Zephyr® Project

Developer Summit 2022

June 8-9, 2022

Mountain View, CA + Virtual

Zephyr® Project
Developer Summit 2022

June 8-9, 2022
Mountain View, CA + Virtual

Connecting Zephyr Logging to the Cloud over Constrained Channels

Marcin Niestrój, Golioth

ABOUT ME

- 10 years in embedded space
- 4 years on Zephyr
- Contributor to networking stack
- Golioth SDK, based on Zephyr

WHAT WE WILL TALK ABOUT

- Logging subsystem in Zephyr
- Golioth SDK, which is based on Zephyr
- Golioth cloud services (logging backend, LightDB, ...)
- CoAP (Constrained Application Protocol)
- CBOR (Concise Binary Object Representation)

MOST INTERESTING THINGS TO LEARN FROM THIS TALK

- How to access Zephyr logs without physical access to IoT device
- How do log messages look like when encoded into CBOR and sent over CoAP

LOGGING SUBSYSTEM IN ZEPHYR

- Logs are generated using dedicated macros:

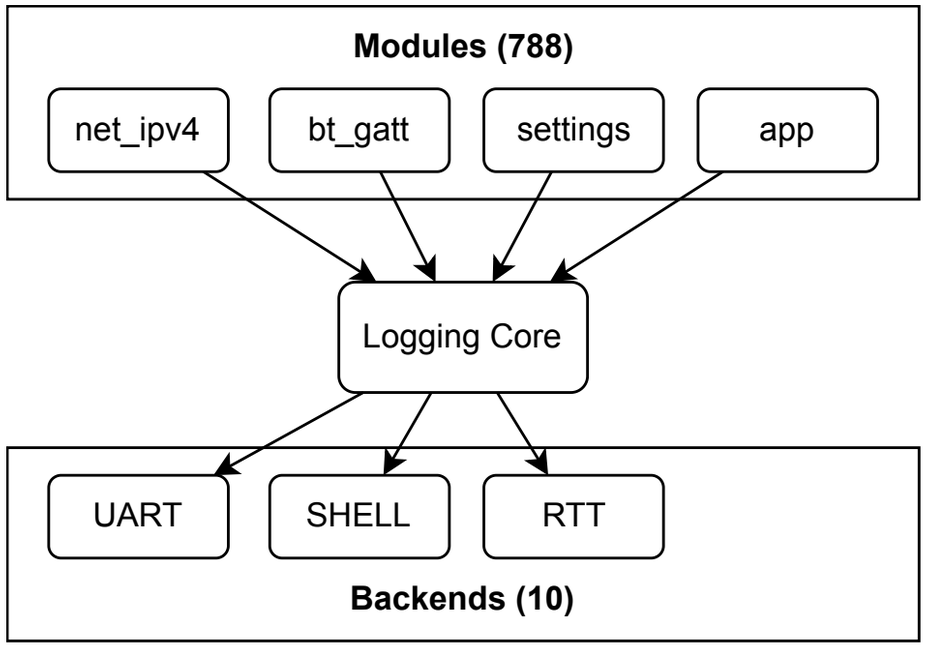
```
LOG_<level>(fmt, ...)
LOG_HEXDUMP_<level>(ptr, len, label)
```

