ROS-OPENTELEMETRY

End-to-End Telemetry for Robotics

ABOUT ME

- Lead Software & Robotics Engineer at Circu Li-ion
- Open-source Contributor

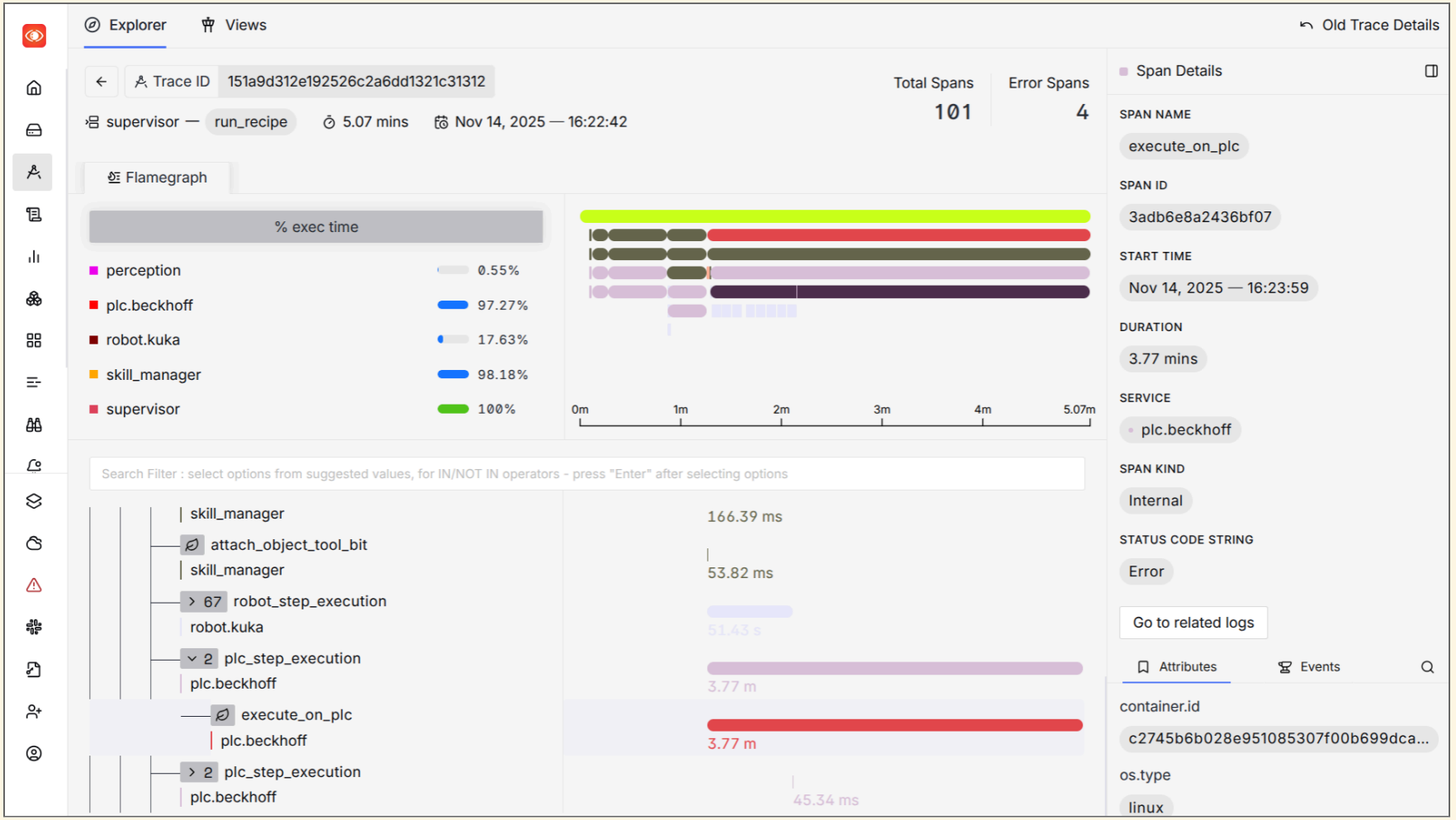
ROS IS A DISTRIBUTED SYSTEM

"*ROS was invented at the moment when micro-services architecture was a hype.*"
Guillaume Binet, Founder @ Copper Robotics

- ROS Node == MicroService
- DDS == Communication Layer
- How people handle it?

