A comprehensive overview of IoT on AWS

19 January 2022 - 8 min. read

Amazon CloudWatch | Amazon DynamoDB | Amazon S3 | AWS IoT Core | AWS Lambda | Internet of Things (IoT)

Over the last few years, we've seen consistent growth in connected devices' popularity.

These connected devices make up the IoT ecosystem.

Among the applications of the IoT in the industrial or retail sector are telemetry, remote control, inventory, biomedical monitoring systems, fitness wearables, connected gadgets for voice assistants, and much more. All those devices need to be accessible from the Internet to constantly track their status, get notifications or control it from a distance.

This article will deal with the topic in a rather informative and discursive way, starting from the most generic areas up to the problems to be solved, the description of the protocols, and finally, an overview of the services that AWS offers for the realization of IoT solutions.

In the following few chapters, we will look at building an IoT ecosystem, the key points to consider, and which tools can help us.

Bandwidth and data consumption are the keys

Suppose we are planning to build a mass production of connected devices. We need to consider that users could place them anywhere, the amount of transferred data might vary, and the communication might need to be bidirectional. We could end up having devices placed in a room with access to a stable internet connection without constraints on the bandwidth and others operating out in the open connected via a mobile network. These limitations force us to choose a lightweight and elastic protocol suitable for various network conditions and device placement.

Security

We seriously have to consider security as a crucial point of our infrastructure. We want every piece of data sent from our devices to be encrypted both in transit and at rest. Even if the traffic doesn't contain sensitive information according to the applicable laws, we don't want others to discover what is sent or obtain access to information about the underlying infrastructure. Furthermore, data can sometimes be hard to classify, and sensitive information may not be evident at first. Therefore, the safest approach is to consider all traffic and data stored in the cloud as delicate and apply a reasonable level of security at all layers. We must secure the connection tunnel because the worst thing it could happen is that we expose backdoors that can potentially allow anyone to gain access to our devices.

Our solution has to be event-driven and scalable

The typical IoT application consists of a set of actions that are performed on the server-side when clients send messages over the network, including message processing and sending back responses or commands/notifications to the devices. Message frequency might vary depending on the application, with spikes of work alternated with periods of inactivity, so our architecture can benefit from scalability and elasticity. Most of the time, the software needs to react to events, so it can benefit from the event-driven design pattern. When a client notifies the broker that a change of state has occurred, a new event is created, and different areas of the back-end application may take action accordingly.

We also want our architecture to scale with the number of connected devices. We might initially have only a few devices connected to our system, but we need to be elastic enough to meet an increasing workload.

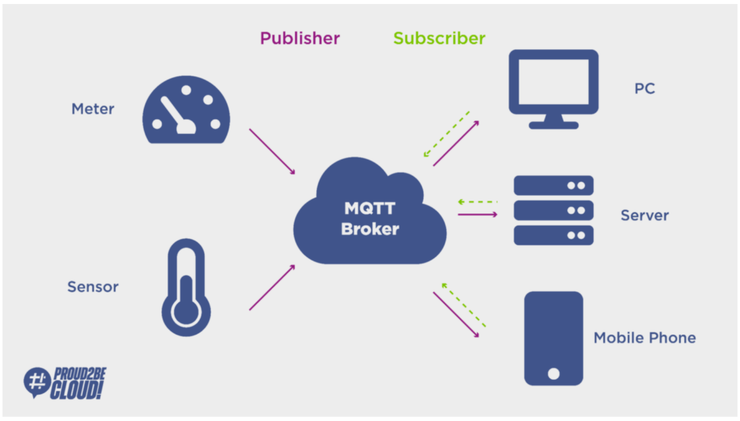
Here comes MQTT

MQTT stands for Message Queuing Telemetry Transport and is a messaging protocol created to suit these purposes. It usually runs on top of TCP/IP and defines two types of network entities:

- The message broker, which is the server that handles the messages received from the clients.
- The clients, which are the devices that send information to the broker.

The message broker offers several topics, which are a way to organize information published by clients, and keeps track of all clients' connection states, security credentials, and certificates.

If the communication needs to be two-way, clients can also subscribe to a specific topic (or multiple if needed) to receive messages published by the message broker.



There are three types of messages clients can send to the message broker:

1. Connect - Creates a connection tunnel that the client will use to send information.
2. Disconnect - Closes the connection tunnel.
3. Publish - Sends a message through the connection tunnel to a topic that the message broker will handle.

You can also configure the quality of service, which is the importance of the information we are transmitting. These are the possible values you can configure:

- 0 - at most once. Messages that couldn't be successfully delivered will be lost.
- 1- at least once. You might end up having duplicated messages on your topic.
- 2 - exactly once. You'll be sure that the message will be delivered.

