AIRFLOW SUMMIT

Beyond Logs: Unlocking Airflow 3.0 Observability with OpenTelemetry Traces

Christos Bisias, G-Research

Christos Bisias

Introduction

- I’m a software engineer from Greece
- I’m part of the G-Research Open Source team

GR-OSS

- G-Research is a leading quantitative research and technology firm
- G-Research is heavily investing in and using open source software
- The GR-OSS team is trying to leverage OSS to solve business problems

My background

- I have been contributing to Open Source projects on behalf of G-Research for the last 3.5 years
- In the past 1 year I've been doing a lot of work around OpenTelemetry and Airflow
- I adjusted Airflow's OTel implementation to make it support context propagation (more on that later)

Agenda

- What OTel is? Why use it with Airflow?
- OpenTelemetry basics
  - Traces and spans
  - Span data and context
  - Context propagation and mechanisms
- Demo
  - Execute an Airflow dag that exports sub spans
  - One of the tasks, makes a call to a running spring boot app that will also export a span

Why is my Dag slow?

What is OpenTelemetry?

- OpenTelemetry is a collection of OS tools that are used for collecting traces, logs and metrics from applications
  - The collected data is later exported to visualization backends
  - This presentation will focus on **traces**

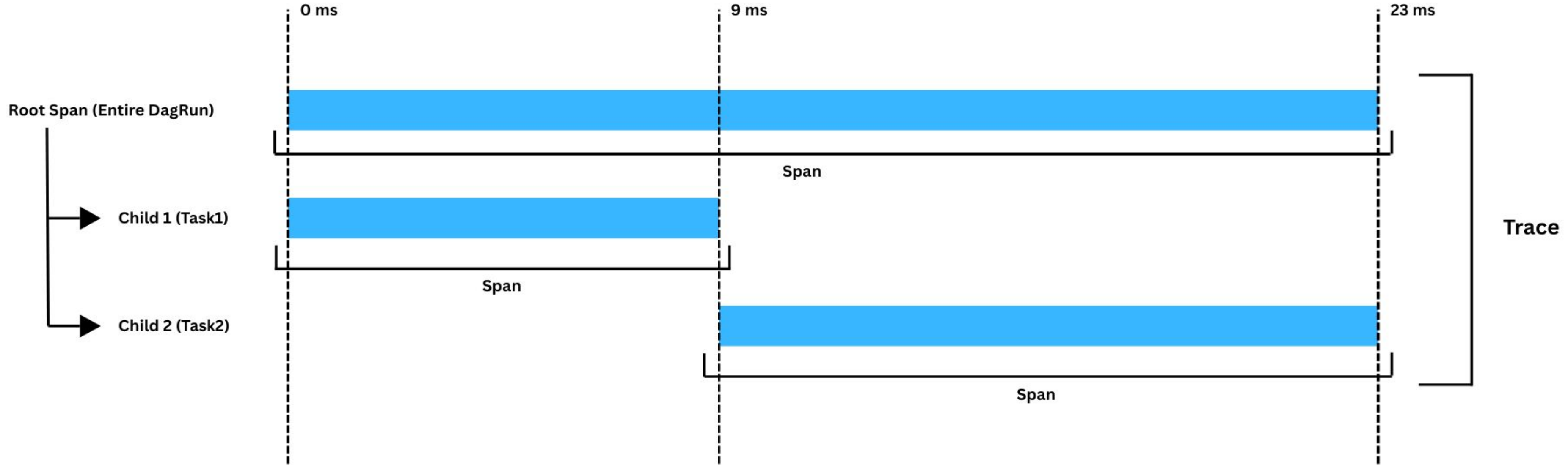
Why OpenTelemetry Traces?