DISTRIBUTED TRACING

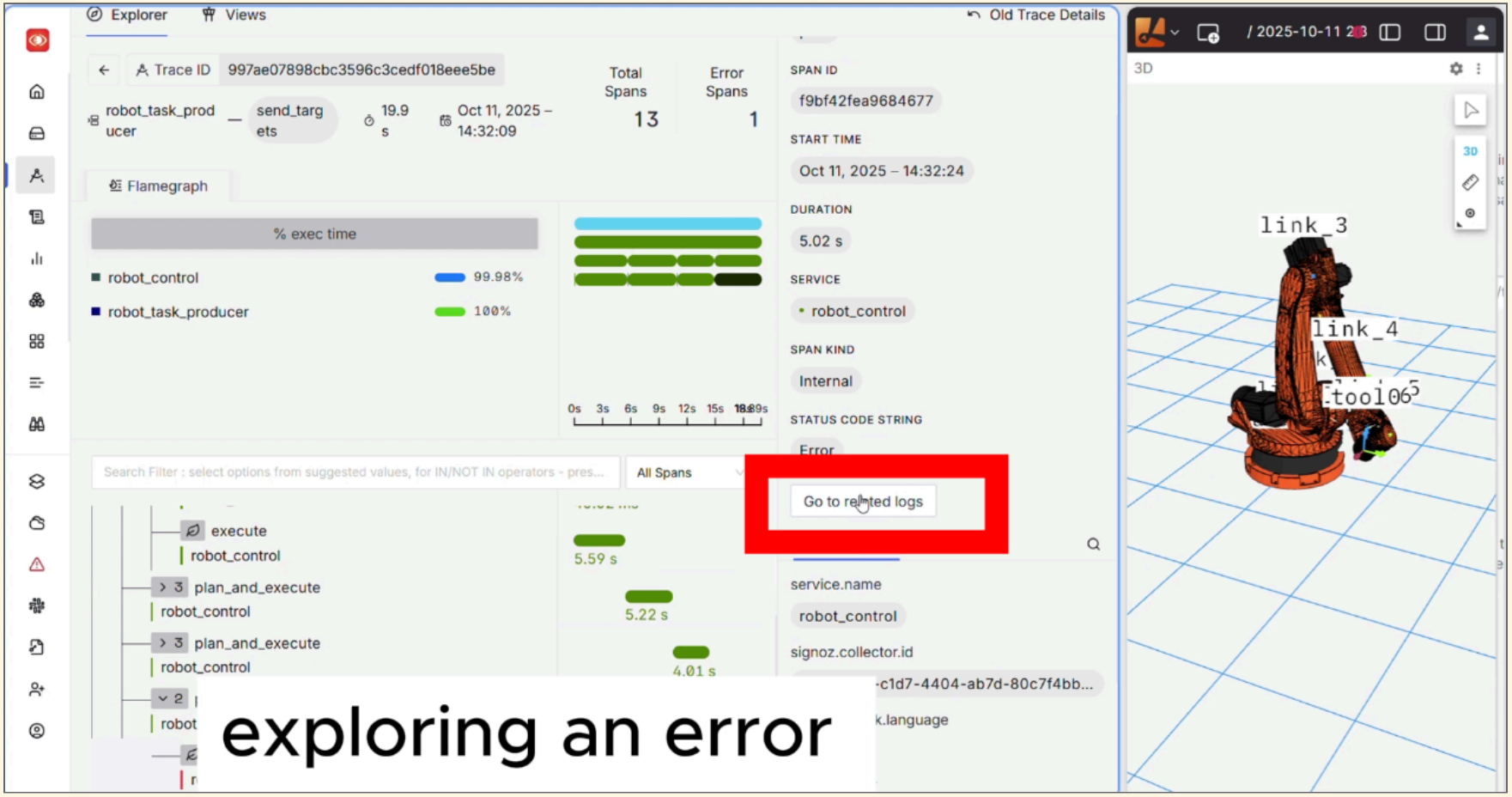
Traces give us the big picture of what happens when a request is made to an application.
Whether your application is a monolith with a single database or a sophisticated mesh of services, traces are essential to understanding the full “path” a request takes in your application.
- opentelemetry.io