Since security is a crucial point of IoT infrastructures, MQTT can protect the transferred information using TLS and authenticate clients using modern authentication protocols, such as OAuth.

Deploying IoT ecosystem to the cloud

As promised at the beginning of the article, we will overview AWS's services that can be used to develop IoT applications in the cloud.

But don't worry, other articles will follow in which we will explore the services and the main architectural patterns that we can use in detail.

IoT Core

AWS offers a wide variety of services that can help us build a solid IoT solution, and the most famous is IoT Core.

IoT Core allows you to manage MQTT connections between your devices and the cloud in an entirely serverless environment. You'll only pay for the resources you will use, and it'll scale automatically based on your needs.

You must create devices beforehand in IoT Core, and then you'll be able to create a new connection. Device creation is straightforward. All you need is a name for your thing and configure the certificates used to create an encrypted connection. You can also organize your IoT thing into distinct groups, assign tags to facilitate the filtering, and place them in their own billing group to allocate costs.

Certificates are easy to create: IoT core will provide you the private and public keys with just a few clicks (it is highly recommended to store them in an encrypted S3 bucket because they are not accessible after the thing creation process). If you want to provision your certificates, IoT Core allows you to upload them.

The last thing to do before starting using your new device is to apply a policy.

Policies define what your device can do. For example, you might have things that can publish and subscribe to a specific subset of topics, others that can only subscribe to receive notifications, etc. These security policies can be later changed, and the modification will take effect instantly.

Device shadow

IoT Core adds shadow capabilities to your things. A device shadow is a JSON document that contains the current state of a specific device. Your devices can update it by simply publishing a message on their own topic endpoint. A thing can have multiple named shadows and only one unnamed. The content of the shadows is accessible by your application even when its device is offline, and it is sent back to the device when it reconnects.

Devices can benefit from the shadow since it is a way to store status and configuration in a centralized cloud space that can only be accessed by it and your application.

IoT Rules

Upon topics update, you might want to trigger events that handle the messages sent by your devices. This can be done with IoT Rules. Every rule has a name, a SQL-like statement that allows you to filter messages and select a subset of fields, and a list of actions to take. You can attach several AWS services to your IoT Rules; here are some examples:

- Lambda functions to apply your business logic.
- SNS to fan out the messages to multiple receivers.
- SQS to decouple your logic.
- S3 and DynamoDB to store information in persistent storage for later usage.
- Kinesis Firehose to process multiple messages in batch.

This last service is also handy when you want to reduce S3 API calls and costs: you don't want to call an S3 PutObject for every update sent by your devices because you might face an increase in your bills. With Kinesis Firehose, you can store multiple messages in a single S3 object with a single API call.

Monitor devices

IoT Core console provides an easy way to monitor the general status of your devices. You can see on the service's homepage the number of currently connected things, the number of messages published and the rules executed.

You can also configure IoT Core to publish activity logs and metrics on Amazon CloudWatch.

An example

Suppose we are building an IoT ecosystem with several sensors that gather weather information. Every sensor is an IoT Core device with its own certificates and periodically sends a JSON object containing the sampling information to one or more MQTT topics. Here is what we want to do:

1. We must automate the creation of new IoT things with a Lambda function that configures the new device on IoT Core and stores certificates in an encrypted S3 bucket.
2. Store every message received in a bucket on S3. We don't know yet if we will need the history of the samplings for a device, so it's a good practice to keep this data in a cost-efficient storage service such as S3.
3. Save messages on DynamoDB (or Timestream) to retrieve them quickly. We want to show charts on a web application stored in its S3 bucket protected with Amazon CloudFront, and data retrieval is mandatory.
4. Handle connection and disconnection events with e-mail notifications. We can easily do this with a Lambda function rule that triggers Simple Email Service upon automatic messages on the connect and disconnect topics. We could also publish a custom metric on CloudWatch to keep track.

This possible architecture satisfies all our needs: it is entirely serverless, which means it can scale based on our needs; it is cost-efficient because we only pay for the resources we are using; it's elastic since it can grow in terms of functionalities because every aspect of our application is decoupled from the others; connections between clients and IoT Core are encrypted, so our infrastructure is safe and secure.

Conclusions

Concluding the article, we explored the main areas of interest regarding the IoT world; we introduced the MQTT protocol and AWS IoT Core and the critical features for building IoT applications on AWS.
Stay tuned for the series of articles in which we will deepen the issues related to IoT and of which this is only the introductory chapter!

Mattia Costamagna

DevOps engineer and cloud-native developer @ beSharp. I love spending my free time reading novels and listening to 70s rock and blues music. Always in search of new technologies and frameworks to test and use. Craft beer is my fuel!