**Open Telemetry**

**What is telemetry data?**

Capturing data is critical to understanding how your applications and infrastructure are performing at any given time. This information is gathered from remote, often inaccessible points within your ecosystem and processed by some sort of tool or equipment. Monitoring begins here. The data is incredibly plentiful and difficult to store over long periods due to capacity limitations — a reason why private and public cloud storage services have been a boon to DevOps teams.

Logs, metrics, and traces make up the bulk of all telemetry data.

Logs are important because you’ll naturally want an event-based record of any notable anomalies across the system. Structured, unstructured, or in plain text, these readable files can tell you the results of any transaction involving an endpoint within your multi cloud environment. However, not all logs are inherently reviewable — a problem that’s given rise to external log analysis tools.

Metrics are numerical data points represented as counts or measures that are often calculated or aggregated over a period of time. Metrics originate from several sources including infrastructure, hosts, and third-party sources. While logs aren’t always accessible, most metrics tend to be reachable via query. Timestamps, values, and even event names can preemptively uncover a growing problem that needs remediation.

Traces are the act of following a process (for example, an API request or other system activity) from start to finish, showing how services connect. Keeping watch over this pathway is critical to understanding how your ecosystem works, if it’s working effectively, and if any troubleshooting is necessary. Span data is a hallmark of tracing — which includes information such as unique identifiers, operation names, timestamps, logs, events, and indexes.

**How does Open Telemetry work?**

OTel is a specialized protocol for collecting telemetry data and exporting it to a target system. Since the CNCF project itself is open source, the end goal is making data collection more system-agnostic than it currently is. But how is that data generated?

The data life cycle has multiple steps from start to finish. Here are the steps the solution takes, and the data it generates along the way:

* Instruments your code with APIs, telling system components what metrics to gather and how to gather them
* Pools the data using SDKs, and transports it for processing and exporting
* Breaks down the data, samples it, filters it to reduce noise or errors, and enriches it using multi-source contextualization
* Converts and exports the data
* Conducts more filtering in time-based batches, then moves the data onward to a predetermined backend

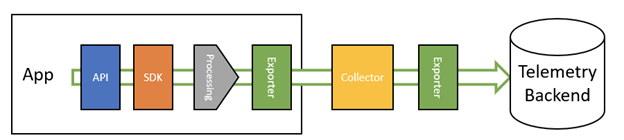
Ingestion is critical to gathering the data we care most about. There are two principal ways to go about this:

* Local ingestion. This occurs once data is safely stored within a local cache. This is common in on-premises or hybrid deployments, where time series data and tags are transmitted to the cloud. Cloud databases excel at storing large volumes of information for later reference, and this data often has business value or privacy restrictions.
* Span ingestion. We can also [ingest trace data](https://www.dynatrace.com/news/blog/the-dynatrace-open-aiops-platform-now-provides-enterprise-grade-observability-for-ingested-opentelemetry-traces/) in span format. Depending on the vendor, this data may be ingested either directly or indirectly. Spans are typically indexed and consist of both root spans and child spans. This data is valuable because it contains key metadata, event information, and more.

These methods are pivotal to the entire pipeline, as the process cannot work without tapping into this information.

**Open Telemetry components**

OTel consists of a few different components as depicted in the following figure. Let’s take a high-level look at each one from left to right:

Open Telemetry Components (Source: Based on [OpenTelemetry: beyond getting started](https://medium.com/opentelemetry/opentelemetry-beyond-getting-started-5ac43cd0fe26" \t "_blank))

APIs

These are core components and language-specific (such as Java, Python, .Net, and so on). APIs provide the basic “plumbing” for your application.

SDK

This is also a language-specific component and is the middleman that provides the bridge between the APIs and the exporter. The SDK allows for additional configuration, such as request filtering and transaction sampling.

In-process exporter

This allows you to configure which backend(s) you want it sent to. The exporter decouples the instrumentation from the backend configuration. This makes it easy to switch backends without the pain of re-instrumenting your code.

Collector

The collector receives, processes, and exports telemetry data. While not technically required, it is an extremely useful component to the Open Telemetry architecture because it allows greater flexibility for receiving and sending the application telemetry to the backend(s).

The collector has two deployment models:

1. An agent that resides on the same host as the application (for example, binary, DaemonSet, sidecar, and so on)
2. A standalone process completely separates from the application

Since the collector is just a specification for collecting and sending telemetry, it still requires a backend to receive and store the data.