

How Embedded Device Observability Helps Latch Build Ultra-Reliable Products

Presented By:

Tyler Hoffman - Co-Founder & Head of Developer Experience, Memfault

Dave Webster - Engineering Manager, Latch



Memfault

LATCH[®]

Today's Speakers



Tyler Hoffman

Co-Founder & Head of
Developer Experience



Memfault



Dave Webster

Engineering Manager

LATCH[®]

Agenda

- ◇ Memfault Overview
- ◇ How Latch Utilizes Memfault
- ◇ Q & A



Poll #1

If you have a cloud or app development teams in house, do they have an observability solution?

Check all that apply

- A. We have a team(s) in house and they have observability***
- B. We have a team(s) but they don't have observability***
- C. We don't have a team(s) in house***
- D. We have a team(s) but I don't know if they have observability***

Poll #2

Do you have observability for your embedded devices?

Check all that apply

- A. Yes, and it's sophisticated***
- B. Yes, but it's limited***
- C. No we don't have anything***
- D. I don't know***



**Why should embedded observability
be so far behind cloud?**

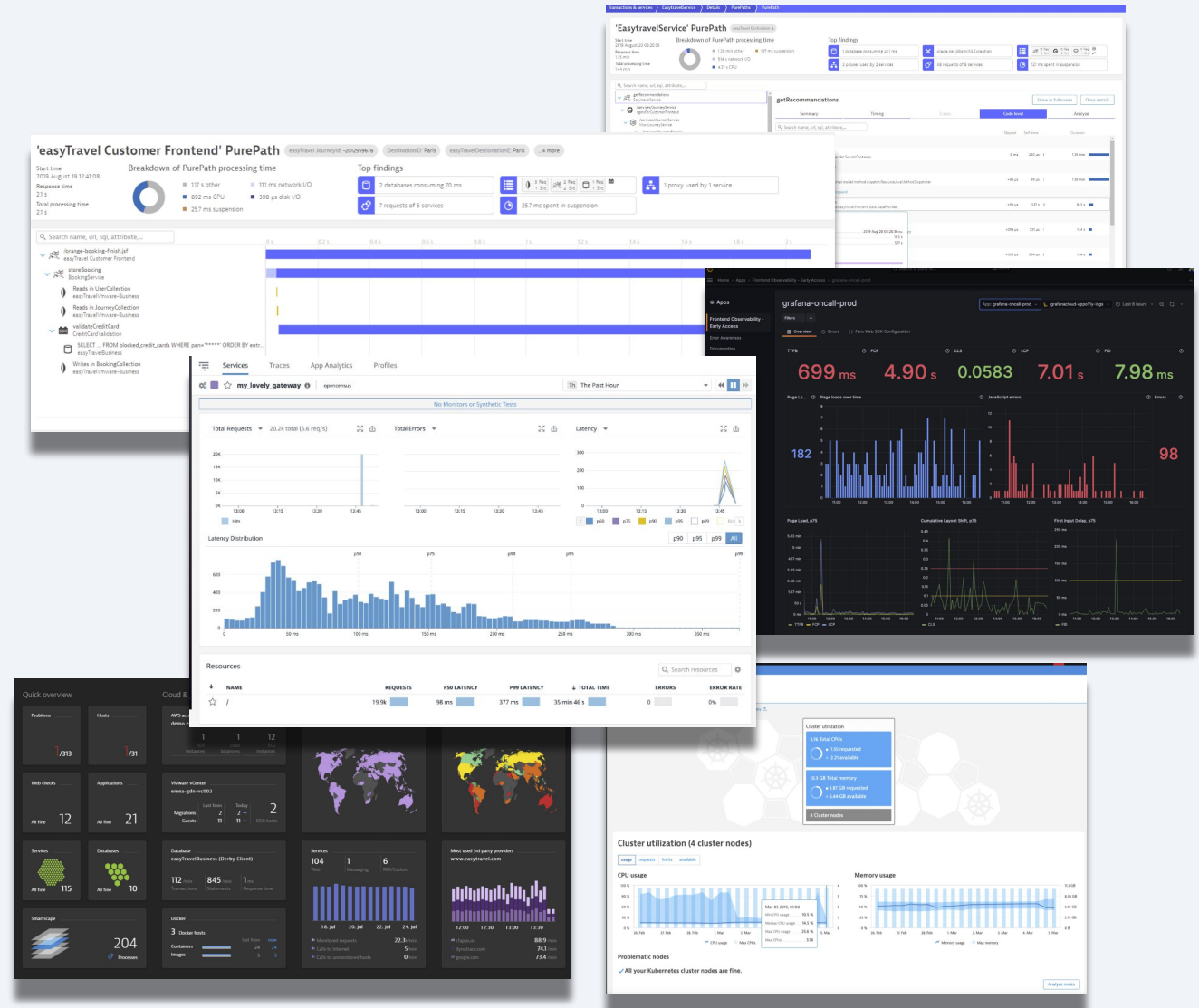
What happens when your cloud system crashes?

You get automatically alerted

Response time is seconds to minutes

Almost too much data to diagnose and solve

Probably fixed before your customer even notices



What happens when your device crashes?

You don't even know it's happened unless a customer reports it

You might be able to collect some logs

You rely on customer feedback to reproduce


You might end up having to replace the whole device



Slow response time

Bad user experience

Potentially expensive resolution



**But hardware should just
work... right?**

Hardware - it works





But what about software?

Hardware + Software = ??