  where <level> can be: ERR, WRN, INF, DBG

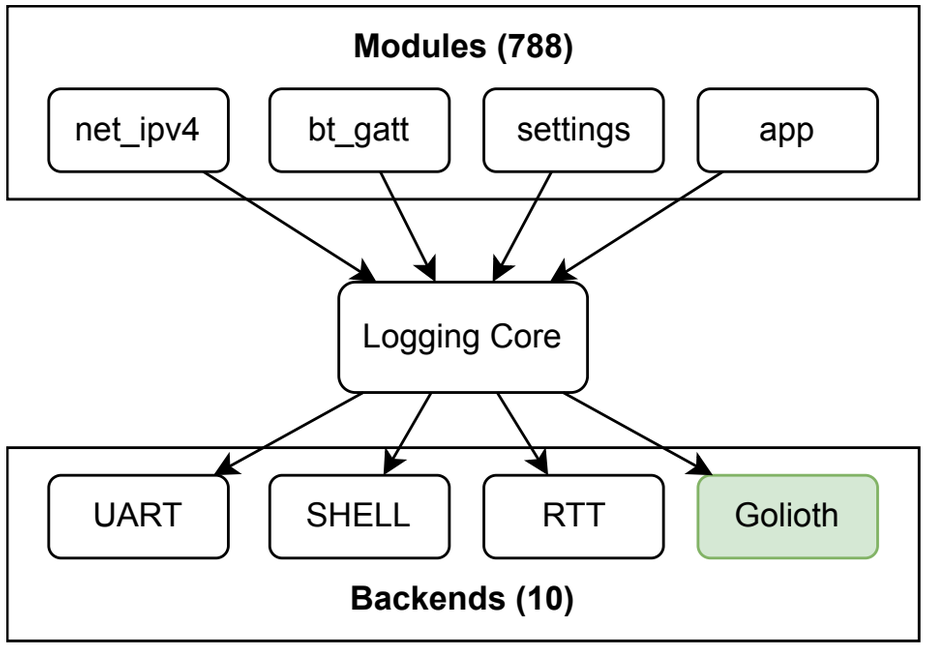
- Immediate and deferred mode
- Configurable log levels via Kconfig
- Multiple backends supported (UART, shell, RTT, remote syslog, ...)
- Supports per module and per backend filtering
- And has many more features ...

LOGGING HIGH-LEVEL MESSAGE FLOW

LOGGING HIGH-LEVEL MESSAGE FLOW



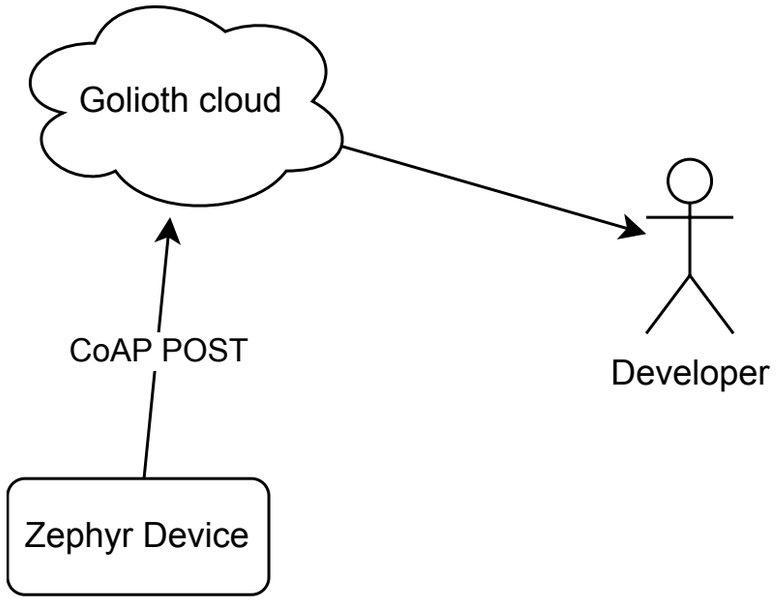
LOGGING HIGH-LEVEL MESSAGE FLOW



GOLIOTH

- Cloud solution for constrained IoT devices
- MCU friendly protocols: CoAP, MQTT
- Secure by design: DTLS, TLS
- JSON and CBOR
- Multiple services: DFU, logging, LightDB, LightDB stream, ...

GOLIOTH LOGGING HIGH-LEVEL FLOW



- Device to cloud: CoAP POST (CBOR payload) over DTLS
- Cloud to user: CLI or WebUI

GOLIOTH LOGGING BACKEND

- It is all about sending log messages over network
- But what is the log message in the first place?

