Selection guide for over-the-air (OTA) software updaters

Requirements to consider while selecting an OTA software updater for connected embedded devices (the Internet of Things)

Tel: +1 650 670-8600 contact@mender.io www.mender.io

As the intelligence of connected devices (the Internet of Things or IoT) increases, the need to update its software increases accordingly. Connected devices still have a long journey ahead to standardize on a dependable software update process compared to updates for the datacenter, where production running services are updated frequently.

If connected devices are to be ubiquitous, a key requirement is the reliability and predictability of the software update process. This guide points to areas of importance when selecting an appropriate over-the-air (OTA) updater.

Where to start

A logical start is what kind of OTA solution fits your project's needs:

1. Open source
2. Closed source (Proprietary)
3. Platform-dependent

Open source has been gaining traction in the world, specifically for embedded development, and OTA will not be an exception. The embedded development world is still searching for more standardization and while that happens, projects require flexibility with no vendor lock-in. Embedded projects also are increasingly in need of a domain independent software updater as end-to-end platforms introduce extraneous complexity and in many instances, the updater is very basic: little consideration to all the reliability and security nuances.

Closed source OTA products currently provide the most maturity, while integrated updaters are found in most platform solutions. Platform solutions typically offers convenience at the cost of flexibility, including the "rip-and-replace" issue for existing toolchains that might be in use, such as OS and application build workflows.

Problem definition

Code Complete by Steven McConnell states that on average, 1-25 bugs/defects exist in 1,000 lines of code. Security vulnerabilities are unavoidable whether they come from your own software or 3rd party libraries used by the device. The operational cost and brand damage of recalling a fleet of devices to enforce a manual update can be unrecoverable. Finding a proper OTA solution is a requirement and something that cannot be taken lightly.

By today's standards, a modern OTA solution needs:

- Ease: to manage the update process for an entire fleet of devices, it must be intuitive to use and quick to deploy

- Secure: considers all security nuances (e.g. authenticity, secure communications protocol) to reduce the risk of compromise
- Extensible: to support evolving requirements and work in conjunction with existing and future tools in the stack

Basic functionality

While OTA is conceptually easy to grasp, it is much harder to implement as there are many considerations and moving pieces. Some basic functionality to consider:

Artifact and device compatibility

Ability to link software versions and device compatibility. Installing incompatible software could brick the device.

Rollback

In the case of an unsuccessful update, there should be a safe way to rollback to the previous version. Snapshots, transaction recording, tracking file changes, and isolating updates from one another are ways to achieve this depending on update type.

Resilient updates

Unfortunately, several things can go wrong during an update. Your OTA should account for potential issues such as power loss, installation error, and poor network connectivity leading to partial or incomplete downloads.

Device grouping

Especially with large fleets, devices need to be managed with arbitrary grouping filters such as customer types, location, hardware version, SLAs, ownership, and more. To avoid manual and error prone work, the OTA updater should support dynamic grouping of devices.

Artifact management

Especially with the extensive lifetime of certain devices, you can expect a myriad of both software and hardware versions. Your OTA updater should include artifact management to help cope with this complexity by incorporating a software artifact repository supporting important metadata about each artifact.

Update type support

The updater must support your preferred method of updating your devices. Four main methods of updating exist: image-based, package-based, file-based, or container-based. All come with their pros and cons and your needs may change over time.

Atomic

An update is fully applied or not at all. Only the updater should be aware of partial updates. If any other software components are exposed to partial updates, the device might be bricked or operationally vulnerable.

Security

Managing connected devices that live outside firewalls call for immaculate attention to security. If your device directly or indirectly impacts the health and lives of people, there is no compromise to security. Below are minimum requirements.

Artifact integrity and authenticity

Software updates should be signed by an authority and verified by the device before installation. Your updater needs to ensure that only the correct software is installed on the correct device.

Encrypted traffic

Unencrypted traffic should never be allowed in order to avoid arbitrarily injecting or exposing update deployments.

Access controls

Basic identity management, authentication, and authorization services are fundamental to your updater. Device lifecycle from onboarding to decommissioning should be gated with relevant access controls.

Secure and simple bootstrapping

The OTA updater should offer a secure way to onboard and authorize known and unknown devices. It should support manufacturing processes where devices are prepopulated with identities and keys.

Operational

Software updates and security patching often suffer from fear of breaking something in production. Today, this is the biggest fear for patching embedded systems in production. And thus a long list of priorities coupled with this justified fear makes many fall behind in patching these known security vulnerabilities. An important aspect of your OTA updater revolves around the operational functionality to help reduce the fear of doing updates.

Logging and compliance

Every action by a user or device as part of an update should be logged.

Monitoring

Visualization of the health of your fleet of devices. If something is wrong with a device or group of devices, you will need data to investigate and address before initiating an update.

Phased rollouts

Assuming all your devices are exactly the same, it is reasonable to think that if an update works well on one device, it should work on the remaining in the fleet. In reality, your devices might not be exactly the same and gradual updates can greatly reduce the consequences of an erroneous update.

Notifications

A notification service reduces the fear of breakage as it allows for quick feedback to the user as to the state of your device fleet.

Device inventory

Whether holding its own inventory or being integrated with a CMDB or other inventory system, the OTA updater needs to manage an inventory through APIs.

Short time to deploy

If something critical occurs, your updater should be able to push out an update immediately.

User access controls

Role-based access control (RBAC) ensures that only authorized users are allowed to use your updater. Along with logging, the ability to know who did what is required. Confidence in having only authorized users granted access is paramount to operations and trust in the system.

Compatibility

Protocol support

Your updater should be able to support various messaging protocols to allow for greater flexibility on the device side. It should be possible to add new protocols if a desired one is not already supported.

3rd party tooling

Most connected device (IoT) projects consist of a combination of homegrown applications and 3rd party libraries and components. On the resource-limited side, your updater should be flexible enough to accommodate the need for integration as well as future-proofing (extensibility).

Conclusion

The Mender team has spoken to many individuals responsible for connected device projects. Most of them have built a homegrown tool to deploy updates to devices in the wild. The vast majority were not able to satisfy several of these key requirements, leaving the update process exposed and vulnerable. While the concept of an updater is quite simple, there are many reliability and security details to accommodate for a successful project.