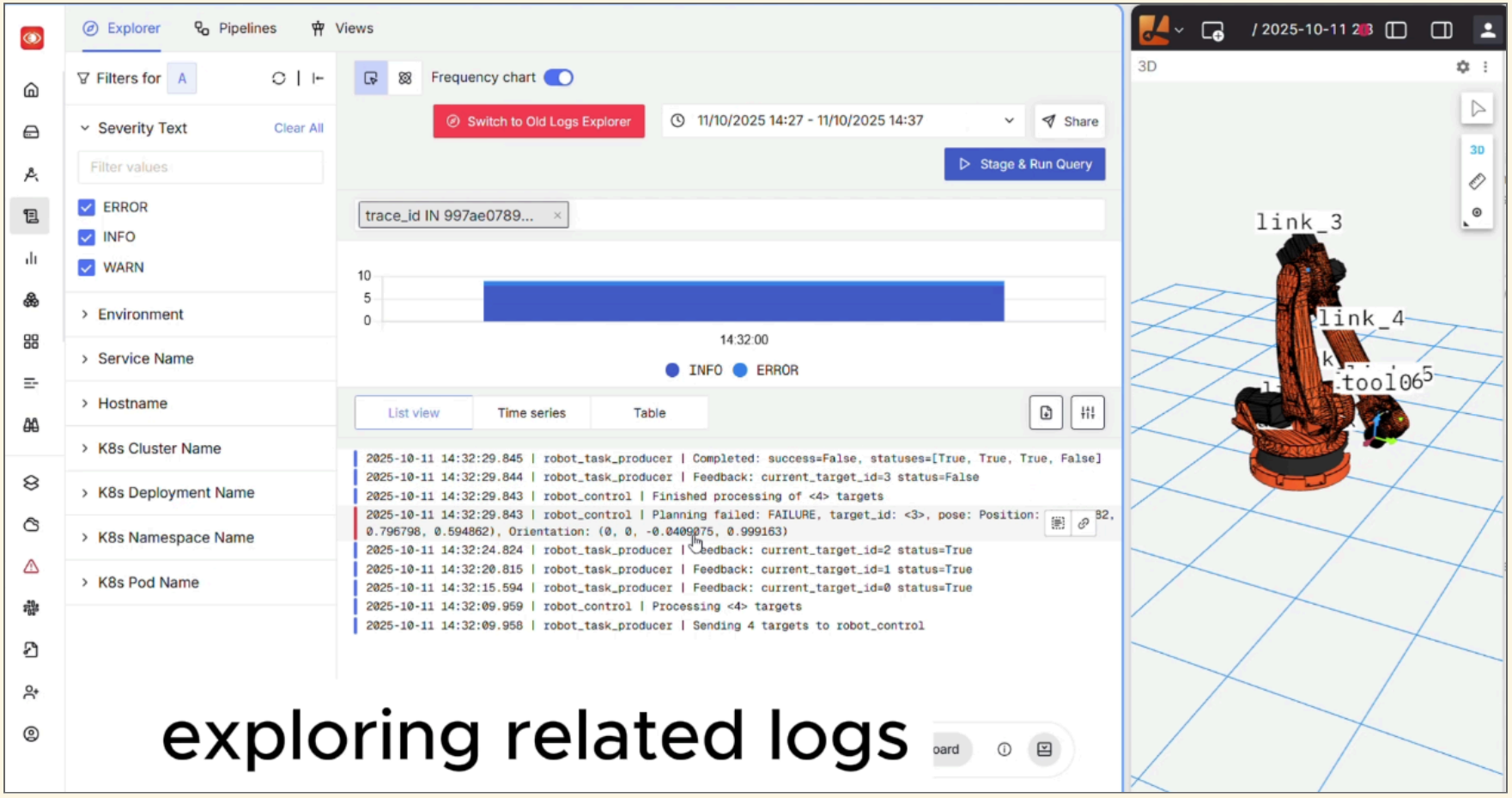
Diassembly request going though our system

