THIS DOCUMENT IS PUBLIC

Morgan McLean, Bogdan Drutu, Carlos Cortez, Sergey Kanzhelev, and Ted Young are currently coordinating the beta release

Please plus morganmclean@google.com (doc author) into any comments that need attention

We're aiming to take OpenTelemetry beta in March 2020 and generally available in mid to late 2020. Over 2020, we aim to take OpenTelemetry through the following launch stages:

- Beta, which provides almost fully complete metrics, tracing, and context propagation functionality but makes no promises around breaking changes
- GA release candidate(s), which are fully functional and include the GA API (no breaking changes between RC and GA)
- V1.0 GA release, which is effectively the GA release candidates with additional hardening, optimizations, automated tests, etc.

Each OpenTelemetry component (components include each language-specific API + SDK, the OpenTelemetry collector, and projects like Java auto-instrumentation) will progress through these launch stages according to its own timeline, though some components may target the same dates. For example, the Java API + SDK may be in GA RC while PHP is still in alpha or beta.

Beta

We are planning to take a wave of OpenTelemetry components to beta in on or after March 16th 2020, followed by an official announcement on March 30th. These components include the Java, JavaScript, Python, Go, and .Net APIs and SDKs, along with the OpenTelemetry collector, though more will be added if they're able to meet dates close to these.

The beta will also include subsequent releases that implement spec version 0.4, until each library transitions to GA release candidates.

We've set the following requirements for components to reach beta:

- All spec version <= 0.3 features must be implemented. For APIs and SDKs this includes all 0.3 features for traces, metrics, context propagation, and resource metadata. For the Collector this includes the OpenTelemetry proto format. Smaller changes are already scheduled for the v0.4 and v0.5 versions of the spec, but they may be implemented during the beta. Presumably, beta feedback will create further changes.

- All APIs will attempt to be final, with the goal of not introducing any breaking changes between beta and GA RC. If breaking changes must be introduced between beta and GA RC, they will be small.
- Components must support the OpenTelemetry-native exporter. Components should support exporters for Jaeger, Prometheus, and Zipkin, though it is acceptable for these to be added soon after the component enters beta. The Collector must also include receivers for these formats.
- APIs must include at least one HTTP and gRPC integration, though these are packaged separately. APIs should include at least one SQL integration, and can include a web framework integration (this is a stretch goal).
- Documentation for the following:
    - Setting up exporters for Prometheus and Jaeger.
    - Creating a manual “hello world” client / server app with completely manual instrumentation. Includes Docker containers for Prometheus and Jaeger and explains how to start them.
    - Adding OpenTelemetry to an existing HTTP client / server application that uses an instrumented framework.
    - Setting up and using the Collector
    - Semantic conventions for creating exporters. For example, how to convert from OpenTelemetry span semantics to Jaeger.
    - Generic documentation that introduces readers to traces, spans, metrics, context propagation

Tracking

Each SIG will need to track their progress towards the beta requirements. This should be done through the following process:

1. Create a GitHub issue for every missing beta feature
2. Do one of the following:
    a. Associate each beta issue with a beta milestone (example); *GitHub doesn’t allow multiple milestones per feature, so this won’t be possible if you are already using milestones for another purpose*
    b. Track the outstanding beta issues as links in another issue, and pin this issue to the top of your repository’s issue list (example)

Morgan McLean, Ted Young, and Carlos Cortez will be joining each SIG's weekly meeting to check in on progress, unblock spec-related issues, and brief contributors about this plan.

Announcement

As mentioned above, the first wave of OpenTelemetry libraries will reach beta on March 16th. We're targeting March 30th for an official announcement.

GA

GA Requirements Brainstorm

We expect that the first wave of GA components will include the Java, JavaScript, Python, Go, and .Net APIs and SDKs, along with the OpenTelemetry Collector and Java auto-instrumentation.

GA requirements:

- GA releases of each component will implement specification version 1.0.
    - For APIs and SDKs this includes all 1.0 features for traces, metrics, context propagation, and resource metadata.
    - For the Collector this includes the OpenTelemetry proto format.
    - Audit for spec compliance.
- API packages (core instrumentation APIs) will not take any breaking changes after the 1.0 release.
- Components must support the OpenTelemetry-native exporter, and exporters for Jaeger, Prometheus, and Zipkin. The Collector must also include receivers for these formats.
- Documentation for the following:
    - Setting up exporters for Prometheus and Jaeger.
    - Creating a manual "hello world" client / server app with completely manual instrumentation. Includes Docker containers for Prometheus and Jaeger and explains how to start them.
    - Adding OpenTelemetry to an existing HTTP client / server application that uses an instrumented framework.
    - Setting up and using the Collector in the above scenarios.

    - Generic documentation for creating exporters. For example, how to convert from OpenTelemetry span semantics to Jaeger, or how to convert metrics semantics to Prometheus.
    - Generic documentation that introduces readers to traces, spans, metrics, context propagation
    - A larger sample application with multiple services written in different languages and uses different frameworks (could use Google's Online Boutique, Weaveworks' Sockshop, or others) that is pre-instrumented with OpenTelemetry.
- Automated testing, stable release pipeline, etc.
    - Testing
        - Performance Benchmarks
        - Load testing
        - Integration testing for context propagation
        - Public access to performance, load, and integration Test results
        - Backwards compatibility tests
        - Instrumentation testing
    - Automated release process
        - Publish performance, load, and integration test results with every release
        - Kicking off a new release automatically builds all current code and automatically publishes new artifacts to the appropriate package repositories

Long Term Support Guarantee

- What does 'support' mean?
    - Critical security and bug fixes
    - SDK guaranteed to work with the stable 1.0 API
- Do we want something similar to the Go or Ubuntu long-term supported versions?
    - 1.0 APIs have a backwards compatibility guarantee, but we need to define SDK security and bugfix guarantee to go along with it.