AgileVision

Building IoT SaaS on AWS

The sharing economy trend became more and more popular between individuals and companies. The mindset of buying instead of building and renting instead of owning became something completely normal for everyone. Indeed, why spend thousands and millions of dollars creating a complex solution if it can be rented from someone. Furthermore, building something takes time, while buying a tool allows using it almost instantly.

In parallel, this shift in mindset completely changed the approach to innovation. Before, access to data and being able to process it were privileges of large corporations. Now cloud computing and powerful tools made it possible for smaller companies to leverage the technology for their growth. The innovation became affordable.

Now, as never before, because of the fast-paced world, individuals demand tools that would make their lives better, while companies are looking for instruments to grow faster and get a competitive advantage. It is no surprise these days **Software as a Service** (abbreviated as SaaS) solutions experience such high demand. According to Gartner, demand for SaaS will continue to grow through 2024.

Another trend we are observing at AgileVision is the high interest of customers in the Internet of Things. Prototyping and production of IoT devices are affordable as never these days and give many opportunities to verify the idea and launch a product. Before, the “lean startup” approach often meant “lean software startup,” but these days, it does not matter since it is possible to quickly validate the idea and create an MVP even if your idea involves a hardware device.

In this whitepaper, we share insights, approaches, and common pitfalls of creating an IoT SaaS on AWS. This whitepaper will be extremely interesting for those who want to build their Software as a Service that involves IoT.

We cover only building B2B (Business-to-Business) IoT-enabled SaaS solutions because it is our main area of expertise. And we are familiar with all the ins and outs of them.

Structure of the whitepaper

This document is separated into the following sections:

Structure of the whitepaper 3

What exactly is SaaS? 4

What is the difference between traditional SaaS and IoT-enabled SaaS? 4

Internet of Things and security 5

Data exchange between the IoT device and the SaaS solution 7

- *Pull-based data exchange* 7
- *Push-based data exchange* 8
- *Combined data exchange* 8

Popular hardware platforms for IoT devices 11

- *Microcontroller-based devices* 11
- *Microprocessor-based devices* 12
- *Device provisioning* 13
- *The manufacturing process allows installing unique security credentials to each device safely before it is delivered to end-users* 14
- *The manufacturing process allows installing shared temporary security credentials to a batch of devices before these devices are delivered to end-users* 14
- *There is no way to install any security credentials during the manufacturing process, but the end-user can use a special app to install credentials to the device* 14

Services and tools provided by AWS that can be leveraged when building IoT SaaS 16

Selecting a Managed Service Provider to develop IoT SaaS 17

What exactly is SaaS?

Software as a Service(SaaS) is the software delivery model in which the user (tenant) receives access to a cloud-based application to perform their tasks instead of installing it on a local machine or on-premise server.

You can think about SaaS as paying for accessing the application functionality instead of receiving a copy of the software. In most cases, all tenants receive the same or similar set of features, which can differ based on the subscription type/tier.

What is the difference between traditional SaaS and IoT-enabled SaaS?

Despite many common things, a SaaS which involves hardware devices can be completely different from the business and technical perspective. Before creating an IoT SaaS, you must have answers to some questions.

Among tricky business model related questions are:

- *Who owns the device?*
- *What should be the price of the device?*
- *Who is responsible for providing an Internet connection for the device?*
- *Who should be provisioning the device?*
- *Can the user bring their own device?*

The most important technical questions are:

- *How to upgrade the device firmware?*
- *How to monitor, troubleshoot and debug devices that are a thousand miles away?*

We will discuss technical questions in the following sections and how these issues can be solved efficiently. Unfortunately, we do not have answers for the business side of things because they completely depend on the selected business model. We can only emphasize that in the case of IoT SaaS, logistics is one of the core contributors to product success.

Internet of Things and security

There is a famous saying: “S in IoT stands for Security.” But it is an oversimplification. Modern IoT service providers like AWS IoT Core are designed so that it is not possible to create a not secure connection to the device. In fact, any security issue connected with IoT will be most likely caused by improper configuration or usage of the tool rather than the problem in this tool itself.