CORRELATED LOGS

Correlating traces and logs allows you to investigate issues by navigating from a specific span in a trace directly to the logs that were generated during that operation.
This makes debugging faster and more intuitive by providing the exact context needed to understand an error or performance problem.
- datadoghq.com



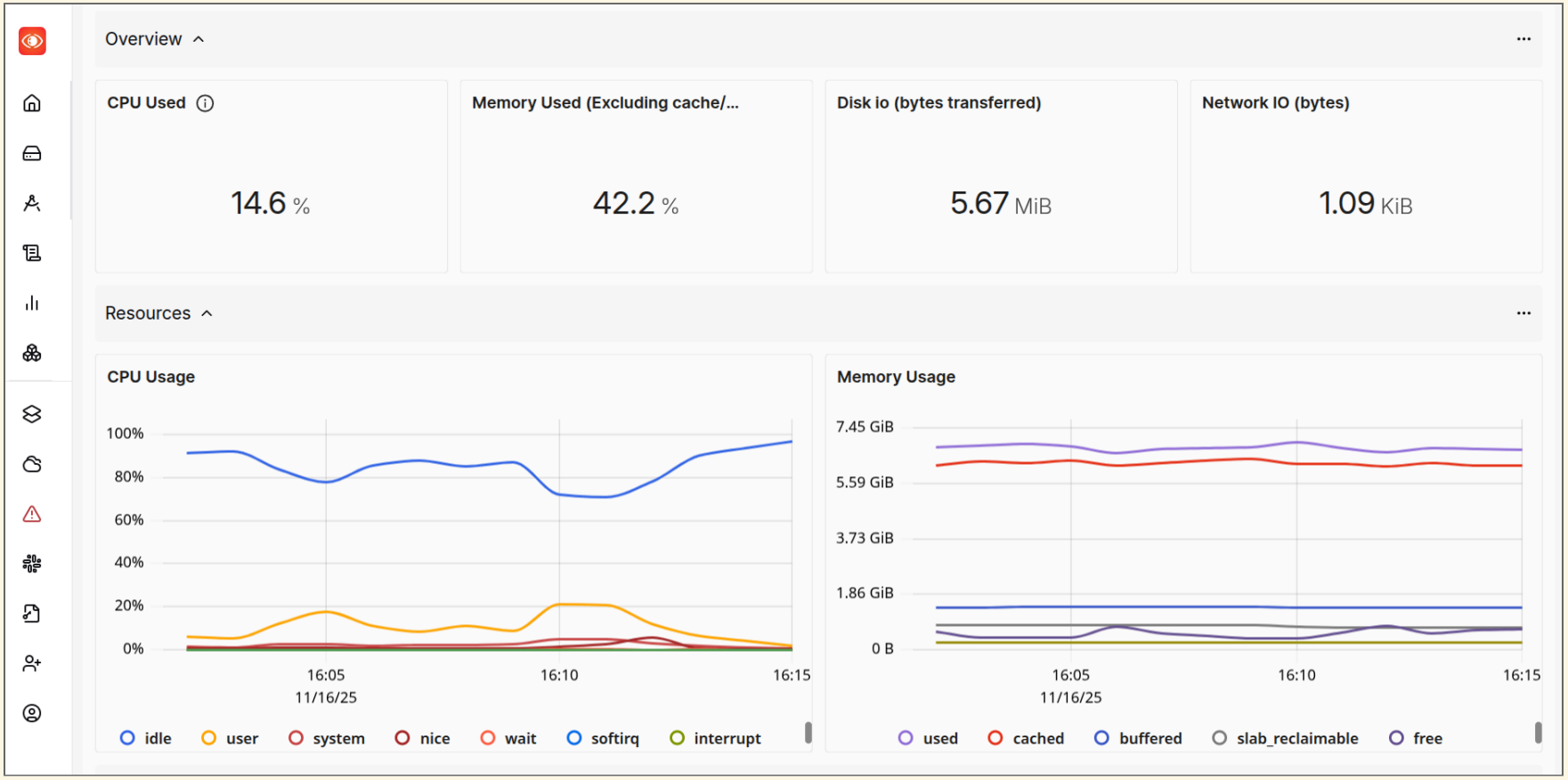
Selecting a trace marked with error



Exploring correlated logs

METRICS

"A measurement captured at runtime."
- *opentelemetry.io*



no more ssh/htop to remote machine

WHY OPENTELEMETRY?

- An observability framework and toolkit designed to facilitate the **Generation**, **Export**, **Collection** of **traces**, **metrics**, and **logs**.
- Open source, Vendor- and tool-agnostic
- Steered by Cloud Native Computing Foundation / Linux Foundation

vendors supporting opentelemetry protocol

Vendors

Vendors who natively support OpenTelemetry

A non-exhaustive list of organizations offering solutions that consume OpenTelemetry natively via OTLP, such as observability backends and observability pipelines.

Some organizations provide a distribution (of customized OpenTelemetry components), that provides additional capabilities or for improved ease of use.

Open Source (OSS) refers to a vendor who has an observability product that is open source. The vendor may still have other products that are closed source, such as a SaaS offering that

Organization	OSS	Commercial	Native OTLP	Learn more
Apache Doris	Yes	No	Yes	github.com/...
Apache SkyWalking	Yes	No	Yes	skywalking.apache.org/...
Fluent Bit	Yes	No	Yes	docs.fluentbit.io/...
Jaeger	Yes	No	Yes	www.jaegertracing.io/...
OpenLIT	Yes	No	No	docs.openlit.io/...
OpenSearch	Yes	No	Yes	github.com/...
Centreon	Yes	Yes	Yes	docs.centreon.com/...