- OTel is **technology agnostic**
- It has a different client implementation for each programming language
    - Python, Java, Golang, etc.
- The OTel SDK can be added as a dependency to any framework or application
- Especially useful in distributed systems
    - We can track an operation spreading across multiple services

Traces & Spans

- Let’s assume that we have a large operation which consists of multiple small steps like API calls, DB calls, calculations etc.
- The collection of **all** the steps, will be a **trace**
  - An entire DagRun
- Each **individual** step that we are observing, will be a **span**
  - A single task

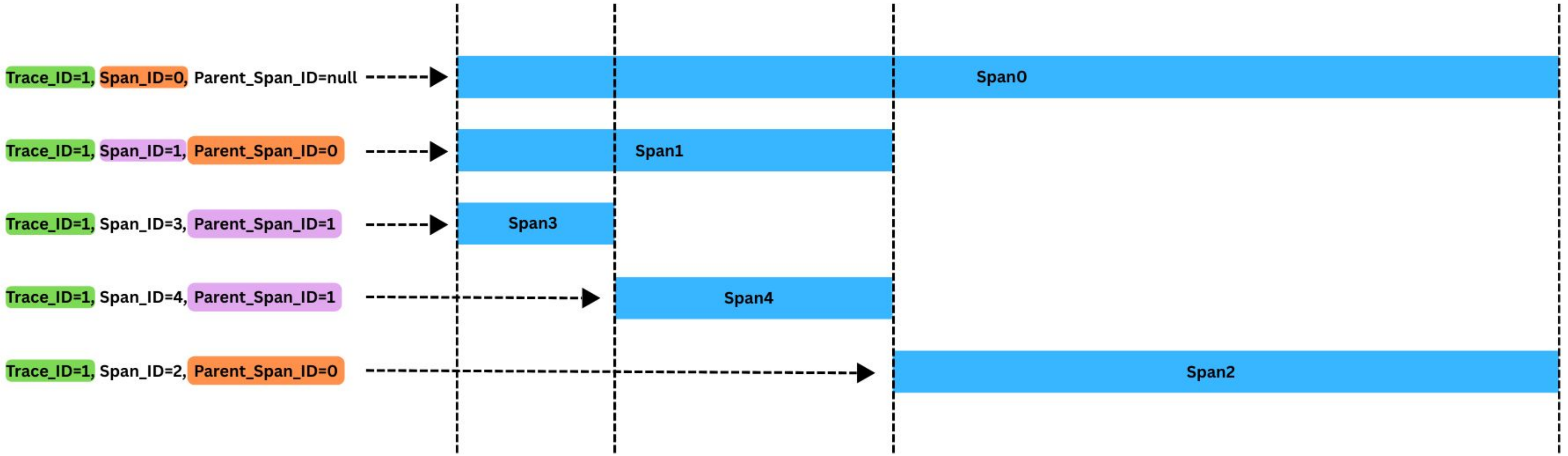
Trace & Spans Visualization



Why use OTel with Airflow?

- We can create our own sub spans under tasks to observe individual operations
- We can also monitor external calls to public APIs or other services running in our network