The most popular way of securing IoT devices is using TLS - an industry standard for protecting data-in-transit. There are already experiments using post-quantum cryptography for TLS, which is critical for long-term hardware deployments.

Things are a bit complicated with data at rest and identity theft. When we are talking about a device located thousands of miles from us inside a facility we do not have access to, the only correct tactic is not to trust the device.

We need to assume at any point in time somebody can:

- *Move the device*
- *Disassemble the device*
- *Dump the device memory*
- *Modify the device memory*

Of course, IoT devices can have tampering detection mechanisms, but let's not try to fool ourselves. Unless formally proven (like the security of TLS) or can be formally proven the opposite, the attacker can obtain complete control of the device without the SaaS provider noticing it. Obviously, there are solutions on the market that can be the device tamper-resistant, but nothing can make your device tamper-proof.

This means:

- *The amount of data stored on the device must be limited;*
- *Security credentials (which will anyway be present on a secure device in some form) must be rotated and should not grant more permissions than the ability for the device to transfer its data;*
- *It should be possible to invalidate security credentials connected with a particular device in case if it turns out to be compromised;*
- *A compromised device must not automatically mean a compromised fleet. In other words, each device should have its own set of credentials;*
- *In order to provide the highest possible level of security and manage expectations of end-users, a shared responsibility model can be applied. A SaaS vendor ensures data is secure in transit and at rest. Meanwhile, it is the end-user responsibility to ensure the physical access to the IoT device can be performed only by authorized team members.*

Data exchange between the IoT device and the SaaS solution

The real value of an IoT device is the possibility to process data somewhere else. To do so, it should be somehow transmitted. First, let's deal with the data transfer initiator.

Data exchange can happen on a pull-based, push-based and combined basis. Let's review each approach in more detail and discuss its advantages and disadvantages.

Pull-based data exchange

In the pull-based approach, the SaaS platform is responsible for requesting data from IoT devices. Especially, it is handy in two cases:

1. *The device was not originally designed to be a part of an IoT system. Instead, it has a way to expose some data for remote access. Remote IoT platform just reads the data and pushes it to some internal queue as it was actually sent by the remote device.*
2. *The device is designed in a way it can only perform data reading on external requests. Or intermediate readings are not required until some condition is met. It can happen for compliance, privacy reasons, or large data volume reasons. For example, you may be dealing with an IP camera with a reasonably large buffer. It can act as an edge device and constantly capture the video/audio stream and perform rotation as needed. But the transfer to the SaaS would only happen if the system is in an alert state.*

The main disadvantage of the pull-based approach is a heavy load on the SaaS platform if a large number of devices must be polled. It may become very challenging to poll thousands of devices in a reasonable timeframe.

Additionally, data that appeared at the beginning of the polling interval will be ready only during the next cycle, which can be somewhat late.

Push-based data exchange

In this case, the IoT device pushes data to the IoT provider endpoint either when the data appears or after enough data is buffered. The main benefit of the push-based approach is reduced load on the SaaS platform itself. With a push-based approach, the platform does not need to spend computing power going over each device and asking whether it has some data ready to be consumed.

Such an approach is very convenient for capturing streaming time-series data. Based on data volume and the number of devices, a receiving party can scale up or down to handle increasing or decreasing load.

On the other hand, if the platform becomes unavailable even for a moment, the data can be lost. It becomes the responsibility of the device to retry data delivery if the IoT endpoint was inaccessible for some reason.

A direct push-based approach can also be problematic when it involves a non-uniform payload size with unpredictable spikes.

Combined data exchange

As you can see, both pull-based and push-based have advantages and disadvantages. Instead of using one of those, it is possible to combine the strong side of both:

1. *Use **push-based** approach for time series or streaming data*
2. *For data exchanges that require large-volume transfers, **notify the SaaS platform using the push-based** approach about the data availability. Then SaaS will be able to queue data fetching and perform it when it is the most appropriate time.*
3. *Allow the SaaS platform to trigger data reading from the IoT device remotely.*

Remember that combining push and pull approaches may not be required for every IoT SaaS, but it can be a powerful tool for some.

