Timestamp format

Summary

Issue

We want to be able to track when things happen by using timestamps and by using a consistent timestamp format that works well across all our systems and third-party systems.

We interact with systems that have different timestamp formats:

- JSON messages do not have a native timestamp format, so we need to choose how to convert a timestamp to a string, and convert a string to a timestamp, i.e. how to serialize/deserialize.
- Some applications are set to use local time, rather than UTC time. This can be convenient for projects that must adjust to local time, such as projects that trigger events that are based on local time.
- Some systems have different time precision needs and capabilities, such as using a time resolution of seconds vs. milliseconds vs. nanoseconds. For example, the Linux operating system `date` command uses a default time precision of seconds, whereas the Nasdaq stock exchange wants a default time precision of nanoseconds.

Decision

We choose the timestamp standard format ISO 8601 with nanosecond precision, specifically "YYYY-MM-DDTHH:MM:SS.NNNNNNNNNZ".

The format shows the year, month, day, hour, minute, second, nanoseconds, and Zulu time zone a.k.a. UTC, GMT.

Status

Decided.

Details

Assumptions

We need to handle these timestamp text strings, to convert from a timestamp to a string (a.k.a. serialize) and convert from a string to a timestamp (a.k.a. deserialize).

We want a format that is generally easy to use, easy to convert, and easy for a person to read.

We want compatibility with a wide range of external systems that we cannot control, such as analytics systems, database systems, financial systems.

Constraints

Some systems have time precision limitations. For example, the macOS operating system `date` command can print time precision in seconds, but not in nanoseconds.

Positions

We considered a range of options:

- Unix epoch i.e. one incrementing number.

- Terse text format "YYYYMMDDTHHMMSSNNNNNNNNN".

- Using a local time zone vs. the UTC time zone.

Argument

For typical use, we value easy to read/write by humans, more than raw speed/size.

For typical use, we want a format that works fine in machine systems, and also works well manually, such as writing sample data, reading JSON output, grepping a log file, etc.

For atypical use, such as high performance computing, we expect we'll want to optimize any text format we choose by converting the text to a faster format, such as a programming language's built-in date object type. So the text format doesn't matter much for HPC.

Implications

Our various text systems and time systems will converge on this format.

Related

Related decisions

We may want a fast/easy way to also track time deltas a.k.a. durations. These are easy with Unix epoch timestamps.

Related requirements

We may want to adjust our decision e.g. if we have a related requirement for a specific kind of logging message stamp, such as for Splunk, Sumo, ELK, etc.

Related artifacts

Language formatters and parsers:

- date-fns: Modern JavaScript date utility library
- Crono: date and time library for Rust

Rosetta Code examples:

- System time
- Data format
- Show the epoch

SixArm examples:

- now_string

Related principles

Easily reversible. We can change pretty easily to a different format, such as Unix epoch.

Defer premature optimization. For typical use we don't care much about a handful of extra characters such as a format that uses dashes and colons.

Notes

Add notes here.