Span data & association

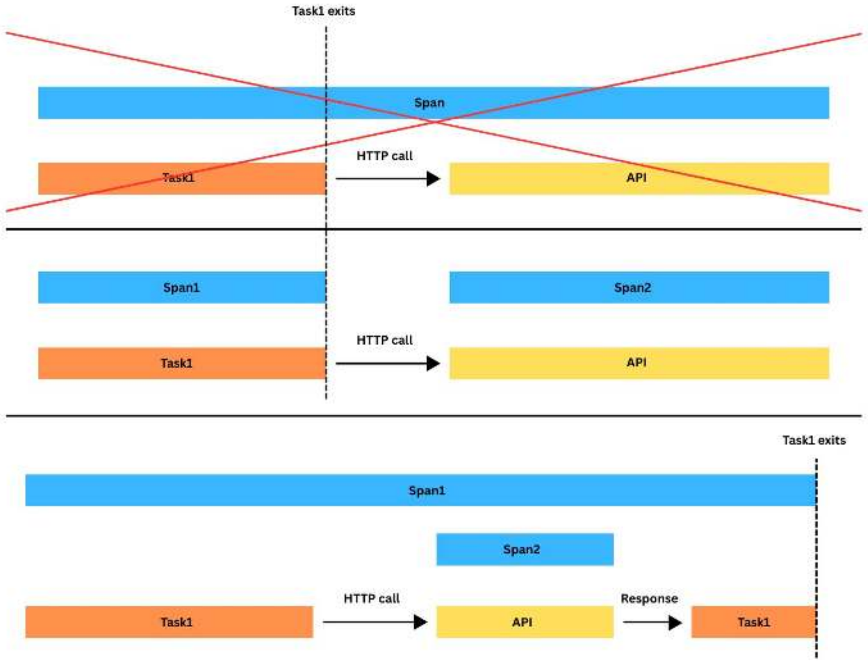


Trace_ID=1, Span_ID=0, Parent_Span_ID=null
Trace_ID=1, Span_ID=1, Parent_Span_ID=0
Trace_ID=1, Span_ID=3, Parent_Span_ID=1
Trace_ID=1, Span_ID=4, Parent_Span_ID=1
Trace_ID=1, Span_ID=2, Parent_Span_ID=0
Span0
Span1
Span3
Span4
Span2

OTel Specifications

- Spans are thread local and cannot be shared outside of the current thread
- This is done to ensure that each process is solely responsible for handling its own spans

Cross-service trace

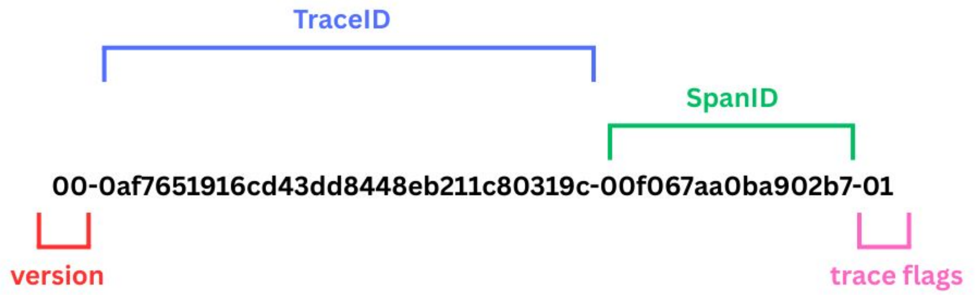


Context Propagation

- Context propagation is the method of sharing a span's context across services in a distributed system
- The context is used to create children spans
- The format follows the W3C specification

W3C Context Example

traceparent: 00-0af7651916cd43dd8448eb211c80319c-00f067aa0ba902b7-01



Propagation Mechanisms

- Common mechanisms of context propagation
  - HTTP calls
    - In the HTTP headers
  - gRPC calls
    - In the metadata
  - Message queues
    - Within the message
- Custom mechanism depending on the system
  - In Airflow we are propagating the context by storing it in the DB and later retrieving it

Capturing & Exporting Spans

- **Auto**-instrumentation
  - the instrumentation library automatically injects telemetry collection code into the application at runtime without requiring manual code changes
  - data are collected for certain frameworks, databases, HTTP clients and other common components
  - this is **strict**, it can't monitor anything that the library doesn't recognize
- **Manual** code injection into the app
  - the user has to write the code
  - it provides **flexibility** to monitor almost anything

<u>Demo</u>

Demo - Setup Explained

- Airflow with OTel traces enabled in the config
- An otel-collector service
- Jaeger as a visualization backend
- A Spring boot application running
    - With OTel SDK for Java
- Both Airflow and the spring boot app will create spans and export them to the common otel-collector
- The otel-collector will forward them to Jaeger

Demo - Dag Code

- The tasks will
  - Create sub spans using context propagation
  - Hook the auto-instrumentation library to monitor a GET request to a public API such as GitHub’s
  - Make a call to the spring boot app and pass the current context in the HTTP headers
    - The app will create a sub span with the context