LOG MESSAGE

- Text string, which provides most information
- But it contains important metadata as well:
  - Timestamp
  - Severity (err, wrn, inf, dbg)
  - Module/component name
  - Message counter / drop counter

LOGGING OUTPUT TYPE

TEXT

- Simple implementation
- Doesn't require specialized tool to read
- Good fit for stream based transport (e.g. UART, TCP)
- Requires to keep order if data is written in chunks
- No defined structure makes output impossible to corrupt (apart from losing data)

DICTIONARY

- More complex implemenation
- Requires tool that understands dictionary structure
- Good fit for datagram based transport (e.g. UDP, CoAP)
- Out of order delivery is not an issue (if including message number)
- Easy to encode metadata (e.g. message log level)
- Dedicated tool allows more efficient filtering based on metadata (e.g. module, log level)

OBJECT (LOG MESSAGE) REPRESENTATION

JSON

- Text based
- Extensible (JSON structures)
- Consumes more network bandwidth
- Requires more processing power
- Encoding binary data adds bulk (base64 encoding)

CUSTOM BINARY

- Not standard (requires dedicated tools)
- Not extensible by default
- Possibly low overhead
- Possibly simple to prototype on embedded platform
- Needs dedicated tool to parse generated logs

CBOR

- Binary based
- Extensible (CBOR maps)
- Concise decoding
- Designed for Internet of Things
- No schema needed

WE KNOW WHAT TO SEND

- Text message
- Metadata (timestamp, severity level, ...)
- Packed efficiently into CBOR

WHERE AND HOW SHOULD WE SEND IT?

- TCP? UDP?
- HTTP? MQTT? CoAP?

APPLICATION PROTOCOL

HTTP

- Text based
- HTTP headers consume a lot of bandwidth
- HTTP/1.1 creates one TCP connection per request
- More bandwidth, more processing power

MQTT

- Binary based
- Works on top of TCP (stream based)
- Small packet header - easy on network usage
- Publish/subscribe (broker) design does not introduce any value
- Specifying content format (JSON vs CBOR) needs to be done in MQTT message payload

COAP

- Binary based
- Works on top of UDP (datagram based)
- May work on top of TCP (CoAP extension)
- Small packet header - easy on network usage
- Request/response model works great for other device services (like LightDB)
- Specifying content format (JSON vs CBOR) is part of CoAP

OSI LAYERS UTILIZED BY GOLIOTH LOGGING BACKEND

- CoAP (carrying CBOR payload)
- DTLS
- UDP

SAMPLE LOG MESSAGE OVER THE WIRE

Zephyr application code:

```
LOG_MODULE_REGISTER(app, LOG_LEVEL_DBG);
...
void main() {
  ...
  LOG_WRN("Warn: %d", counter);
  ...
}
```

CoAP packet sent over network:

```
58 02 b0 64 74 2a d6 76  ff f4 70 b2 b4 6c 6f 67
73 11 3c ff a5 66 75 70  74 69 6d 65 1a 03 5c e6
a0 66 6d 6f 64 75 6c 65  63 61 70 70 65 6c 65 76
65 6c 64 77 61 72 6e 65  69 6e 64 65 78 19 01 6c
63 6d 73 67 68 57 61 72  6e 3a 20 31 31
```

```
58 02 b0 64 74 2a d6 76  ff f4 70 b2 b4 6c 6f 67
73 11 3c ff a5 66 75 70  74 69 6d 65 1a 03 5c e6
a0 66 6d 6f 64 75 6c 65  63 61 70 70 65 6c 65 76
65 6c 64 77 61 72 6e 65  69 6e 64 65 78 19 01 6c
63 6d 73 67 68 57 61 72  6e 3a 20 31 31
```

COAP HEADER

Version: 1

Type: Non-Confirmable (1)