Hardware



- Can have manufacturing defects
- Is exposed to the environment
- Wears out
- Can rust
- Can get dented/bent/damaged

Needs replaced when broken

Software

- Can have developer mistakes
- Docker is a godsend
- Only “rots”, and very slowly
- Can write in C, C++, Rust, and more!
- If it works once, it will likely “just work”

Can be fixed without leaving your desk



**Making software the most
reliable part of the IoT**

Step 1: Acknowledge it's never going to be perfect

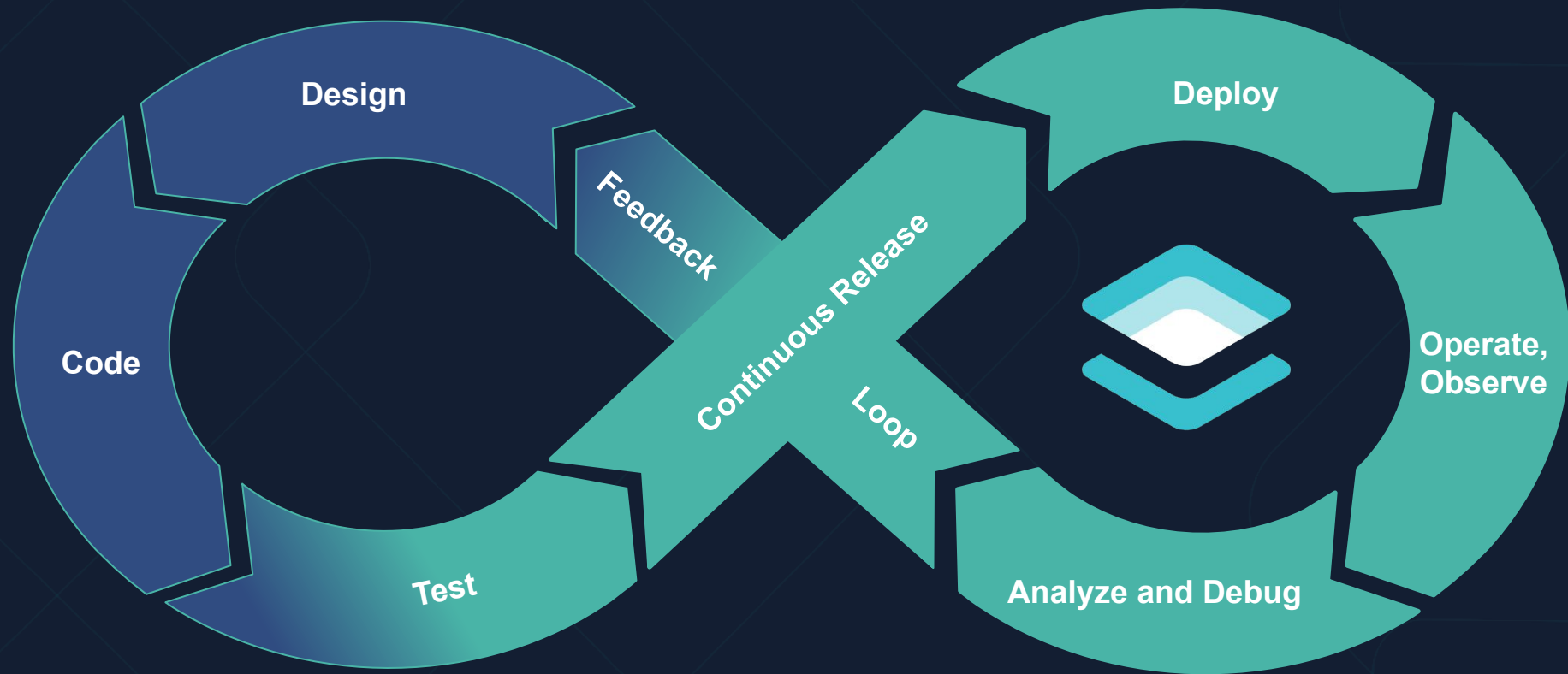
Software is never going to be perfect.

You will ship bugs.

“The top 1% of developers will ship 11 defects per 1000 lines of code.”

Jack Ganssle,
Embedded Systems Expert

Get a process in place



Identify & Prioritize

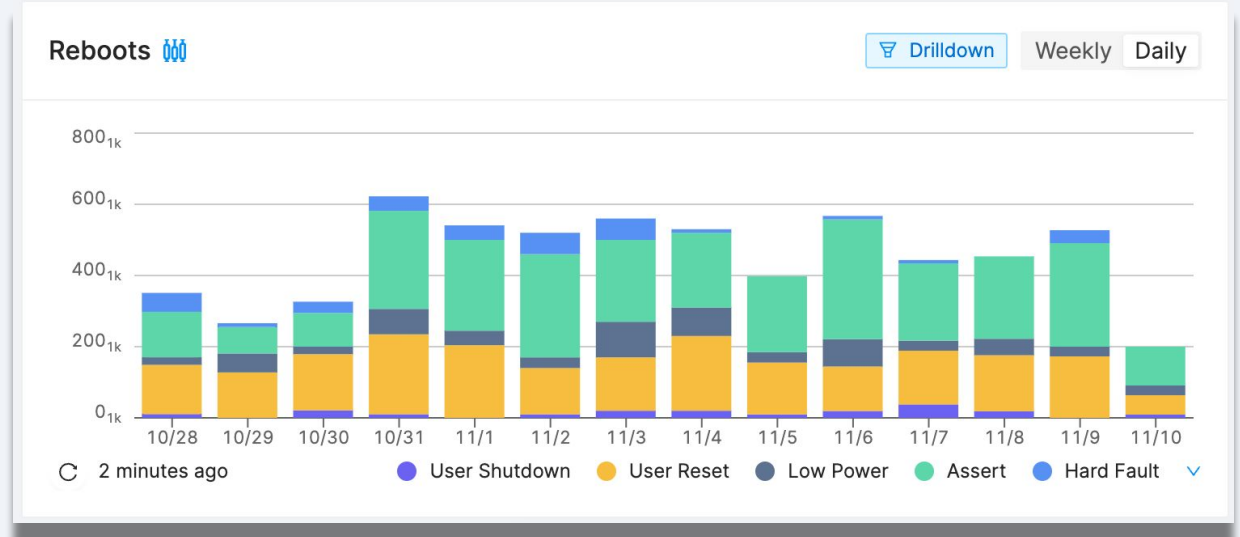
Are my devices crashing?

