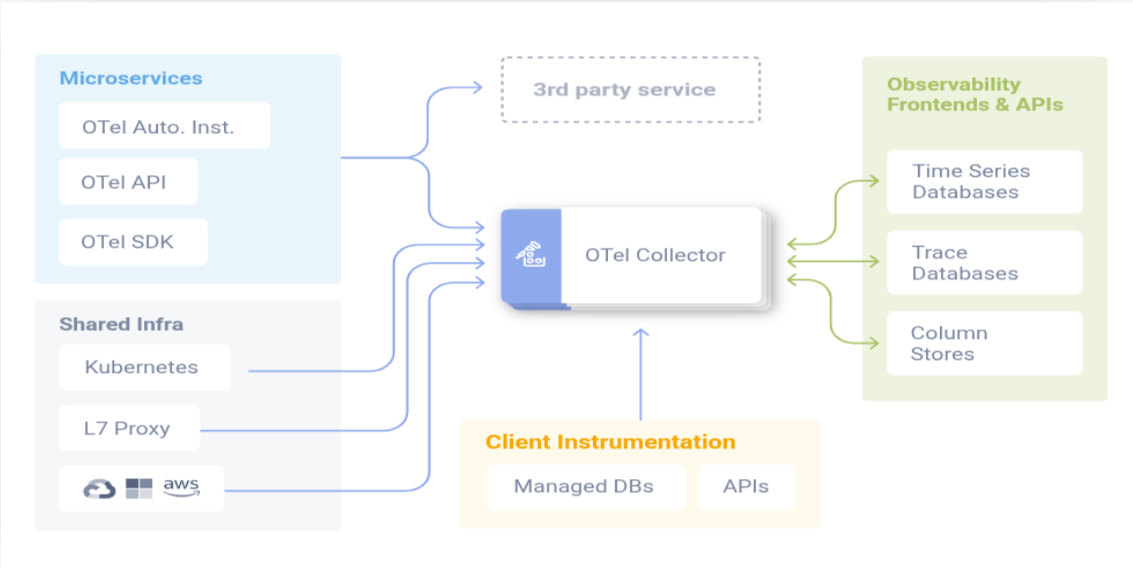
What is OpenTelemetry?

- OpenTelemetry (OTel) is an open source **observability framework** that provides standardized ***APIs, SDKs, and tools*** for collecting and routing telemetry data.
- Created under the Cloud Native Computing Foundation (CNCF) through the merger of the OpenTracing and OpenCensus projects.
- OpenTelemetry provides a consistent, **vendor-neutral format** for instrumenting, generating, gathering, and exporting the three pillars of observability — **metrics, logs, and traces** — to any compatible monitoring backend.



OpenTelemetry enables observability

- **Definition** → Ability to understand system behavior from external outputs (logs, metrics, traces).
- Hazel Weakly's view → "Ask meaningful questions, get useful answers, act effectively."
- Importance → Modern distributed systems are complex, unpredictable, and need deeper visibility.

What is telemetry data?

- Telemetry is the process of gathering and transmitting signals (data) emitted by **instrumentation code** within a system's components.
- Traces, metrics, and logs make up most of all telemetry data.

Traces

- Traces result from 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.
- Traces consist of individual operations called spans, which include unique identifiers, such as operation name, timestamp, context, attributes, events, and status.

Metrics

- Metrics are **numerical data points**, either counts or measures, that systems can calculate or aggregate over time.
- Metrics originate from several sources, including infrastructure, hosts, and third-party sources.
- While logs may not always be accessible, most metrics are readily available via query.
- Timestamps, values, and even event names can preemptively uncover a growing problem that needs remediation.

Logs

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

OpenTelemetry Features

- **Auto-instrumentation**
- Get started in minutes with zero-code instrumentation for popular frameworks and libraries. Automatic instrumentation agents capture traces, metrics, and logs without modifying your source code.
- **Collector pipeline**
- Process, filter, and route telemetry data with the **OpenTelemetry Collector**. Deploy as an agent or gateway to receive, process, and export telemetry at scale with 200+ components.
- **Context propagation**
- Automatically correlate traces across service boundaries. Distributed context flows through your entire request path, connecting logs, metrics, and traces into a unified view.
- **Multi-language support**
- Native SDKs for 12+ languages including Java, Kotlin, Python, Go, JavaScript, .NET, Ruby, PHP, Rust, C++, Swift, and Erlang. Use your preferred language with first-class OpenTelemetry support
- **Stable and production-ready**
- Tracing and metrics APIs are stable across all major languages. Thousands of organizations run OpenTelemetry in production. Backed by the CNCF and major cloud providers
- **Open specifications**

- **Built on open, vendor-neutral specifications** for APIs, SDKs, and the wire protocol (OTLP). Transparent governance under the CNCF ensures long-term stability and community-driven evolution