Token Length: 8

Code: POST (2)

Message ID: 45156

Token: 742ad676fff470b2

Uri-Path: logs

Content-Format: application/cbor

```
58 02 b0 64 74 2a d6 76  ff f4 70 b2 b4 6c 6f 67
73 11 3c ff a5 66 75 70  74 69 6d 65 1a 03 5c e6
a0 66 6d 6f 64 75 6c 65  63 61 70 70 65 6c 65 76
65 6c 64 77 61 72 6e 65  69 6e 64 65 78 19 01 6c
63 6d 73 67 68 57 61 72  6e 3a 20 31 31
```

COAP HEADER

Version: 1

Type: Non-Confirmable (1)

Token Length: 8

Code: POST (2)

Message ID: 45156

Token: 742ad676fff470b2

Uri-Path: logs

Content-Format: application/cbor

```
58 02 b0 64 74 2a d6 76  ff f4 70 b2 b4 6c 6f 67
73 11 3c ff a5 66 75 70  74 69 6d 65 1a 03 5c e6
a0 66 6d 6f 64 75 6c 65  63 61 70 70 65 6c 65 76
65 6c 64 77 61 72 6e 65  69 6e 64 65 78 19 01 6c
63 6d 73 67 68 57 61 72  6e 3a 20 31 31
```

COAP HEADER

Version: 1

Type: Non-Confirmable (1)

Token Length: 8

Code: POST (2)

Message ID: 45156

Token: 742ad676fff470b2

Uri-Path: logs

Content-Format: application/cbor

```
58 02 b0 64 74 2a d6 76  ff f4 70 b2 b4 6c 6f 67
73 11 3c ff a5 66 75 70  74 69 6d 65 1a 03 5c e6
a0 66 6d 6f 64 75 6c 65  63 61 70 70 65 6c 65 76
65 6c 64 77 61 72 6e 65  69 6e 64 65 78 19 01 6c
63 6d 73 67 68 57 61 72  6e 3a 20 31 31
```

COAP HEADER

Version: 1

Type: Non-Confirmable (1)

Token Length: 8

Code: POST (2)

Message ID: 45156

Token: 742ad676fff470b2

Uri-Path: logs

Content-Format: application/cbor

```
58 02 b0 64 74 2a d6 76  ff f4 70 b2 b4 6c 6f 67
73 11 3c ff a5 66 75 70  74 69 6d 65 1a 03 5c e6
a0 66 6d 6f 64 75 6c 65  63 61 70 70 65 6c 65 76
65 6c 64 77 61 72 6e 65  69 6e 64 65 78 19 01 6c
63 6d 73 67 68 57 61 72  6e 3a 20 31 31
```

COAP HEADER

Version: 1

Type: Non-Confirmable (1)

Token Length: 8

Code: POST (2)

Message ID: 45156

Token: 742ad676fff470b2

Uri-Path: logs

Content-Format: application/cbor

```
58 02 b0 64 74 2a d6 76  ff f4 70 b2 b4 6c 6f 67
73 11 3c ff a5 66 75 70  74 69 6d 65 1a 03 5c e6
a0 66 6d 6f 64 75 6c 65  63 61 70 70 65 6c 65 76
65 6c 64 77 61 72 6e 65  69 6e 64 65 78 19 01 6c
63 6d 73 67 68 57 61 72  6e 3a 20 31 31
```

COAP HEADER

Version: 1

Type: Non-Confirmable (1)

Token Length: 8

Code: POST (2)

Message ID: 45156

Token: 742ad676fff470b2

Uri-Path: logs

Content-Format: application/cbor

```
58 02 b0 64 74 2a d6 76  ff f4 70 b2 b4 6c 6f 67
73 11 3c ff a5 66 75 70  74 69 6d 65 1a 03 5c e6
a0 66 6d 6f 64 75 6c 65  63 61 70 70 65 6c 65 76
65 6c 64 77 61 72 6e 65  69 6e 64 65 78 19 01 6c
63 6d 73 67 68 57 61 72  6e 3a 20 31 31
```