How many different types of crashes are happening?

How often is each type of crash happening?

Then prioritize.

	Traces ?	Devices ?
<input type="checkbox"/> Assert at gatt_send_next_pkt pvt-software 1.1.0 14 hours ago - 10 days ago Assert	88	12
<input type="checkbox"/> Assert at gatt_enqueue_next_pkt pvt-software 1.1.0 14 hours ago - 10 days ago Assert	92	12
<input type="checkbox"/> Assert at queue_dequeue pvt-software 1.1.1 - 1.0.0 18 hours ago - 15 days ago Assert	49	39



Diagnose (and fix)

Reproducing issues that happen in the field is really hard.

Avoid it at all costs.

Skip reproducing, save loads of time and get straight to fixing.

The screenshot shows a crash report for an application. At the top, the title is "Assert at gatt_send_next_pkt". Below the title, there are buttons for "Resolve" and "Merge", and a "Create in Jira" button. A summary bar shows "First Seen 10 days ago", "Last Seen 14 hours ago", "Recent Traces 88", and "Devices Impacted 12".

The "Details" section shows the following information:

- Device: MFLT0000110
- Cohort: Production
- Software: 1.1.0 (pvt-software)
- Hardware: pvt

Below the details, there are three blue bars representing a stack trace. To the right, there are navigation buttons for "Older" and "Newer", and a "Captured a day ago" indicator.

The "State" section shows "Logs" and "Coredump" options. Below this, there are tabs for "Threads", "Exceptions", "Registers & Locals", "Globals & Statics", "Heap", and "Memory Viewer".

The "Threads" section shows a list of threads:

- main (2) (1) [RUNNING]
 - 0 prv_fault_handling_assert in .../memfault_fault_handling_arm.c at line 555
 - 1 memfault_fault_handling_assert in .../memfault_fault_handling_arm.c at line 567
 - 2 gatt_send_next_pkt in .../src/ble/gatt_service.c at line 10
 - 3 send_char_notify in .../src/ble/gatt_service.c at line 44
 - 4 main in .../shapemate/src/main.c at line 95
- idle (3) [READY]
- shell_uart (4) [READY]
- sysworkq (5) [BLOCKED]
 - 0 arch_irq_unlock in .../aarch32/asm_inlined_gcc.h at line 96
 - 1 arch_swap in .../arm/core/aarch32/swap.c at line 44
 - 2 z_swap_irqlock in .../kernel/include/kswap.h at line 185
 - 3 z_swap in .../kernel/include/kswap.h at line 196
 - 4 z_pend_curr in .../zephyr/kernel/sched.c at line 842
 - 5 z_sched_wait in .../zephyr/kernel/sched.c at line 1893
 - 6 work_queue_main in .../zephyr/kernel/work.c at line 660
 - 7 z_thread_entry in .../lib/os/thread_entry.c at line 36
 - 8 0xa0000000

The "Memory Viewer" section shows a list of memory addresses and their contents:

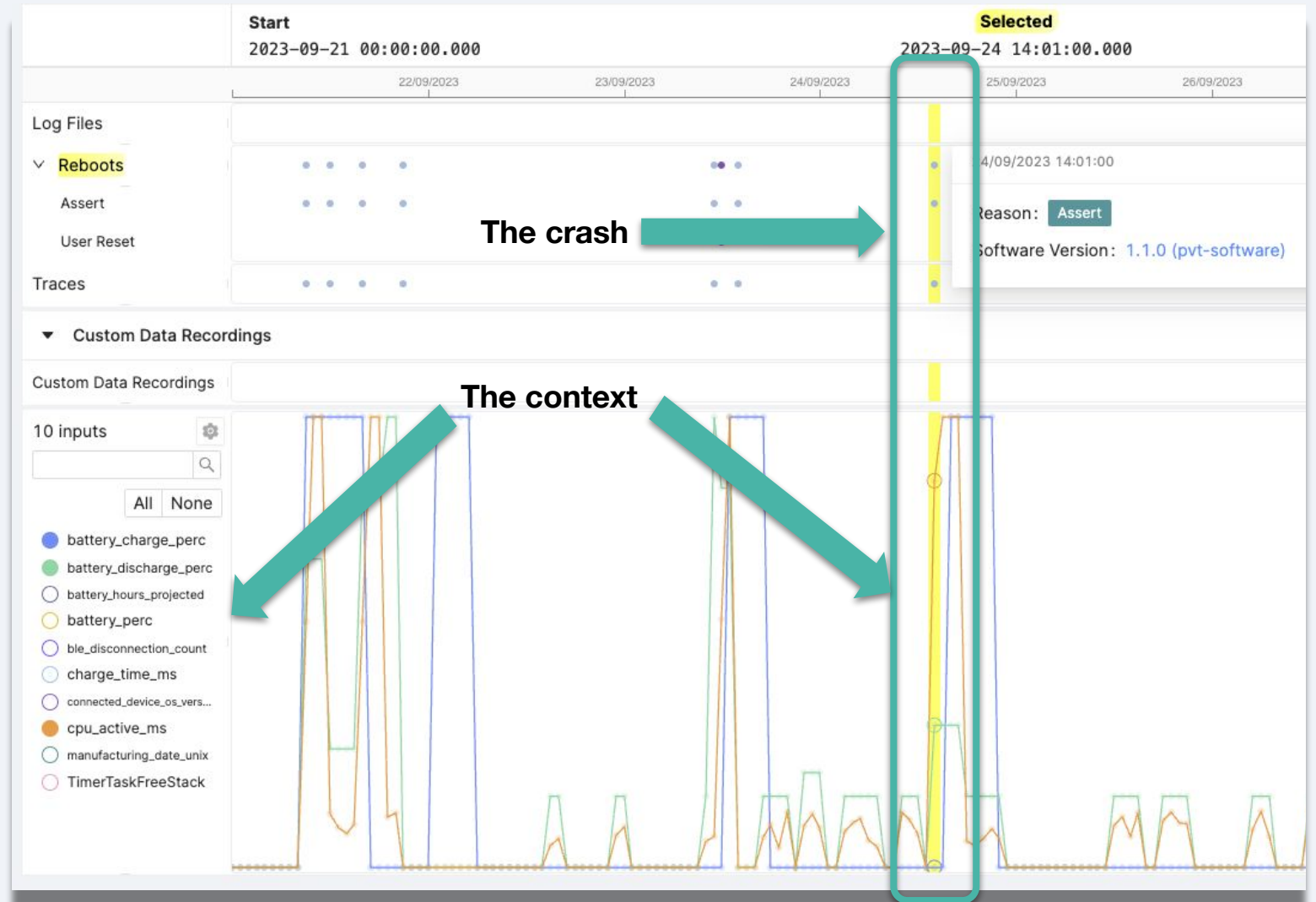
Address	Content
0x20000108	log_dynamic_gatt_service = log_source_dynamic_data {...}
0x2000010c	log_dynamic_log = log_source_dynamic_data {...}
0x20000110	log_dynamic_log_uart = log_source_dynamic_data {...}
0x20000114	log_dynamic_main = log_source_dynamic_data {...}
0x20000118	log_dynamic_mfit = log_source_dynamic_data {...}
0x2000011c	log_dynamic_os = log_source_dynamic_data {...}
0x20000120	log_dynamic_shell_uart = log_source_dynamic_data {...}
0x20000124	log_dynamic_shell_uart = log_source_dynamic_data {...}
0x20000188	s_log_flush_semaphore = k_semaphore {...}
0x20000258	s_memfault_log_data_source_ctx = sMfLogDataSourceCtx {...}
0x20000258	triggered = _Bool 0
0x2000025c	num_logs = size_t 0
0x20000260	trigger_time = sMemfaultCurrentTime {...}
0x200004e0	s_log_buf_storage = uint8_t[512] {...}
0x200004e0	* = uint8_t ""
0x200006fc	backend_cb_log_backend_uart = log_backend_control_block {...}
0x20000704	log_output_uart_control_block = log_output_control_block {...}
0x20000714	log_format_current = uint32_t 0
0x20000dcc	s_memfault_ram_logger = sMfRamLogger {...}
0x2000175c	s_mfit_log_buf_storage = uint8_t[512] {...}
0x2000175c	* = uint8_t "vent at 0,19 state on R<inf> mfit: gatt: kscan_pr...
0x2000195c	backend_cb_log_backend_mfit = log_backend_control_block {...}
0x20001964	s_memfault_log_backend = const log_backend * {...}
0x20001968	s_log_output_mfit_control_block = log_output_control_block {...}

The crash and the context

Sometimes just the code isn't enough.

Understand device behavior and condition at exactly the time of the crash.

Connect metrics with traces for precise insights.

