IoT Device Observability Requirements Checklist

Memfault | EMBEDDEDARTISTRY

Software Build Capabilities

Requirement	Details	Risks
Unique Software Versions	Each software build needs a unique identifier to differentiate it from other builds. Ideally, this version will follow a standard format (such as Semantic Versioning).	You cannot correlate information in device logs, crash dumps, and debugger output with a specific version of software, making it difficult-to-impossible to properly interpret debug information. You also have no idea where to start your investigation, since you have no idea what changes are present on the device
Store and Index Software Build Artifacts	Multiple software builds will be used in the field at any given time. Because function and variable address locations will change from one build to another, you must be able to access build artifacts for all versioned builds to debug them appropriately.	Debugging without the debug symbols or map file for a given build means that you cannot properly decode addresses into functions, variables, and specific lines of code. Without a foolproof system in place for storing and indexing artifacts, these files can go missing or be mislabeled.

Device-Side Capabilities

Requirement	Details	Risks
Debug Log	Enables developers to write strings and values in a human-readable format that can be read back later to find error messages or recreate a sequence of events that occurred on the device.	Debugging sealed or production units with a debugger is often difficult or impossible. Devices that are unable to be retrieved (e.g., they are in another country) are impossible to debug in a systematic way without an RMA process.
Reset Reason Detection	Many processors have a status register that indicates why the system (re)booted. The system should also write the cause for a manual reboot to a known, reserved memory location. On boot, this information can be read, logged, and used to determine whether specific fallback behaviors will be executed (e.g., boot into a fail-safe build when an infinite processor fault loop is detected).	It becomes difficult-to-impossible to detect the reason why the system resets due to abnormal causes (e.g., watchdog timeout, brownout) and whether your device is stuck in an infinite reboot loop
On-Device Crash Dumps	When an assertion or processor fault occurs, the software can automatically collect, display, and store commonly collected debug information: software version, values of local variables,	Faults and assertions that occur in the field cannot be easily debugged by developers due to lack of context and information...

	register values, a call stack backtrace, function inputs, and contents of the internal log buffer (if used). Automating this process improves the speed and ease with which developers can debug issues and enables them to debug assertions and faults without a debugger physically attached to the device.	Reproducing the issue with a debugger attached is likely the only recourse, and this significantly lengthens the debugging process by days or weeks.
Device Metrics	Development teams want to track battery life, power consumption, memory usage and more. Product teams and executives want insight into how often features are where engineering investments should be made. The device needs a way to log this information for future collection and analysis.	Without device metrics, development teams could introduce performance, power, or stability regressions into firmware upgrades without knowing. Without metrics on usage, the company will have little insight into how devices are used by customers, making it difficult to assess the value of past and future product investments.
Persistent Storage	Key debugging information should be saved to persistent storage (e.g., SD card, flash) so that information can be recovered even if power is lost. When RMA units are received, critical debugging information will persist on the device.	The device will only be able to report information that occurred since the most recent boot-up / power sequence. Devices with limited RAM risk having older-yet-still-valuable debug information overwritten before it can be reported.

Debugging Infrastructure

Requirement	Details	Risks
Automated Symbolication of Crash Dumps	The raw addresses from crash dumps (e.g., 0x8000ABCD) need to be symbolicated into human-readable variable/function names (e.g., main.c::325) using the debug symbols or map file.	Every time a developer looks at a crash log, they must manually convert the addresses to function/variable names and offsets within functions. The tedium of this process means that crash logs will not be used by the development team for debugging except for the direst circumstances.
Automatic Issue Detection	Crash dumps are positive indications of problems with the software. Software automation can be used to convert collected crash dumps (and other error scenarios) to issue reports. Developers can be notified immediately when a new issue is detected.	Developers must manually review logs and crash dumps and file tickets for new issues. Because this process is both manual and time-consuming, it will cause a delay between when a problem is introduced and when it is noticed by the team.
Automatic Issue De-duplication	If one device is seeing a problem, the odds are high that other devices will see the problem as well, causing the team to be flooded with issue reports. Software automation can be employed to identify duplicate issues without manual developer involvement and to annotate issues with occurrence rate across software versions.	When a team becomes flooded with issue reports, manually triaging reports becomes a bulk processing task due to its time-consuming and soulless nature. New issues, potentially critical ones, will be missed while waiting for someone to triage the backlog of issue reports.

Remote Monitoring and Management

Requirement	Details	Risks
Device Data Collection	Devices must be able to send information to the centralized monitoring system, including reset reason, crash dumps, debug logs, and device metrics. Once in the monitoring system, data can be aggregated and further investigations can be made.	Without access to the rich array of information provided by devices, fleet monitoring is impossible. Rather than detecting issues automatically during a firmware update rollout, the first signs of trouble will be customer support calls and angry tweets. By then, it's too late.
Heartbeat Messages	Devices should send periodic "heartbeat" messages to the monitoring system to indicate that they are alive and well. Messages should be sent on boot and at a regular interval.	Without a check-in on boot, it is hard to determine whether a remote software update was successful. Without periodic check-ins, it is difficult to know whether devices in the field are functioning properly or stalled/frozen
Issue Alerting	Remote monitoring software should observe debug information, check-ins, and device metrics. In addition to de-duplicating and filing issues when crash dumps are received, the system should automatically report issues when certain conditions are met (e.g., a device doesn't check in within a certain period of time after an update, a device's power level drops below a specific threshold).	Developers must spend time manually monitoring incoming device data for problems. Manual monitoring will involve delays, and potentially important problems will not be caught as early as they could have with automatic detection.
View of Device-Level Metrics Over Time	Device metrics should be collected into a view that can show how the metrics for a single device change over time. This provides deeper insight into how a single device is performing and enables correlations between metrics (e.g., observing that a large power drop correlates with a spike in CPU usage). This view is critical for customer support teams	Factors leading to the degradation of a device's performance over time may not be noticed, increasing debugging difficulty. It also increases the difficulty in providing preventative maintenance and support.
View of Fleet-Level Metrics Over Time	Fleet-wide parametric data metrics can be aggregated so that your team can gain insight into the total population of devices. With this view, your team can observe trends and spot regressions of key metrics after updates, between software versions, and in relation to external factors (e.g., mobile phone or gateway updates).	Without fleet-wide metrics, your team will primarily be looking only at problematic devices (i.e., those that trigger issue alerts) or noisy customers.
Cohort Binning of Devices	You should be able to group specific devices together in a "cohort", such as "development devices", "beta testers", and "standard customers". You should be able to filter devices based on their cohort when viewing fleet-level metrics and performing updates.	Without the ability to put devices into cohorts, it becomes difficult to separate internal development devices, beta test devices, and standard customer devices from one another, impacting our analysis of fleet-level metrics and making it difficult to control which devices receive which software versions.