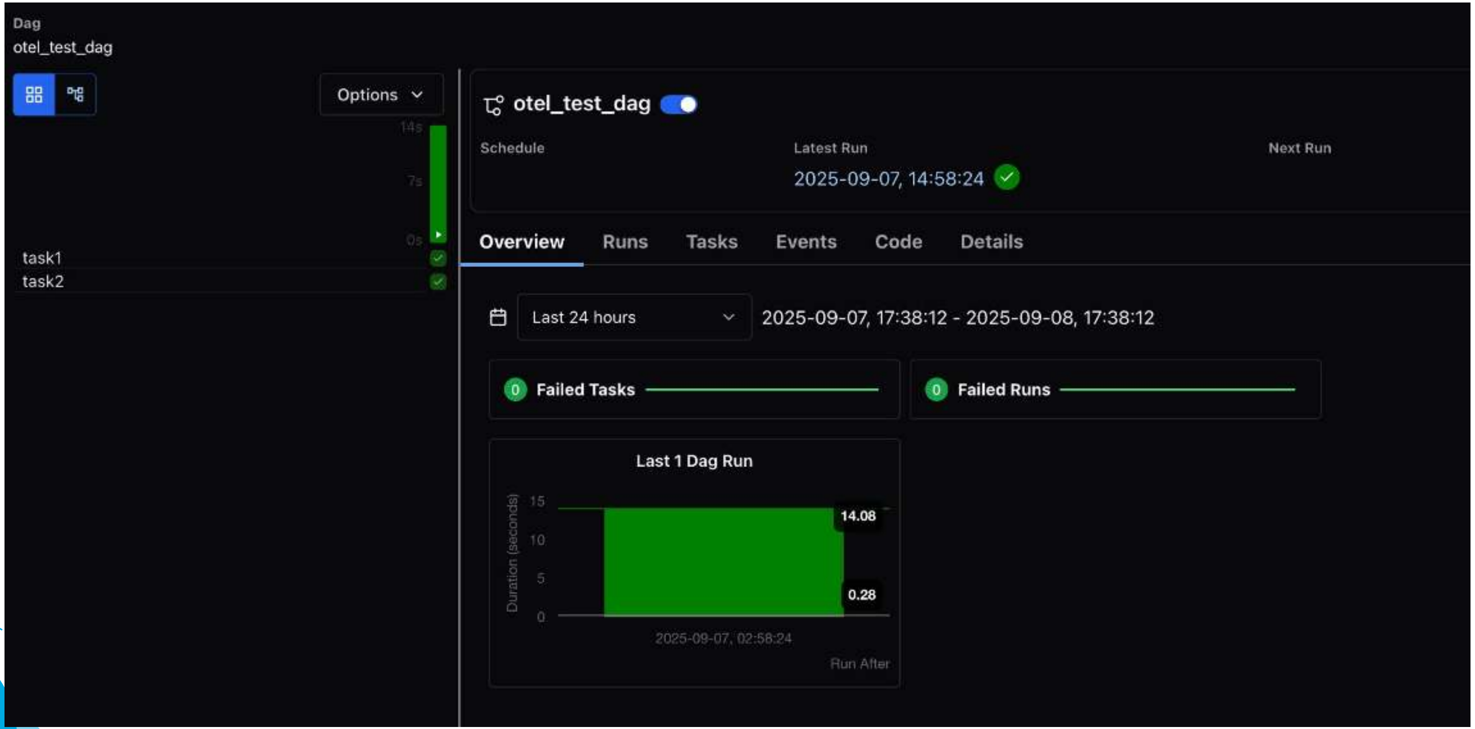
Demo - New Span

```
@task
def task1(ti):
     context_carrier = ti.context_carrier
     parent_context = otel_task_tracer.extract(context_carrier)

     with otel_task_tracer.start_child_span(
         span_name="part1_with_parent_ctx",
         parent_context=parent_context,
         component="dag",
     ) as p1_with_ctx_s:
         # Some work.
         logger.info("From part1_with_parent_ctx.")
```

Demo - Inject context into the headers

```
@task
def task2(ti):
    context_carrier = ti.context_carrier

    res = requests.get(
        "http://java-tester:7777/api/work",
        headers=context_carrier,
        timeout=25
    )
```