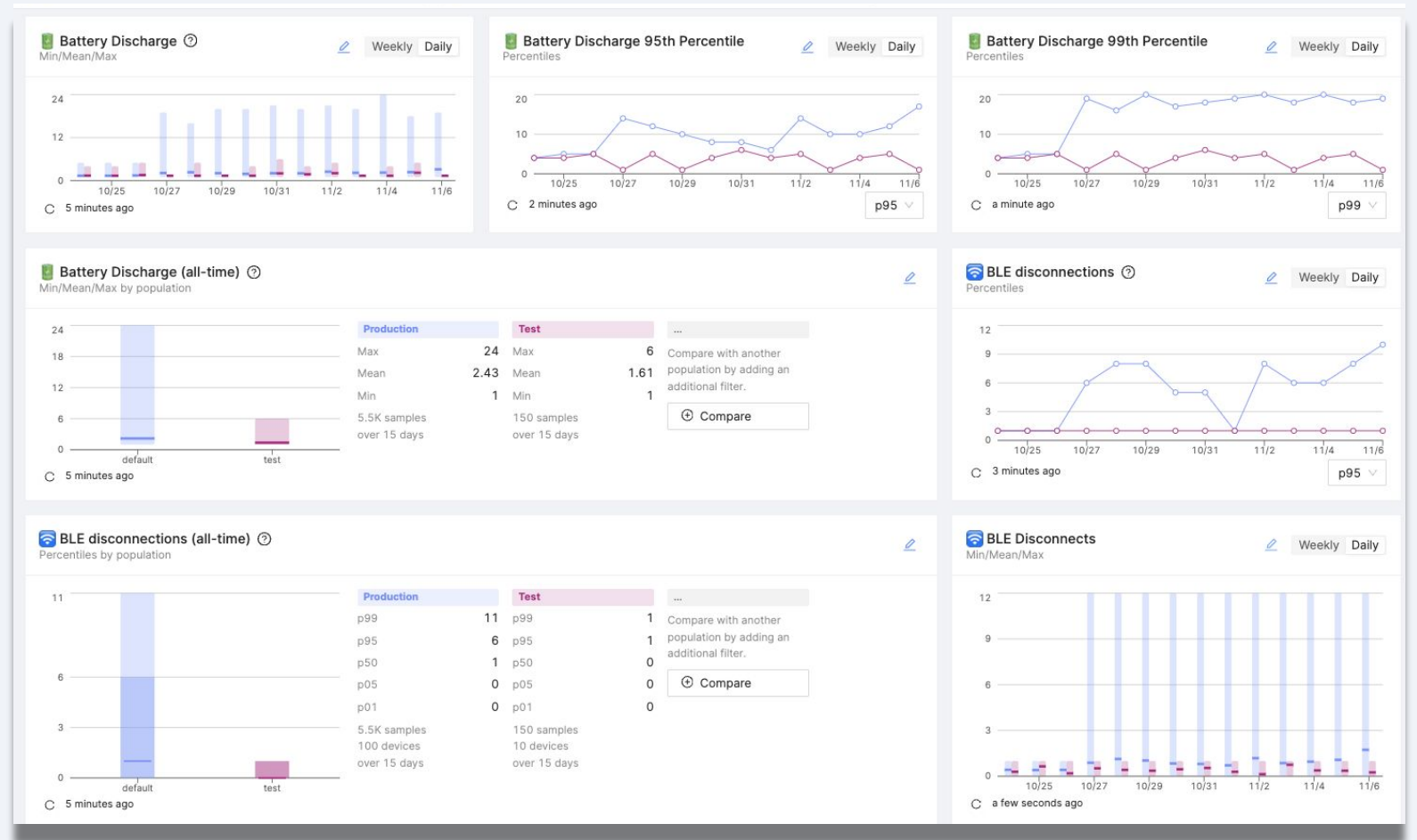


Fleet Wide Visibility

Start tracking health and performance across all your devices

Understand what normal looks like, build informed KPIs for performance and quality

Identify poor performance before it causes a problem



Data to make decisions

This data does so much more than just help you debug.

It has power across your whole organization.

Embedded Engineer

- What are the top priority issues based on prevalence and severity
- Is this new software version stable enough for a full release

Product Manager

- How are my customers using my product
- Where are the opportunities to add new features or make improvements

Head of Engineering

- Are my products highly reliable
- Where should I focus my teams resources
- Is my teams work making our devices better over time

Embedded Development Maturity

Reactive
and no visibility

I don't know what I don't know.

Reactive
but with visibility

I know that I don't know, but I am still behind.

I can collect traces and logs on demand.

Proactive
with insight

I can see what's coming and can take action ahead of time.

I am actively monitoring health metrics fleetwide.

Proactive and Strategic

I am able to make critical decisions based on the information I am collecting and can take action fast.

I am integrating this data into decision making.

Transformative

The way I work is completely different from how it used to be.

This is now ingrained in everything we do.



But...my device is resource constrained

Embedded devices are unique

	Power	Connectivity	Compute
Requirement	Should be as power efficient as possible without sacrificing performance required to complete its core function.	Must be as reliable as possible and send exactly enough data to complete the required task and no more within an acceptable time frame .	Must be as high performance as possible without jeopardizing the requirements of the other two pillars.
Impact	Customer experience	Customer experience	Customer experience
	Device lifespan (if battery)	Functionality	Functionality
	OPEX cost	OPEX cost	CAPEX cost
Memfault	Helps you identify opportunities to improve power efficiency without adding any additional load.	Helps you increase reliability without requiring significantly more bandwidth or connection time.	Has no performance overhead and helps you maximize the return on your CAPEX investment.

Built for embedded devices



Power

Has no measurable battery impact.

Collects full system data to help you improve efficiency.

✓ Works on your low power device

Connectivity

Buffers data on device and sends when connectivity becomes available.

Can chunk data into very small packet sizes (≥ 9 bytes).

✓ Works with any connectivity set-up

Compute

Does not add any additional performance load to system.

Lightweight on device profile
4.5kB FLASH, 1.5kB RAM

✓ Works on highly constrained devices



How Latch build ultra-reliable Products using Memfault

We make spaces better places to live, work, and visit.

Designed to work together seamlessly. Our award-winning products are stunning and provide flexible connection options, allowing for easy installation and working uninterrupted even when internet connectivity is not available. Grant and revoke access remotely, and enhance your buildings' operations, all while relying on cutting-edge encryption technology and user experience.