Key Components of OpenTelemetry

- **APIs and SDKs**
- These are tools that developers use to add telemetry data collection to their applications. OpenTelemetry provides these tools for many programming languages like Java, Python, Go, and JavaScript.
- **Instrumentation Libraries**
- These libraries automatically collect telemetry data from popular libraries and frameworks. For example, they can automatically gather data from web frameworks like Express (Node.js), Spring (Java), and Django (Python).
- **Collector**
- The **OpenTelemetry Collector** is a service that receives, processes, and exports telemetry data. It can be run as an agent (alongside your application) or as a gateway (a central service). The Collector helps manage and transform the data before sending it to a backend system for analysis.
- **Exporters**
- Exporters are components that send the collected telemetry data to various backend systems like Prometheus, Jaeger, Zipkin, and others. These systems store and help analyze the data.

OpenTelemetry Architecture

- **Instrumentation**
- This is the process of adding code to your application to collect telemetry data. You can do this manually using OpenTelemetry APIs or automatically using instrumentation libraries. This code creates spans (for traces) and metrics, and adds contextual information.
- **SDKs and Resource Detection**

- SDKs handle the collection of telemetry data. They also detect and attach resource information like the application name, version, and environment (e.g., production, staging) to the data.
- **Context Propagation**
- This ensures that trace context (trace IDs and span IDs) is passed across different services so that you can see the full journey of a request through your system.
- **Processing with the Collector**
- The Collector processes telemetry data, applying tasks like batching, filtering, and sampling before sending it to backends. It can run as an agent or a gateway.
- **Exporting Data**
- Exporters send the processed telemetry data to various backend systems where it can be stored and analyzed.

Benefits of OpenTelemetry

- **Implement consistent data collection**. OpenTelemetry enables teams to collect telemetry data consistently across different apps, systems and use cases.
- **Avoid vendor lock-in.** OTel is a vendor-neutral solution that enables developers to switch out backend observability tools and providers without making significant changes to instrumentation code.
- **Streamline observability.** OTel simplifies observability by collecting telemetry data without requiring IT personnel to modify source code or metadata. It also enables developers to maintain full-stack visibility, regardless of which observability backends or vendors they use.
- **Future-proof IT ecosystems**. Because OpenTelemetry is backed by an open source community and the CNCF, it's made to evolve as computing technologies and observability needs change. For instance, OTel recently added continuous profiling to its group of primary telemetry signals.1
- **Track resource usage**. OpenTelemetry can capture data requests between servers to categorize resource usage by groups, helping IT teams track utilization between shared systems.
- **Prioritize data requests**. OpenTelemetry can create a priority ranking pipeline for data requests within the architecture so that competing requests traverse the network in order

of mission criticality.

- **Expand access to monitoring data**. With OTel, developers can monitor telemetry data and receive alerts on any web browser or device, making it easier to track software performance and overall system health in real time.

Key Challenges of OpenTelemetry

- **Uneven stability**
- Traces and metrics are stable across major languages (Java, Python, Go, .NET, etc.).
- Logs are less mature — some languages have stable log bridges, others are still evolving.
- Teams must check the status page before committing to a language/signal.
- **Instrumentation coverage gaps**
- Auto-instrumentation works well for popular frameworks (HTTP, gRPC, databases).
- Niche frameworks or internal libraries often require manual instrumentation.
- Heterogeneous stacks should audit coverage early to avoid surprises.
- **Collector complexity**
- The OTel Collector is powerful but operationally demanding.
- Tail-based sampling, multi-pipeline setups, and high-throughput ingestion add complexity.
- Requires monitoring, tuning, and scaling strategies to run reliably in production.
- **Cost considerations**
- OTel is free, but observability costs don’t vanish.
- Expenses shift to infrastructure (Collector clusters), backend storage/ingestion, and engineering time.
- Moving from proprietary platforms may reduce vendor lock-in, but total cost depends on scale and backend choice.

OpenTelemetry Collector architecture.

Key Components

- **Data Sources**
- Systems like OpenTelemetry SDKs, Kubernetes, and PostgreSQL generate telemetry data (metrics, logs, traces).
- **Receivers**
- These are entry points into the collector. Examples: OpenTelemetry receiver, Prometheus receiver, PostgreSQL receiver.
- **Processors**
- Middle layer that manipulates data:
- Batch → groups data for efficiency
- Filter → removes unwanted signals
- Transform → modifies or enriches data
- **Exporters**
- Send processed data to external systems like Elasticsearch, Google Cloud, or another OpenTelemetry pipeline.
- **Back-ends**
- Final destinations for analysis and visualization: Datadog, Elasticsearch, Google Cloud.
- **Extensions**
- Extra functionality such as health checks, profiling (pprof), debugging (zpages), and authentication.