ClickHouse	Yes	Yes	No	github.com/...
Coroot	Yes	Yes	Yes	github.com/...
Elastic	Yes	Yes	Yes	www.elastic.co/...
Embrace	Yes	Yes	Yes	embrace.io/...
Grafana Labs	Yes	Yes	Yes	grafana.com/...
GreptimeDB	Yes	Yes	Yes	docs.greptime.com/...
Highlight	Yes	Yes	Yes	highlight.io/...
HyperDX	Yes	Yes	Yes	www.hyperdx.io/...
Langfuse	Yes	Yes	Yes	langfuse.com/...
MetricsHub	Yes	Yes	Yes	www.metricshub.com/...
observIQ	Yes	Yes	Yes	docs.bindplane.observiq.com
Odigos	Yes	Yes	Yes	docs.odigos.io
OneUptime	Yes	Yes	Yes	oneuptime.com/...
OpenObserve	Yes	Yes	Yes	openobserve.ai/...
Parseable	Yes	Yes	Yes	www.parseable.com/...
qryn	Yes	Yes	Yes	qryn.metrico.in/...
Red Hat	Yes	Yes	Yes	docs.redhat.com/...
SigNoz	Yes	Yes	Yes	signoz.io
Tracetest	Yes	Yes	Yes	docs.tracetest.io/...
Uptrace	Yes	Yes	Yes	uptrace.dev/...
VictoriaMetrics	Yes	Yes	Yes	github.com/...
Alauda	No	Yes	Yes	www.alauda.io/...
Alibaba Cloud	No	Yes	Yes	www.alibabacloud.com/...
Apica	No	Yes	Yes	docs.apica.io/...
AppDynamics (Cisco)	No	Yes	Yes	docs.appdynamics.com/...
Aria by VMware (Wavefront)	No	Yes	Yes	docs.wavefront.com/...
Arize Phoenix	No	Yes	Yes	docs.arize.com/...
Aspecto	No	Yes	Yes	www.aspecto.io
AWS	No	Yes	Yes	aws-otel.github.io
Axiom	No	Yes	Yes	axiom.co/...
Azure	No	Yes	No	docs.microsoft.com/...
Better Stack	No	Yes	Yes	betterstack.com/...
BMC Helix	No	Yes	Yes	docs.bmc.com/...
Bonree	No	Yes	Yes	one.bonree.com/...
BugSnag	No	Yes	Yes	docs.bugsnag.com/...
Causely	No	Yes	Yes	www.causely.ai/...
Chronosphere	No	Yes	Yes	docs.chronosphere.io/...
Control Plane	No	Yes	Yes	docs.controlplane.com/...
ControlTheory	No	Yes	Yes	www.controltheory.com/...
Coralogix	No	Yes	Yes	coralogix.com/...
Cribl	No	Yes	Yes	docs.cribl.io/...
DaoCloud	No	Yes	Yes	docs.daocloud.io/...
Dash0	No	Yes	Yes	www.dash0.com/...
Datadog	No	Yes	Yes	docs.datadoghq.com/...

