AWS IoT Core Device Management in action

18 March 2022 - 7 min. read

AWS IoT Core *Internet of Things (IoT)*

IoT ecosystems consist of thousands or more devices connected to a centralized backend that handles connections and inward and outward messages. When building our project, we need to consider how to organize our resources to allow us to search them quickly. If we don't do this, we might operate in a chaotic environment that is difficult to manage and debug.

Device management means a way to filter our objects quickly, find out which ones aren't operating as they should, have organized storage for certificates and messages in a scalable manner.

This article concludes the trilogy of our IoT overview: we described what you should consider when creating your IoT project, how you can take advantage of AWS Rules management to trigger other services in an event-driven designed application, and finally, how to organize your devices pool in the best possible way!

The main issues we need to consider

Automation

Many tasks, such as device creation, deletion, and certificate renewal, usually require multiple actions. We must automate these operations because there is a high chance of error. We can quickly do this on AWS with Lambda functions (or Step functions if the automation is more complex) that can be triggered as events by various other parts of our application.

Security

Security is probably the most crucial aspect of IoT applications: as discussed before, we don't want unauthorized people to access our devices. Certificates need to be stored in encrypted storage with limited access and replicated to multiple Availability Zones and, if possible, regions to reduce the chance of data loss. S3 is the perfect service for this scope because it offers encryption at rest (either managed or with KMS), replication to another bucket in another AWS region, security policies, and IAM integration to give access only to the people and applications that need to perform operations on it.

Scalability

What is the order of magnitude we expect as the number of devices connected to our ecosystem? Is our application B2B or B2C? These are some questions we want to ask ourselves to define a way to organize our devices and information in an ordered manner that will help us find what we are looking for, even if the amount of filterable data is massive.

Scalability also involves the ability of our system to grow automatically based on the traffic. Serverless architectures are built to suit this purpose, so every time we add new features, we should look for a solution that doesn't require static resource provisioning.

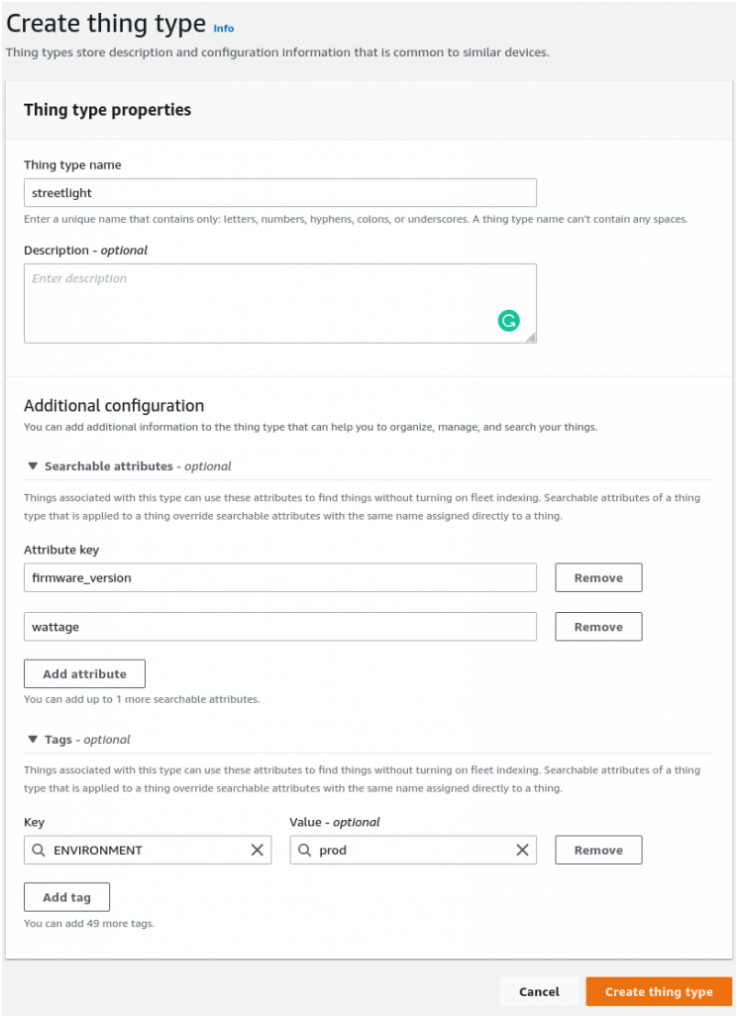
AWS IoT Device Management

IoT Device Management is one of the many features of AWS IoT Core. It

offers many tools to help you organize your fleet of devices in a centralized webpage. Let's see some of these features.