Webserver UI



Dag
otel_test_dag

otel_test_dag

Schedule

Latest Run
2025-09-07, 14:58:24

Next Run

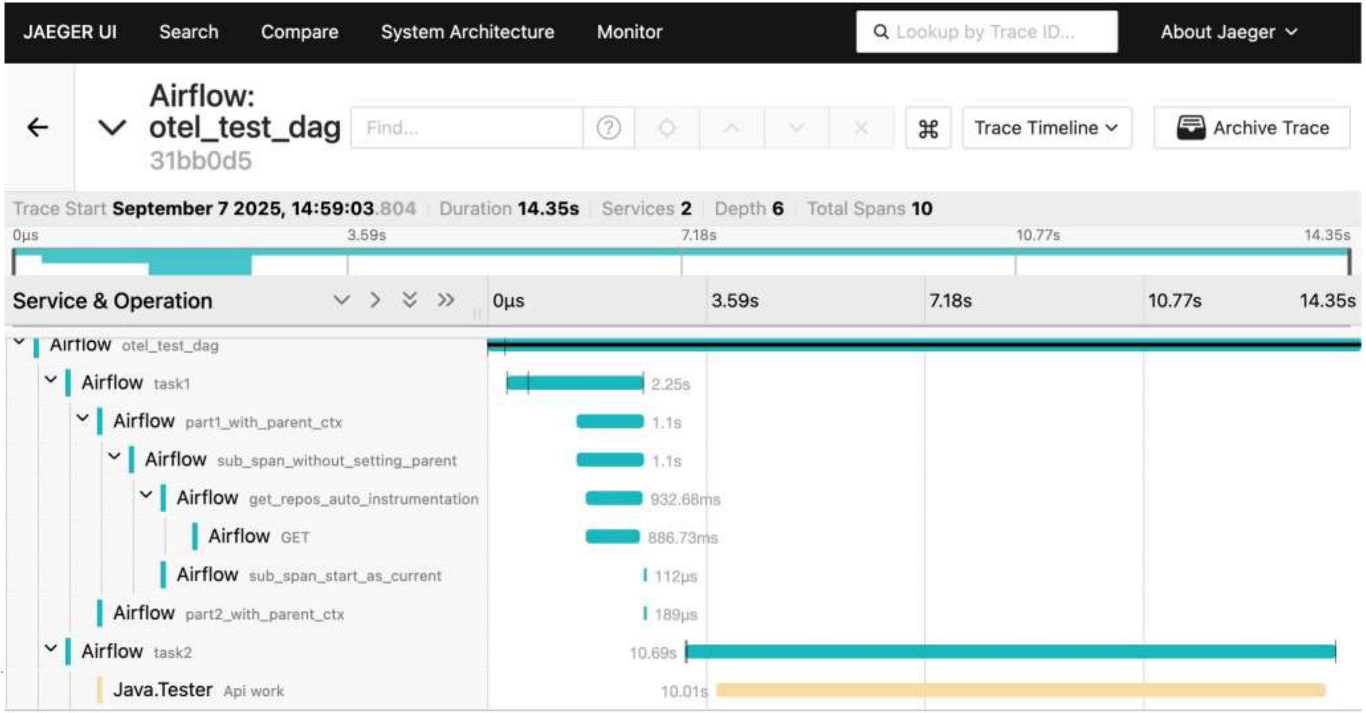
2025-09-07, 17:38:12 - 2025-09-08, 17:38:12

0 Failed Tasks

0 Failed Runs

Last 1 Dag Run

Jaeger UI



Airflow: otel_test_dag 31bb0d5

Trace Start **September 7 2025, 14:59:03**.804 Duration **14.35s** Services **2** Depth **6** Total Spans **10**

0μs 3.59s 7.18s 10.77s 14.35s

Service & Operation

Service & Operation	Duration
Airflow otel_test_dag	
Airflow task1	2.25s
Airflow part1_with_parent_ctx	1.1s
Airflow sub_span_without_setting_parent	1.1s
Airflow get_repos_auto_instrumentation	932.68ms
Airflow GET	886.73ms
Airflow sub_span_start_as_current	112μs
Airflow part2_with_parent_ctx	189μs
Airflow task2	10.69s
Java.Tester Api work	10.01s

Next Topics to Research

- OpenTelemetry SDK initialization
- When to use
  - Simple vs Batch SpanProcessor
- Span Attributes
- Propagators
  - Context `inject()` and `extract()`
- SDK static context variable
  - Starting and ending span manually
- App or Request auto-instrumentation
- OTel Metrics and visualization with Prometheus and Grafana

Thank you! Questions?

RESEARCH | OPEN SOURCE SOFTWARE

Connect with me

Demo Code