Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Observability with OpenTelemetry

Class Duration

14 hours of live training delivered over 2 days.

Student Prerequisites

- Software development experience in Python, C#/.NET, or Node.js
- Basic understanding of HTTP services and APIs
- Familiarity with containers is helpful but not required
- No prior observability or OpenTelemetry experience required

Target Audience

This course is designed for software engineers, SREs, and platform engineers who need to make distributed systems observable. It suits teams instrumenting services for the first time, teams migrating from vendor-specific agents to vendor-neutral OpenTelemetry, and engineers responsible for choosing and operating observability backends.

Description

Observability with OpenTelemetry teaches participants to instrument, collect, and analyze telemetry from distributed systems using the industry-standard OpenTelemetry project. The course begins with observability foundations — the three pillars and how OpenTelemetry models traces, metrics, and logs as stable, correlated signals, with profiling emerging as a fourth signal — then moves into hands-on instrumentation. Participants compare zero-code auto-instrumentation with manual instrumentation in Python, C#/.NET, and Node.js, and learn how context propagation ties a single request together across service boundaries.

The course then covers the operational core of an OpenTelemetry deployment: the Collector and its receiver/processor/exporter pipelines, head-based and tail-based sampling strategies, and the semantic conventions that make telemetry queryable across teams and tools. Participants evaluate open source backends (Grafana, Prometheus, Jaeger, Tempo, Loki) alongside commercial platforms, then put telemetry to work with SLOs, alerting, and dashboards — while keeping ingest and storage costs under control. The course closes with observability for LLM applications

using the emerging GenAI semantic conventions, a bridge to our dedicated LLM Observability and Cost Engineering course.

Learning Outcomes

- Explain the three pillars of observability and how OpenTelemetry models them as correlated signals
- Describe the OpenTelemetry architecture: API, SDK, OTLP, Collector, and semantic conventions
- Apply zero-code auto-instrumentation to Python, .NET, and Node.js services
- Add manual spans, attributes, events, and custom metrics where auto-instrumentation falls short
- Propagate trace context across HTTP calls, message queues, and async boundaries with W3C Trace Context
- Deploy and configure the OpenTelemetry Collector with receivers, processors, and exporters
- Choose and implement appropriate sampling strategies, including tail-based sampling in the Collector
- Apply semantic conventions so telemetry is consistent and queryable across services
- Evaluate open source and commercial backends and export telemetry to Grafana, Prometheus, Jaeger, and Tempo
- Define SLIs and SLOs and build alerting that pages on symptoms, not noise
- Build dashboards that support real debugging workflows
- Control observability costs through sampling, filtering, aggregation, and retention policies
- Instrument LLM application calls using the OpenTelemetry GenAI semantic conventions

Training Materials

Comprehensive courseware is distributed online at the start of class. All students receive a downloadable MP4 recording of the training.

Software Requirements

Students will need a free, personal GitHub account to access the courseware, Docker Desktop or Podman to run the Collector and backends locally, a code editor (VS Code recommended), and a development environment for at least

one of the lab languages: Python 3.12+, .NET 8+, or Node.js 22+. If students are unable to configure a local environment, a cloud-based environment can be provided.

Training Topics

Observability Foundations

- Monitoring vs. observability
- The three pillars: traces, metrics, and logs
- Why correlation between signals matters
- Cardinality, dimensions, and the cost of telemetry
- The case for vendor-neutral instrumentation

The OpenTelemetry Project

- Project scope: API, SDK, OTLP protocol, Collector, conventions
- Signal maturity: stable traces, metrics, and logs
- Profiling as the emerging fourth signal
- The log bridge: integrating existing logging frameworks
- Instrumentation libraries and the registry

Instrumenting Python Services

- Zero-code instrumentation with opentelemetry-instrument
- SDK setup: tracer and meter providers, resources
- Manual spans, attributes, and span events
- Custom metrics: counters, histograms, gauges
- Instrumenting FastAPI, requests, and database clients

Instrumenting C#/.NET Services

- Built-in support: ActivitySource and System.Diagnostics.Metrics
- The OpenTelemetry .NET SDK and exporters
- ASP.NET Core and HttpClient instrumentation
- Enriching telemetry with middleware and processors

Instrumenting Node.js Services

- The Node.js SDK and auto-instrumentations package
- Instrumenting Express, Fastify, and HTTP clients
- Manual spans in async code
- Avoiding common pitfalls with bundlers and ESM

Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Context Propagation

- Trace context: trace IDs, span IDs, and parenting
- W3C Trace Context and baggage
- Propagation across HTTP, gRPC, and message queues
- Async boundaries and background jobs
- Debugging broken traces

The OpenTelemetry Collector

- Agent vs. gateway deployment patterns
- Receivers, processors, and exporters
- Batching, memory limiting, and resource detection
- Filtering, transforming, and redacting telemetry
- Collector distributions and the builder
- Running the Collector in Kubernetes

Sampling Strategies

- Why sample: signal vs. cost
- Head-based sampling: always-on, ratio, parent-based
- Tail-based sampling in the Collector
- Keeping errors and slow requests
- Sampling metrics and logs: aggregation and filtering

Semantic Conventions

- Why naming consistency matters
- Resource attributes: service, deployment, host
- HTTP, database, and messaging conventions
- Stability levels and migrating conventions
- Defining team-level attribute standards

Backends and Visualization

- Storage by signal: Prometheus/Mimir, Tempo, Jaeger, Loki
- Grafana as the unified query and dashboard layer
- Commercial platforms: evaluation criteria and trade-offs
- OTLP-native ingestion and avoiding lock-in
- Correlating traces, metrics, and logs in one UI

SLOs, Alerting, and Dashboards

- SLIs, SLOs, and error budgets

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

- Alerting on symptoms, not causes
- Burn-rate alerts from SLOs
- Dashboards for debugging vs. dashboards for reporting
- Linking alerts to traces for fast diagnosis

Cost Control

- Where observability spend goes: ingest, storage, query
- Sampling, filtering, and aggregation as cost levers
- Retention tiers and downsampling
- Measuring the value of telemetry you keep

Observability for LLM Applications

- Why LLM calls need tracing: latency, tokens, cost, quality
- The OpenTelemetry GenAI semantic conventions (in development)
- Tracing model calls, tool use, and agent steps
- Capturing prompts and completions safely
- Going deeper: our LLM Observability and Cost Engineering course