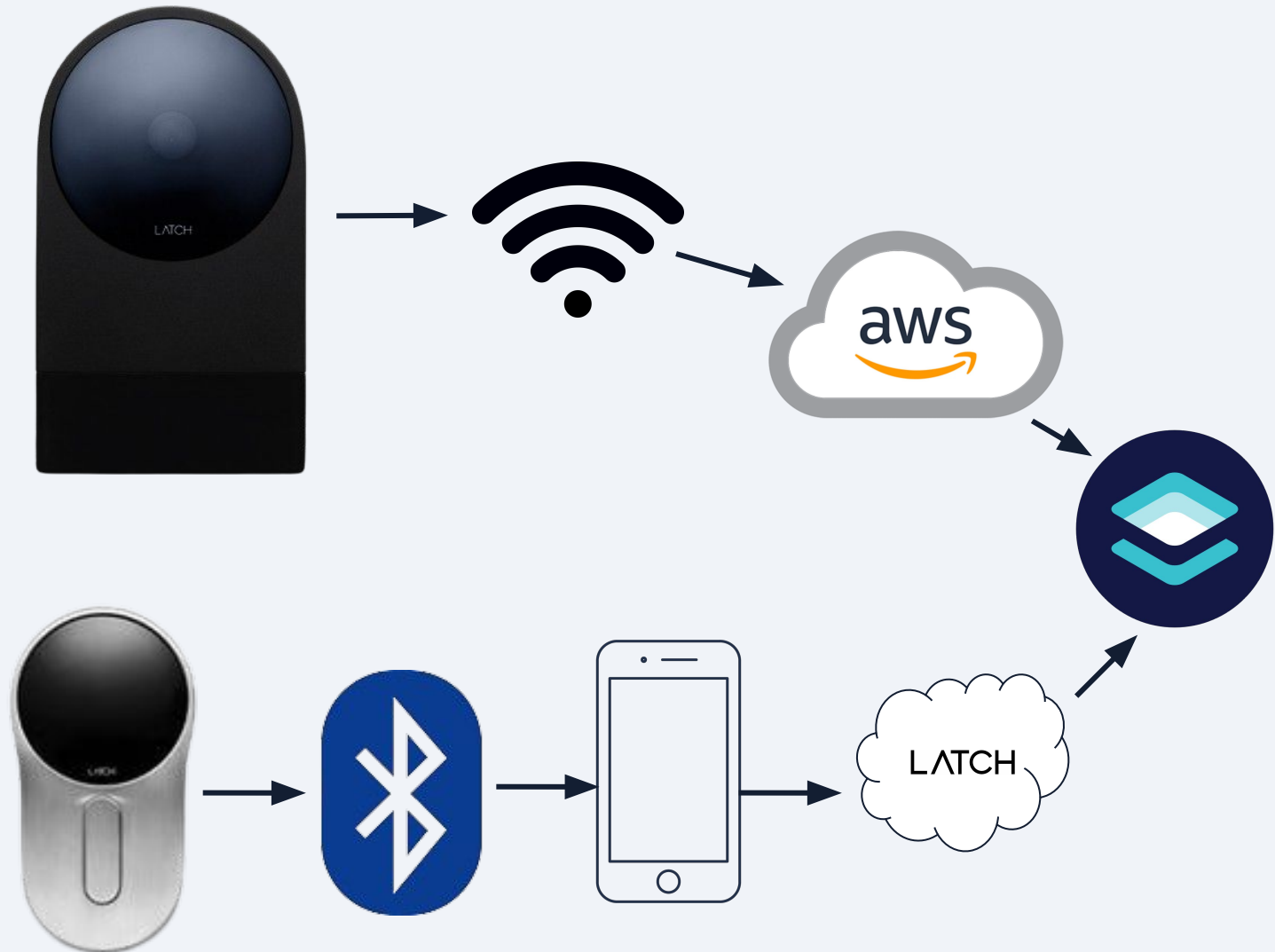
Latch Intro



- Latch's core products provide smart access to common and private spaces.
- Secure and reliable access to these spaces is critical.
- Memfault helps Latch proactively monitor both internal and external fleets at all stages of development.

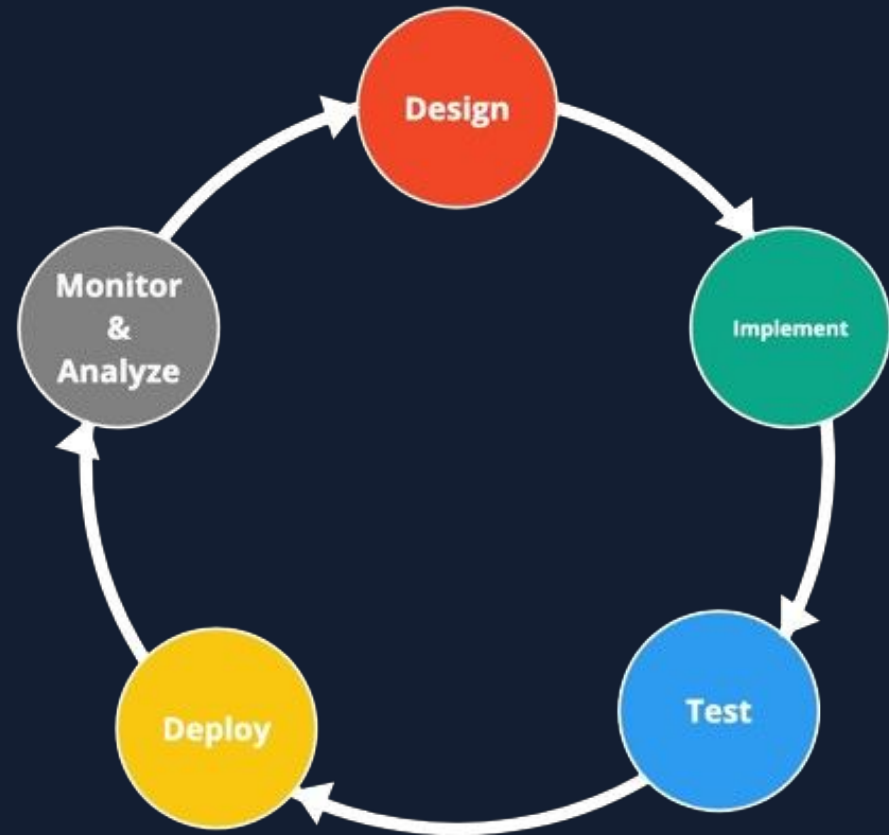
Latch <> Memfault

- Latch has deployed Memfault to 10s of thousands of devices.
- Devices leverage multiple transports for data upload.
- Utilizing almost all Memfault capabilities
 - Coredumps
 - Traces
 - Heap tracing
 - Heartbeat metrics
 - Compact logs
 - Custom Data Recordings and more...