Besides the data transfer initiator, you need to think about the protocol used for data exchange. Among popular protocols are:

- *MQTT*
- *HTTPS*
- *Websockets*

Additionally, some devices use file transfer protocols like:

- *FTP*
- *SFTP*
- *FTPS*
- *SMB*

Finally, when the device needs to transfer the data in some custom binary format, the connection between the IoT device and SaaS may be established using lower-level transport protocols like TCP or UDP. In this case, the encryption/decryption, retries, and many other things must be implemented from scratch.

While data transfer protocol does not make a SaaS IoT-enabled or not, underlying IoT services like AWS IoT Core usually support only MQTT, HTTPS, and Websockets. Any other protocol will require an additional custom implementation.

If it is possible, you should be using a protocol supported by the IoT service of your choice. Since, besides data ingestion, platforms like Amazon Web Services have many useful tools that would automatically work with the IoT service, like streaming the data into the database, performing analytics in real-time, and much more.

Popular hardware platforms for IoT devices

The choice of platforms for IoT devices is so big; this section can be easily turned into a separate whitepaper. I will focus on the popular and affordable ones with many available developer kits and strong support from the open-source community.

First, you need to decide the amount of computing power and memory required by your particular device. Based on that, there are two big options:

- *Microcontroller-based device*
- *Microprocessor-based device*

While both sound very similar, there is a substantial difference between them.

Microcontroller-based devices

Microcontrollers (MCU) can be characterized as fully autonomous devices. Inside an MCU, you can find a CPU, RAM, ROM, and IO controls. They possess way less computing power than microprocessors but usually are more energy-efficient, occupy less space, and are cheaper.

Many popular controllers include mechanisms to sign the code and prevent further modification of the firmware. Modern microcontrollers are also shipped with Wi-Fi and Bluetooth modules.

A microcontroller is a great choice for situations when edge computing is not required, while the main goal of the device is to control other simple devices and read sensor data without further processing.

Among popular microcontroller platforms are:

- *ESP32*
- *Cortex M (e.g. STM32 or Nordic nRF53 series)*

The trend of recent years is to get rid of 8-bit controllers like PIC or AVR and replace these with more powerful and cost-efficient Cortex M or ESP32.

Microprocessor-based devices

Microprocessors, unlike MCUs, are more powerful and can be used for edge computing purposes, processing large streams of data. The downside is that microprocessors require additional modules to be attached, such as RAM.

Of course, more computational power results in higher energy consumption and less battery life. While some microcontroller-based devices can run for years on a single small battery like CR2032, microprocessor-based devices usually require some external power supply. Besides that, microprocessors are often more expensive than microcontrollers.

A microprocessor is a great choice for complicated devices that require large data volumes processing on edge, intense computation, or processing audio/video streams.

Device provisioning

The key difference between ordinary SaaS and IoT-enabled SaaS is present physical devices (IoT Gateways, IoT Hubs, sensors, appliances). To make the device talk to the cloud and the application, you must configure it with corresponding firmware and security credentials. This process is referred to as “provisioning.”

There are many approaches to device provisioning, though any of these has the same goal: to configure the device so that it can communicate with the SaaS in a trusted manner.

As I discussed in previous sections, physical devices are specifically prone to MITM (man-in-the-middle) attacks and require even more sophisticated security measures than the software. Still, a good old problem of securely transferring a secret message is being solved:

1. *We have a device somewhere in the field*
2. *We have a SaaS platform*
3. *Both the device and the SaaS platform should have a secure channel to talk to each other*

A combination of symmetric and asymmetric cryptography methods widely known as “TLS” is the most common solution to the problem. The trick is a reliable exchange of public keys in this case. Everything is simple with the SaaS platform - it is hosted in the cloud, the key can be published via the Internet. The complexity comes on the opposite side. How do we know the public key of the device is a valid key rather than a key sent by an impostor? Depending on the manufacturing process and other factors, there can be different solutions. Let’s review four possible situations.