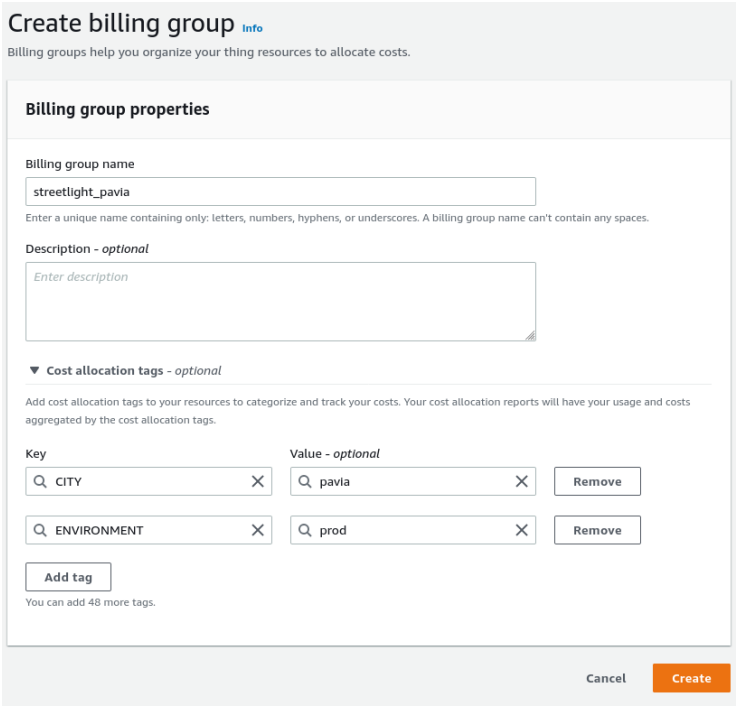
Thing types

Thing types allow you to define a general description of your things and a set of attributes that every device associated with it must have. An IoT thing can be of one thing type. Thing types are immutable, so you cannot add an attribute to your thing type after creating it. If you need to do this, you have to deprecate your thing type and create a new one. You can't associate a thing to a deprecated thing type. Think of thing type as skeletons for things configuration: if a device is of type "streetlight", it will have the attributes "wattage" and "firmware_version".



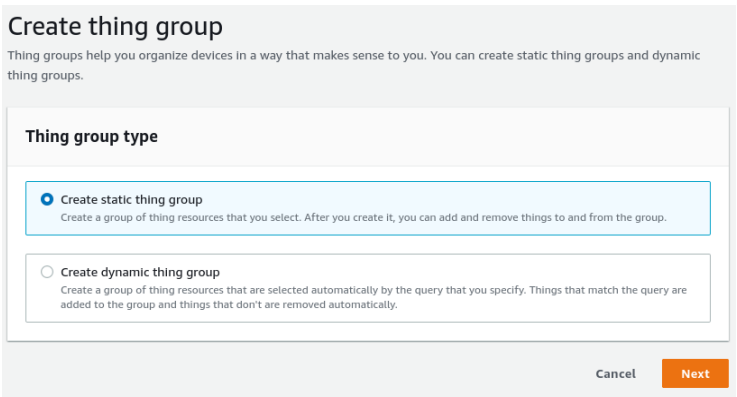
Thing groups

Thing groups help you classify your things in a way that makes sense to you. They can define attributes as well, but unlike thing types, you define both the attribute name and value, and this information will be shared across all things that are part of this group. Groups can either be static or dynamic. Static groups are, in some ways, similar to thing types, except they can be nested hierarchically. Static groups are also helpful if you need to attach the same policies to multiple devices, but you want to abstract this to a higher level. Dynamic groups help you gather things assertively based on query conditions. You can't attach policies to dynamic groups or nest them inside one other, but you can leverage them if you need to do something to all the devices that meet some criteria.



Billing groups

Billing groups are used to allocate, categorize and track your cost usage with the help of tags. When you apply tags to billing groups, AWS generates a cost allocation report file with your usage and costs aggregated by your tags so you can determine how your budget is distributed across your things fleet.