COAP HEADER

Version: 1

Type: Non-Confirmable (1)

Token Length: 8

Code: POST (2)

Message ID: 45156

Token: 742ad676fff470b2

Uri-Path: logs

Content-Format: application/cbor

```
58 02 b0 64 74 2a d6 76  ff f4 70 b2 b4 6c 6f 67
73 11 3c ff a5 66 75 70  74 69 6d 65 1a 03 5c e6
a0 66 6d 6f 64 75 6c 65  63 61 70 70 65 6c 65 76
65 6c 64 77 61 72 6e 65  69 6e 64 65 78 19 01 6c
63 6d 73 67 68 57 61 72  6e 3a 20 31 31
```

COAP HEADER/PAYLOAD BOUNDARY

End of options marker: 255

```
58 02 b0 64 74 2a d6 76  ff f4 70 b2 b4 6c 6f 67
73 11 3c ff a5 66 75 70  74 69 6d 65 1a 03 5c e6
a0 66 6d 6f 64 75 6c 65  63 61 70 70 65 6c 65 76
65 6c 64 77 61 72 6e 65  69 6e 64 65 78 19 01 6c
63 6d 73 67 68 57 61 72  6e 3a 20 31 31
```

COAP PAYLOAD (CBOR)

SAMPLE LOG MESSAGE OVER THE WIRE

CBOR (57 BYTES)

```
A5                      # map(5)
   66                   # text(6)
      757074696D65      # "uptime"
   1A 035CE6A0          # unsigned(56420000)
   66                   # text(6)
      6D6F64756C65      # "module"
   63                   # text(3)
      617070            # "app"
   65                   # text(5)
      6C6576656C        # "level"
   64                   # text(4)
      7761726E          # "warn"
   65                   # text(5)
      696E646578        # "index"
   19 016C              # unsigned(364)
   63                   # text(3)
      6D7367            # "msg"
   68                   # text(8)
      5761726E3A203131  # "Warn: 11"
```

JSON EQUIVALENT (77 BYTES)

```
{
  "uptime": 56420000,
  "module": "app",
  "level": "warn",
  "index": 364,
  "msg": "Warn: 11"
}
```

Zephyr application code:

```
LOG_MODULE_REGISTER(app, LOG_LEVEL_DBG);
...
void main() {
  ...
  LOG_WRN("Warn: %d", counter);
  ...
}
```

HEXDUMP LOG VARIANT

```
LOG_HEXDUMP_INF(&counter, sizeof(counter),
                "Counter hexdump");
```

```
[00:00:56.420,000]  app: Counter hexdump
                         0b 00 00 00             |....
```

```
58 02 b0 66 7c 1b 42 65  b2 e5 dc a0 b4 6c 6f 67
73 11 3c ff a6 66 75 70  74 69 6d 65 1a 03 5c e6
a0 66 6d 6f 64 75 6c 65  63 61 70 70 65 6c 65 76
65 6c 64 69 6e 66 6f 65  69 6e 64 65 78 19 01 6e
63 6d 73 67 6f 43 6f 75  6e 74 65 72 20 68 65 78
64 75 6d 70 67 68 65 78  64 75 6d 70 44 0b 00 00
00
```

HEXDUMP LOG VARIANT

```
LOG_HEXDUMP_INF(&counter, sizeof(counter),
                "Counter hexdump");
```

```
[00:00:56.420,000]  app: Counter hexdump
                         0b 00 00 00             |....
```

```
58 02 b0 66 7c 1b 42 65  b2 e5 dc a0 b4 6c 6f 67
73 11 3c ff a6 66 75 70  74 69 6d 65 1a 03 5c e6
a0 66 6d 6f 64 75 6c 65  63 61 70 70 65 6c 65 76
65 6c 64 69 6e 66 6f 65  69 6e 64 65 78 19 01 6e
63 6d 73 67 6f 43 6f 75  6e 74 65 72 20 68 65 78
64 75 6d 70 67 68 65 78  64 75 6d 70 44 0b 00 00
00
```

```
A6                                      # map(6)
   ...
   67                                   # text(7)
      68657864756D70                    # "hexdump"
   44                                   # bytes(4)
      0B000000                          # "\u000b\u0000\u0000\u0000"
```