The manufacturing process allows installing unique security credentials to each device safely before it is delivered to end-users

It is the ideal situation from the security perspective. In this case, the user needs to turn on the device, connect it to some network (if required), and it can start working. Unfortunately, this is difficult to achieve for small-batch devices with limited security capabilities. For this method to work, the thing is that the device must have a secure permanent read-only area to store the public key and a special chip for storing the private key. In the other case, it will be possible to dump the device memory and implement the MITM attack.

The manufacturing process allows installing shared temporary security credentials to a batch of devices before these devices are delivered to end-users.

This situation is somewhat similar to the previous one. The only difference is that all devices from the batch share the same temporary security credentials. It is often caused by the fact that provisioning each device will not be feasible or impose too much overhead.

Of course, compromising a single device in such a case affects all devices in the batch. That is why it is even more critical to ensure temporary security credentials cannot be retrieved by a third party.

There is no way to install any security credentials during the manufacturing process, but the end-user can use a special app to install credentials to the deviceю

Sometimes it is not possible to provide security credentials during the

manufacturing process at all. For example, because the device does not have a secure area to store these credentials. In order to handle the provisioning of security credentials, a third entity is required: a trusted user.

The flow looks as follows:

1. *A trusted user initiates the provisioning process on the SaaS side by using a web UI or a special application;*
2. *A trusted user puts the device in a provisioning mode by using some physical control (e.g., hardware reset button on the device);*
3. *SaaS generates temporary provisioning credentials, usually an alphanumeric code or a QR code;*
4. *A trusted user inputs provisioning credentials into the device via a special app or using the keyboard on the device. The important thing is that the provisioning code is being transferred to the target device without reaching any public networks. Possible options are: transferring credentials via a Bluetooth connection, establishing a peer-to-peer Wi-Fi connection between the app and the device, entering activation code using a physical keyboard of the device.*

As you probably already noticed, a trusted user establishes a secure offline channel between the SaaS and the device provisioned.

Services and tools provided by AWS that can be leveraged when building IoT SaaS

Amazon Web Services provides a wide range of managed services for different purposes. The AWS IoT portfolio is extremely rich. Let's focus on the most popular and widely used services for building IoT-enabled applications.

AWS IoT Core

The AWS IoT Core service can be fairly called a foundation of Amazon Web Services IoT offering. It provides a backbone for communication between the cloud and IoT devices via the MQTT and HTTPS protocol. AWS IoT Core handles the main security features of IoT like certificate management, permissions, and policies.

This service consists of a control plane and a data plane. Besides the security and communication components, AWS IoT Core provides a powerful rules engine. AWS IoT Rules Engine provides a scalable and easy-to-use way of ingesting, processing, and analyzing data coming from IoT-enabled devices.

AWS IoT Greengrass

AWS IoT Greengrass is a cloud service and a runtime to create edge computing solutions. This service allows seamless integration of local data processing on the edge device with powerful AWS cloud services to streamline video processing, ML inference, and data streaming.

Selecting a Managed Service Provider to develop IoT SaaS

Since not always the required expertise can be found in-house, a managed service provider can be a good extension for an in-house team (or even an alternative to!). Finding a proper vendor is a challenging task. You can find more details about it in another **blog post.**

Just remember, the process will take some time and may require substantial effort from your side as a company owner/CEO to ensure you are working with a partner who can deliver.

Final words

As a business owner and a co-founder of several startups, I admire your desire to create a product. It is a very challenging (and sometimes rewarding) path. A Software as a Service with IoT capabilities is an even more difficult story. I hope this write-up was useful and shed some light on the possible challenges of creating an IoT SaaS.

If you have some questions, feel free to ping me over email (volodymyr@agilevision.io) or via LinkedIn (linkedin.com/in/vrudyi).

AgileVision