What is a Client Session?

In the context of client observability, **a session is a contiguous, uniquely referenceable period of time during the running of an application that can be associated with telemetry recorded during its duration**. Conceptually, a data model that fully describes a session comprises the following:

1. **Core data**: a session can be identified uniquely by an ID, and it should have start and end timestamps that denote the period during which the session was active.
2. **Extended data and metadata**: a session can contain mutable and immutable data that describes itself (e.g. background/foreground userId) and its runtime environment (e.g. network connection status). This metadata can be transitively applied to the telemetry associated with the session.
3. **Relationship to signals**: a session can be associated with telemetry explicitly (e.g. a log has the attribute *session.id* that denotes the session it was recorded in), or implicitly (e.g. if telemetry overlaps with the time in which a session is active).

Beyond the data model, implementations of sessions can operate differently along a number of dimensions, including but not limited to the following:

- What triggers a session to start or end?
- Is there a maximum or minimum to session duration?
- How is a session associated with telemetry? Explicitly or implicitly?
- When an observed client is active, must there be an active session?
- Can sessions overlap?
- Is there a limit to how much telemetry can be associated with a session?
- And so on…

Standardizing Sessions in OTel

Currently, sessions are represented in OTel via semantic conventions and Events. Notably, the session.id attribute can be used to associate a signal with a particular session, while the session.start and session.end events capture the starting and ending of sessions. This covers some key aspects of the data model, but much is still missing, namely a common understanding of how sessions can operate, and a way (or object?) to attach additional data and metadata to.

Right now, sessions mostly exist so you can group and filter telemetry by a particular unique ID to find out all the telemetry associated with that given session. And the only way to get information about a session is to examine the telemetry associated with them or look at the data on the start and end events. This is suboptimal as there's no guarantee of data integrity between the different signals that a session can be associated with. At best, this can be

considered a workaround, but it's probably more fair to say that there is no good way to associate data and metadata to a session in OTel beyond the core data found in the events.

There is also nothing in the OTLP spec that precisely defines the concept of a session or how it should operate within the ecosystem. Similarly, there is nothing in the OTel APIs that allow users to work with sessions beyond setting the attributes with the right attribute names. The lack of standardization means that we are leaving it up to vendors and users to decide how sessions work and how SDK users should interact with them. This means that while semantic conventions offer a basic description of a session as it applies to signals, implementations of sessions are bespoke, and are thus not portable from vendor to vendor based on how they are defined now. This goes against the ethos of OTel and should be remedied.

Goals

Improving the state of support for sessions in OTel can be broken down into the following high-level goals:

- Defining the canonical representation of sessions in the OTel data model
    - Building on what current exists, this would consist of a richer set of semantic conventions and a way to persist session data and (mutable) metadata independent of telemetry and events
    - Ideally, the first version of this would not require any modifications to OTLP so it can be rolled out as an additive feature on top of existing tooling
- An API that makes it easy to associate OTel signals with sessions, abstracting implementation detail away from the SDK user
- An implementation of the API that balances the need to offer flexibility in how sessions operate to support the myriad vendor and client use cases, but well-defined enough to offer enough of a structure to allow instrumentations to be portable.
- The starting and ending of sessions should be directly observable from outside the application without trying to deduce them from recorded telemetry or whatever data might be persisted for a session. The existing session start and end events seem like a good starting point to accomplish this goal.
- The design should explicitly support the following client platforms:
    - Web
    - iOS
    - Android
    - React Native

Stretch Goals

There are additional goals that should be considered, if not for the first version, but for future versions. Designs should consider these so as to not prevent any of them from being pursued in the future.

- Support in the API and implementation for long-running sessions that are resilient to unexpected termination and other adverse runtime conditions that make clients unreliable.
- Support in the API and implementation for telemetry associated with a session to be transmitted to the telemetry backend with guarantees about whether more data from a session can be expected.

Non-Goals

To limit the scope of what the project is expected deliver, the following is explicitly out of scope:

- Definition of how sessions work at the Collector level. This effort is focused on the data model and API levels at this point.
- Harmonize the concept of application users with sessions. While sessions and users may be related concepts, this effort will not address how users are represented in OTel.
- Provide solutions to other related concepts that differentiate the modeling of usage of stateful client application execution with that of stateless microservices. While many of these concepts can be referenced by or be related to sessions, creating a data model and API to make working with them easier is not in scope.
- Explicitly call out authenticated vs unauthenticated sessions as a top level concern. The design should allow whether a session is authenticated to be persisted as a property of a session, it will not be elevated and dealt with explicitly at the API level. This will likely be captured as a semantic convention if referenced at all.
- Design that is appropriate to use for backend applications that suffer from some of the limitations being addressed. While there might be overlapping concerns, introducing backend, distributed system use cases will introduce myriad new wrinkles that will greatly increase the complexity of this effort
- Implementation in every language SDK. Not all languages and platforms require the notion of sessions, so languages SDKs where it doesn’t make sense need not be considered.