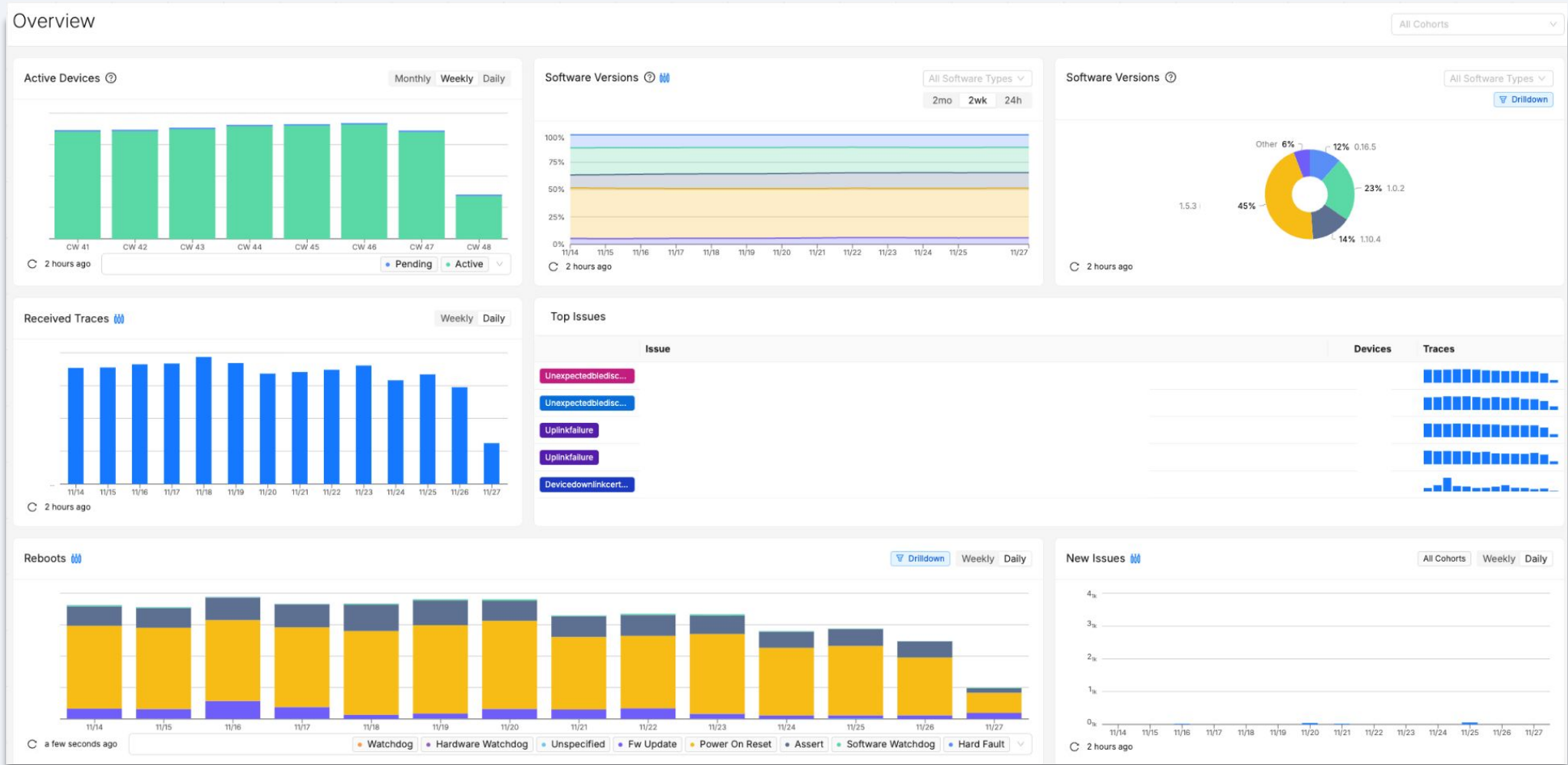
Latch <> Memfault

- Memfault is a first-class citizen in our design and development process.
 - All new feature designs are analyzed for opportunities to add observability via Memfault.
 - Bugfixes always beg the question, “Can I prevent or catch this next time with Memfault?”
 - Weekly/Bi-weekly “*Memfault Data Review*”



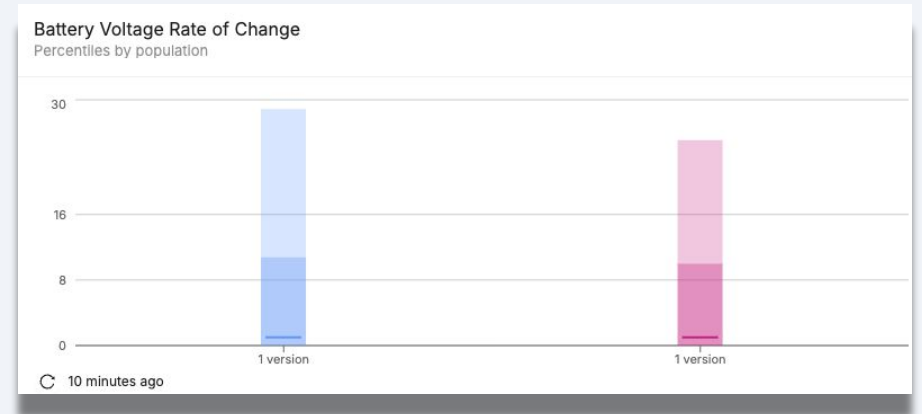
Memfault Data Review

Latch's "Memfault Data Review" is a dedicated effort to proactively monitor and analyze our internal and external fleets for potential performance and reliability issues.