HEXDUMP LOG VARIANT

```
LOG_HEXDUMP_INF(&counter, sizeof(counter),
                "Counter hexdump");
```

```
[00:00:56.420,000]  app: Counter hexdump
                         0b 00 00 00             |....
```

```
58 02 b0 66 7c 1b 42 65  b2 e5 dc a0 b4 6c 6f 67
73 11 3c ff a6 66 75 70  74 69 6d 65 1a 03 5c e6
a0 66 6d 6f 64 75 6c 65  63 61 70 70 65 6c 65 76
65 6c 64 69 6e 66 6f 65  69 6e 64 65 78 19 01 6e
63 6d 73 67 6f 43 6f 75  6e 74 65 72 20 68 65 78
64 75 6d 70 67 68 65 78  64 75 6d 70 44 0b 00 00
00
```

```
A6                                      # map(6)
   ...
   67                                   # text(7)
      68657864756D70                    # "hexdump"
   44                                   # bytes(4)
      0B000000                          # "\u000b\u0000\u0000\u0000"
```

RETRIEVING LOG MESSAGE

Zephyr application code (`void main()`)

```
LOG_DBG("Debug info! %d", counter);
```

Output from UART shell

```
[06:49:09.000,000] <dbg> app: main: Debug info! 4900
```

Output from `goliothctl`

```
$ goliothctl logs listen
[2022-05-26T00:54:13Z] level:DEBUG  \
    module:"app"  message:"Debug info! 4900"  \
    device_id:"6033cc457016b281d671df53" \
    metadata:"{"func":"main","index":14732,"uptime":24549000000}"
```

COMBINING WITH JQ

Output from UART shell

```
[06:49:09.000,000] <dbg> app: main: Debug info! 4900
```

Output from `goliothctl`

```
$ goliothctl logs listen
[2022-05-26T00:54:13Z] level:DEBUG  \
    module:"app"  message:"Debug info! 4900"  \
    device_id:"6033cc457016b281d671df53" \
    metadata:"{"func":"main","index":14732,"uptime":24549000000}"
```

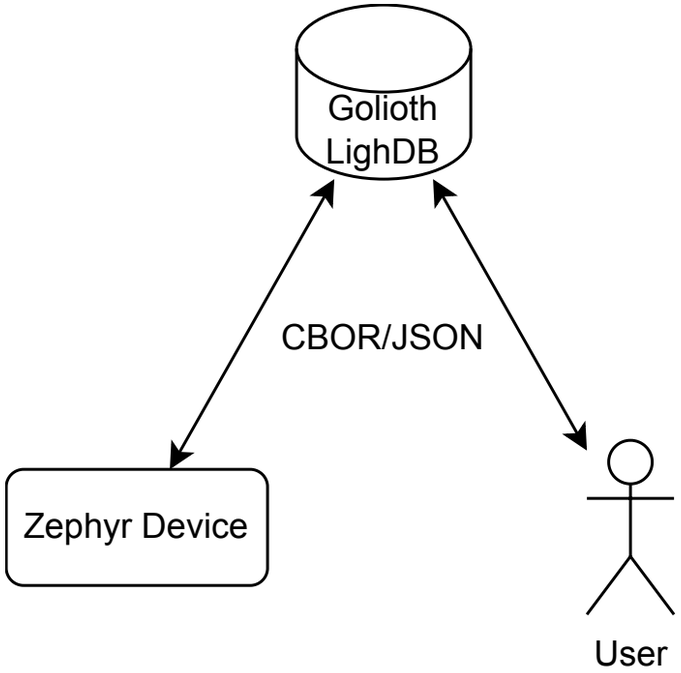
Output from `goliothctl` + `jq`

```
$ goliothctl logs listen --json | jq -r '"[\(.metadata.uptime/1000000|strftime("%H:
[06:49:09] <DEBUG> app: main: Debug info! 4900
```

