente formattato.

Come azioni abbiamo configurato l’inoltro dei dati su S3, passando da Kinesis Firehose, e su Timestream, il database serverless gestito da AWS per le time series.

E’ possibile selezionare fino a 10 action per regola. Opzionalmente si può anche configurare una regola per gestire gli errori in fase di processamento della regola stessa.

Analizziamo ora le altre 2 regole, quelle utilizzate per comandare accensione, spegnimento o qualsiasi attuatore della nostra smart citiy.

Prendendo ad esempio la regola di accensione, lo statement SQL sarà formattato come di seguito:

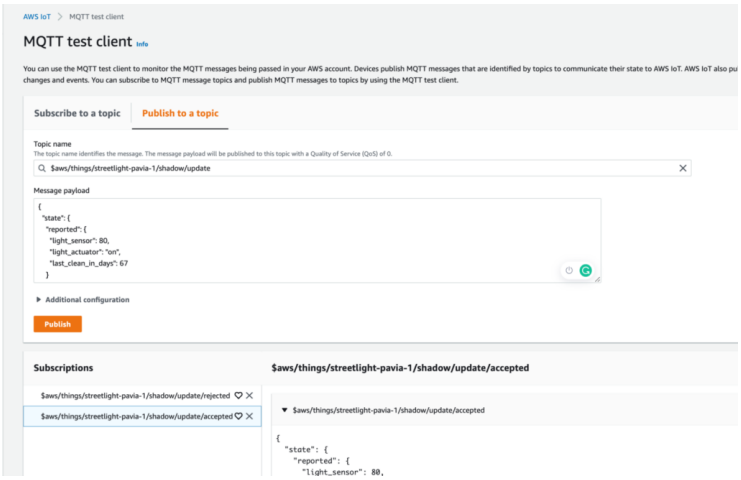
```
SELECT topic(3) as device_id,* FROM '$aws/things/+/shadow/update/accepted' where state.reported.light_sensor < 60 and state.reported.light_actuator = 'off'
```

Con questa regola stiamo filtrando per campi specifici nel device shadow riportato dal device. Inoltre, nella clausola di select vediamo come recuperare l'identificativo del device, aspetto fondamentale in quanto la regola viene valutata per ogni device, grazie alla wildcard '+'.

Il payload riportato nella SELECT verrà poi inoltrato ad una Lambda function che gestirà effettivamente le logiche di business e comunicherà con i device.

Come può avvenire questa comunicazione? Sempre attraverso i nostri topic MQTT! Come le IoT Rule sono in ascolto su dei topic MQTT, anche i device possono essere in ascolto su questi topic, reagendo anch'essi a nuovi eventi.

Vediamo ora come testare il tutto. Nel pannello IoT Core, è presente la voce di menù *Test* tramite cui si può trovare un client MQTT. Da qui possiamo sottoscriverci e pubblicare messaggi su topic MQTT a nostro piacimento, senza quindi avere un dispositivo realmente sul campo.



Conclusione e osservazioni

In questo articolo abbiamo visto com'è possibile approcciarsi in modo relativamente semplice al mondo IoT e abbiamo visto come i servizi offerti da AWS possono aiutare a ridurre notevolmente il time to market.

Una domanda potrebbe sorgere spontanea: perchè “complicarci” la vita è progettare un sistema così articolato quando potrei demandare tutta la gestione ai dispositivi?

La risposta è semplice: **Gestione Centralizzata**.

Sfruttando il cloud, possiamo infatti centralizzare tutta la gestione, demandando la mera funzionalità di sensoristica ai nostri dispositivi IoT sul campo per poi **gestire in un unico punto tutte logiche** delle nostre applicazioni. I sistemi informatici sono in continuo cambiamento e aggiornamento. Immaginate di dover rilsaciare un quaslasi update software, ogni volta, su tutti i dispositivi (tra l’altro, questa è una feature disponibile con *IoT greengrass*)...

Per questo primo deep-dive nell’ecosistema IoT su AWS è tutto. Appuntamento al terzo capito della nostra serie per parlare della **gestione dei device**.

Iscriviti alla newsletter di Proud2beCloud per essere avvisato!

Alessandro Bertini

DevOps Engineer @ beSharp, mi occupo di sviluppo software Cloud-native, fortemente orientato al paradigma Serverless!Appassionato di giochi da tavolo e videogame (come ogni buon smanettone!)