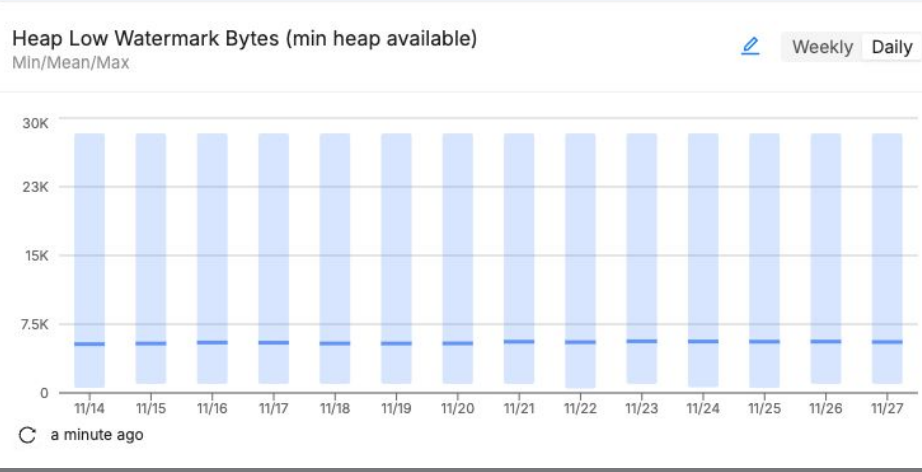


Memfault Data Review

Flexible metrics monitoring helps us identify fleetwide as well as device-specific trends.



	1 version	1 version
p99	28.87	25.07
p95	10.77	10
p50	1	1
p05	0	0
p01	0	0



Metrics

All Cohorts 1.5.3 All Cohorts 1.10.4 Compare

Access - ANFC Unlocks Total

Min/Mean/Max

2 hours ago

Access - BLE Unlocks Total

Min/Mean/Max

2 hours ago

Access - ANFC Unlocks Total

Sum

2 hours ago

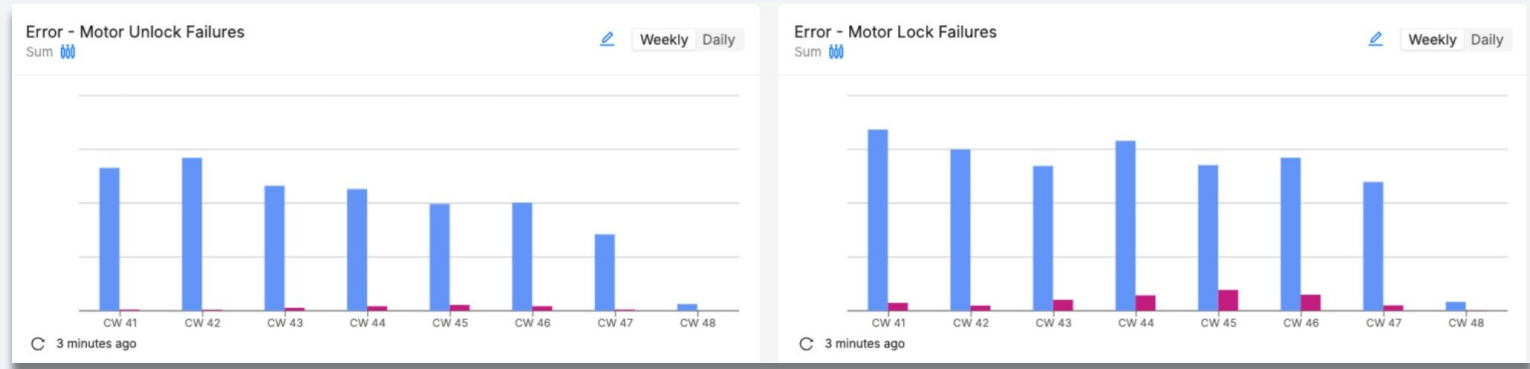
Access - BLE Unlocks Total

Sum

2 hours ago

Real World Examples

- Motor Errors



11/26/2023 1:33:12 PM 2218823162 Assert (ef6cee78) at PC = 00030E66, LR = 0002D5E7
11/25/2023 10:35:32 PM 9025089456 Assert (ef6cee78) at PC = 00030E66, LR = 0002D5E7

- Crash Debugging

Assert at `tlsEstablishConnection`

Device: 5055454835
Cohort: default
Software: 2.0.1 (HS_STM)
Hardware: REV_D.2

State: Logs

Threads

- netComms (2) **RUNNING**
 - 0 prv_fault_handling_assert in .../memfault_fault_handling_arm.c at line 555
 - 1 memfault_fault_handling_assert in .../memfault_fault_handling_arm.c at line 567
 - 2 tlsEstablishConnection in .../drivers/ethernet.c at line 650
 - 3 ethernet_tls_connect in .../drivers/ethernet.c at line 453
 - 4 mqtt_client_connect in .../mqtt_client.c at line 133
 - 5 networkCommsThread in .../tasks/network_comms.c at line 229
 - 6 prvPortStartFirstTask in .../GCC/ARM_CM3/port.c at line 242
- IDLE (3)
- TCP/IP Stack (4)
- TCP/IP Stack (5)
- Tmr Svc (6)

Exceptions: Registers & Locals, Globals & Statics, Heap, ISR Analysis, MPU

Assert at `tlsEstablishConnection`

Analysis

Configurable Fault (i.e UsageFault, BusFault, MemManage) escalated to HardFault

Assert was triggered by software

Fault Register	Value	Hex Value
CFSR	65536	0x00100000
HFSR	1073741824	0x40000000
SHCSR	0	0x00000000

Thank you! Here are resources

- ◇ Learn more at www.memfault.com and www.latch.com
- ◇ Case Study: [Unlocking the Value of Embedded Observability at Latch](#)
- ◇ Embedded engineering community & blog at <https://interrupt.memfault.com>



Tyler Hoffman

Co-Founder & Head of
Developer Experience,
Memfault



Dave Webster

Engineering Manager,
Latch

Q&A



Memfault



LATCH[®]