WHEN SENDING ALL MESSAGES IS A PROBLEM

- What if we cannot afford to send all messages?
- Logging filters and Golioth LighDB come handy

GOLIOTH LIGHTDB



- Simple database accessible by Zephyr device **and** user

LIGHTDB - EXAMPLE CONTENTS

```
{
    "logs": {
        "app": "dbg"
    }
}
```

- Set module "app" to log severity level "dbg"

LIGHTDB - DEVICE PERSPECTIVE

- Resource observable using CoAP OBSERVE - notifications on each change in DB
- Database contents are downloaded as CBOR map
- Configuration is applied using `log_filter_set()`

```
log_filter_set(log_backend_golioth_get(), CONFIG_LOG_DOMAIN_ID,
    module_id, level_id);`
```

LIGHTDB - USER PERSPECTIVE

Command format to change logging level

```
goliothctl lightdb set <DEVICE> /logs/<MODULE> -b \"<LEVEL>\"
```

Example: change "app" module logging level to "dbg"

```
goliothctl lightdb set nrf52 /logs/app -b \"dbg\"
```

LOGGING LEVEL CONFIGURATION

Change logging level to DBG

```
$ goliothctl lightdb set nrf52 /logs/app -b \"dbg\"
"dbg"
$ goliothctl lightdb get nrf52 /logs
{"app":"dbg"}
$ goliothctl logs listen --json | jq -r '"[\(.metadata.uptime/1000000|strftime("%H:
[17:53:23] <INFO> app: null: Counter hexdump NzIAAA==
[17:53:23] <DEBUG> app: func_1: Log 1: 12855 null
[17:53:23] <DEBUG> app: func_2: Log 2: 12855 null
[17:53:23] <WARN> app: null: Warn: 12855 null
[17:53:23] <DEBUG> app: main: Debug info! 12855 null
[17:53:23] <ERROR> app: null: Err: 12855 null
```

LOGGING LEVEL CONFIGURATION

Change logging level to WRN

```
$ goliothctl lightdb set nrf52 /logs/app -b \"wrn\"
"wrn"
$ goliothctl lightdb get nrf52 /logs
{"app":"wrn"}
$ goliothctl logs listen --json | jq -r '"[\(.metadata.uptime/1000000|strftime("%H:
[17:56:48] <ERROR> app: null: Err: 12896 null
[17:56:48] <WARN> app: null: Warn: 12896 null
```

WHY IS LOGGING OVER INTERNET SO COOL?

- Framework is already built into Zephyr
- There are already almost 800 registered logging modules in Zephyr (no need to modify them to use custom logging APIs)
- Same tool used for early development can be used for early prototypes field-testing and even deployed devices
- The only difference is transport channel (logging backend)
- Easy for user to control which messages are sent by device

ROOM FOR FUTURE IMPROVEMENTS

- (Optional) reliability / resend logic
- Packing multiple log messages into single datagram
- Collect messages and send later all at once
- Configuration of collected/sent metadata (e.g. via LightDB)
- Sending module id (integer) instead of module name (text)

USEFUL LINKS

- https://docs.zephyrproject.org/3.0.0/reference/logging/index.html
- https://golioth.io/
- https://github.com/golioth/golioth-zephyr-sdk/tree/zds2022-logging
- CBOR RFC: https://datatracker.ietf.org/doc/html/rfc8949
- CoAP RFC: https://datatracker.ietf.org/doc/html/rfc7252