Digitate (Ignio)	No	Yes	Yes	docs.digitate.com/...
DX O2 by Broadcom	No	Yes	Yes	techdocs.broadcom.com/...
Dynatrace	No	Yes	Yes	www.dynatrace.com/...
Google Cloud Platform	No	Yes	Yes	github.com/...
groundcover	No	Yes	Yes	docs.groundcover.com/...
GuanChaYi by LogEase	No	Yes	Yes	logease.cn/...
Helios	No	Yes	Yes	gethelios.dev/...
Heroku	No	Yes	Yes	devcenter.heroku.com/...
Honeycomb	No	Yes	Yes	docs.honeycomb.io/...
Immersive Fusion	No	Yes	Yes	docs.immersivefusion.com/...
Instana	No	Yes	Yes	www.ibm.com/...
ITRS	No	Yes	Yes	docs.itrsgroup.com/...
KloudFuse	No	Yes	Yes	docs.kloudfuse.com/...
KloudMate	No	Yes	Yes	docs.kloudmate.com/...
Last9	No	Yes	Yes	last9.io/...
LogicMonitor	No	Yes	Yes	www.logicmonitor.com/...
LogScale by Crowdstrike (Humio)	No	Yes	Yes	library.humio.com/...
Logz.io	No	Yes	Yes	docs.logz.io/...

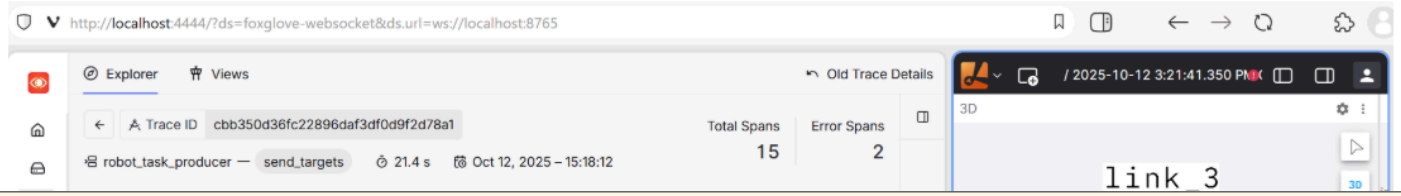
ROS2 X OPENTELEMETRY = ROS-OPENTELEMETRY

https://github.com/szobov/ros-opentelemetry

README License

ROS2 OpenTelemetry Integration Library

A production-grade integration library for instrumenting ROS2 (Robot Operating System 2) applications with OpenTelemetry distributed tracing and observability capabilities. This project provides a comprehensive toolchain for building, deploying, and monitoring ROS2 workspaces with native OpenTelemetry support for both C++ and Python nodes.