Real-world use case

With these tools, we can organize our devices in a granular way. Think about an application that manages a fleet of connected street lights: as we said, every device has a thing type associated with it that defines a set of attributes that every device must have (the version of the installed firmware, the drawn wattage, etc.), a static thing group with a policy attached that defines the set of allowed actions, a dynamic group that contains all the devices that run an old version of the firmware (so we can use this group as the target for our update firmware job), and finally a billing group for each city we are placing things to (so we can better track costs).

Jobs

IoT jobs are a set of operations that we want to send to a target. Targets can be specific things or thing groups. We can define the operations in the form of a UTF-8 encoded JSON document that contains one or more URLs used by the device(s) to download updates and other data. Jobs are helpful to manage once-in-awhile operations like updating the device firmware, downloading local configuration, renewing certificates, etc.

Here's a job example that tells the devices to download a file from an S3 bucket, execute it and finally perform a system reboot:

```
{
    "version": "1.0",
    "steps": [
        {
            "action": {
                "name": "Download-File",
                "type": "runHandler",
                "input": {
                    "handler": "download-file.sh",
                    "args": [
                        "${aws:iot:parameter:downloadUrl}",
                        "${aws:iot:parameter:filePath}"
                    ],
                    "path": "${aws:iot:parameter:pathToHandler}"
                },
                "runAsUser": "${aws:iot:parameter:runAsUser}"
            }
        },
        {
            "action": {
                "name": "Install-Application",
                "type": "runHandler",
                "input": {
                    "handler": "exec-script.sh",
                    "args": [
```

```
                        "${aws:iot:parameter:filePath}"
                    ],
                    "path": "${aws:iot:parameter:pathToHandler}"
                },
                "runAsUser": "${aws:iot:parameter:runAsUser}"
            }
        },
        {
            "action": {
                "name": "Reboot",
                "type": "runHandler",
                "input": {
                    "handler": "reboot.sh",
                    "path": "${aws:iot:parameter:pathToHandler}"
                },
                "runAsUser": "${aws:iot:parameter:runAsUser}"
            }
        }
    ]
}
```

Tunnels

Frequently, in IoT infrastructure, you need to gain access to your devices, especially for debugging problems. If your device is deployed behind restricted firewalls is not that easy to do so, and here comes AWS IoT Tunnels. This feature lets you establish bidirectional secured communications to any device of yours and can save you the time and cost of sending a technician to the device's place to investigate the problem. To create a tunnel, all you need is to configure which service you want to use to establish the connection (e.g., SSH), the device you want to connect to, and the duration of your tunnel lifespan. At the end of the creation process, you'll receive two access tokens that are needed to start a local proxy (you can find the source code here https://github.com/aws-samples/aws-iot-securetunneling-localproxy) on your local machine. After launching the local proxy, you'll be able to SSH into your device through the local proxy. The tunnel will automatically close itself after the given timeout expires.

Managing certificates and policies

Certificates management is the foundation for a well-architected infrastructure. It would help if you had reliable automation for creating, renewing, and deleting certificates, as these actions are prone to errors. AWS IoT provides a centralized page in the console to control your devices' certificates, but it should be used more for monitoring than management. Certificates are shown in a paginated table view, and you can select each entry to view its details. You shouldn't create certificates directly from this page (unless you are just testing a new solution or building a POC) because clean-up will be challenging as many resources are linked to each other, and you might end up working in a chaotic environment. Instead, it would be best to manage certificates through a coded process (Lambda is ideally suited for this task). Also, remember that you can download certificates during the creation phase only. You won't be able to retrieve them afterward, so be sure to save them to persistent encrypted storage for later use! Another best practice is to create devices with dedicated certificates to add a security layer to your infrastructure: if one of your certificates is compromised, you won't expose access to every device, and you can quickly disable and delete the impaired certificate.

Policies define what your devices will be able to do. You can attach a policy to a certificate or a thing group to reuse the same policy for multiple devices. When you define policies, you should follow the standard security advice of granting *least-privilege* access to your resources. If you don't do this, you could get unexpected behaviors and expose access to parts of your application that need to be hidden.

Conclusion

AWS IoT offers many centralized features to organize your resources without building this from scratch. Take advantage of them! Solid device management is the foundation for a reliable application, so you should initially spend some time designing this part in the best possible way.

In our 3-article series we approached some of the most crucial aspects to consider when developing IoT projects on AWS.

Did you face any other critical aspects you would like to know more about? Let us know in comments!

And see you again in 14 days on Proud2beCloud for a new blog post!

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!