INSTRUMENTING YOUR CODE

MESSAGE INTERFACE

```
# Goal
RobotTarget[] targets

ros_opentelemetry_interfaces/TraceMetadata trace_metadata # <-- trace interface
\---
# Result
bool success
bool[] targets_status
\---
# Feedback
uint32 current_target_id
bool status
```

```
  <depend>ros_opentelemetry_interfaces</depend>
```

SETTING UP TRACER

```cpp
#include "ros_opentelemetry_cpp/ros_opentelemetry_cpp.hpp"

std::string otlp_grpc_endpoint = "hostname-of-your-otel-collector:4317";
ros_opentelemetry_cpp::setup_tracer("robot_control", otlp_grpc_endpoint);
```

```python
from ros_opentelemetry_py import setup_tracer

# somewhere at the start of your node
if __name__ == "__main__":
    # Expects environment variable OTLP_ENDPOINT set
    setup_tracer("robot_task_producer")
```

TRACING CODE

```
#include <opentelemetry/trace/span.h>
#include <opentelemetry/trace/tracer.h>
#include <opentelemetry/trace/provider.h>
#include <opentelemetry/trace/scope.h>


auto tracer =
opentelemetry::trace::Provider::GetTracerProvider()->GetTracer(
      "name_of_your_component");
auto span = tracer-
>StartSpan("handleActionOrServiceOrOtherCallback");
{

    auto target_span = tracer->StartSpan("nested_span");
    opentelemetry::trace::Scope scope(span);
    // your code
}
```

```python
from opentelemetry import trace

tracer: trace.Tracer = trace.get_tracer(__name__)

@tracer.start_as_current_span("method_of_your_node")
def method_of_your_node(self, params):
    ...
```

INJECTING TRACE CONTEXT

```
from ros_opentelemetry_py import inject_trace_context

example_msg = ExampleActionMessage.Goal()
example_msg.trace_metadata = inject_trace_context()
```

EXTRACTING TRACE CONTEXT

```
#include <opentelemetry/context/runtime_context.h>

const auto goal = goal_handle->get_goal();
auto extracted_ctx =
ros_opentelemetry_cpp::extract_trace_context(&goal-
>trace_metadata);
[[maybe_unused]] auto ctx_token =

opentelemetry::context::RuntimeContext::Attach(extracted_ctx);
```

CORRELATING LOGS

```
RCLCPP_ERROR_TRACED(this->get_logger(), "logger")
```

```
from ros_opentelemetry_py import wrap_logger
self._traced_logger = wrap_logger(self.get_logger())
```

COLLECTING & EXPORTING LOGS

```
receivers:
  filelog:
    include: ["/opt/logs/**/*.log"]
    start_at: end
    multiline:
      line_start_pattern: '^\[\w+\] \[\d+\.\d+\] \[.*\]:'  # To
support cases when we output multiline json
    operators:
      - type: regex_parser
        regex: '^\[(?P<level>\w+)\] \[(?P<timestamp>\d+\.\d+)\] \
[(?P<source>[^\]]+)\]: (?P<message>.*)$'
        timestamp:
          parse_from: attributes.timestamp
          layout_type: epoch
          layout: "s.ns"
        severity:
          parse_from: attributes.level
      - type: regex_parser
        parse_from: attributes.message
        regex: '^(?:\[trace_id=(?P<trace_id>[0-9a-f]
{32})\s+span_id=(?P<span_id>[0-9a-f]{16})\]\s*)?(?P<body>.*)$'
...
```

DEMO EXAMPLE

```
# To run example telemetry setup
just docker-up-telemetry

# Run example
just docker-up-example
```

THANKS FOR LISTENING

- linkedin/szobovdev
- github/szobov
- blog.szobov.ru

Speaker notes