

Introduction to OpenTelemetry with Flask

FlaskCon 2024



Jessica Garson

@JessicaGarson

Senior Developer Advocate



I had attempted to find a short primer on starting with OpenTelemetry with Flask but couldn't find much more than a [demo application](#).



ChatGPT

Sure! Imagine you have a big box of crayons (these are like different parts of a computer program) and you want to make a beautiful picture (which is like running the program to do something cool). But sometimes, while you're drawing, a crayon might break or not color the way it should. This can make your picture not turn out as nice as you wanted.

OpenTelemetry is like having a magical friend who watches you while you draw. This friend takes notes on which crayons you use, how they perform, and if they break. Then, if something goes wrong, your magical friend can tell you exactly which crayon broke and why, so you can fix it and make sure your picture turns out great next time!

So, OpenTelemetry helps people who make computer programs make sure their programs run smoothly and fix any problems that happen.



Observability fundamentals

Observability helps you determine what's happening inside your system by analyzing the data it outputs



What isn't operating as expected

Beyond detecting problems

What is OpenTelemetry (OTel)?

OpenTelemetry (OTel) is an observability framework that is open source and vendor neutral, designed to work with any backend system.

It provides standardized APIs, libraries, and tools to collect telemetry data such as metrics, logs, and traces.

OpenTelemetry enables comprehensive monitoring
across services and platforms.

There are many OpenTelemetry
components

- A specification for all components
- A standard protocol that defines the shape of telemetry data
- Semantic conventions that define a standard naming scheme for common telemetry data types
- APIs that define how to generate telemetry data
- Language SDKs that implement the specification, APIs, and export of telemetry data
- A library ecosystem that implements instrumentation for common libraries and frameworks
- Automatic instrumentation components that generate telemetry data without requiring code changes
- The OpenTelemetry Collector, a proxy that receives, processes, and exports telemetry data
- Various other tools, such as the OpenTelemetry Operator for Kubernetes, OpenTelemetry Helm Charts, and community assets for FaaS

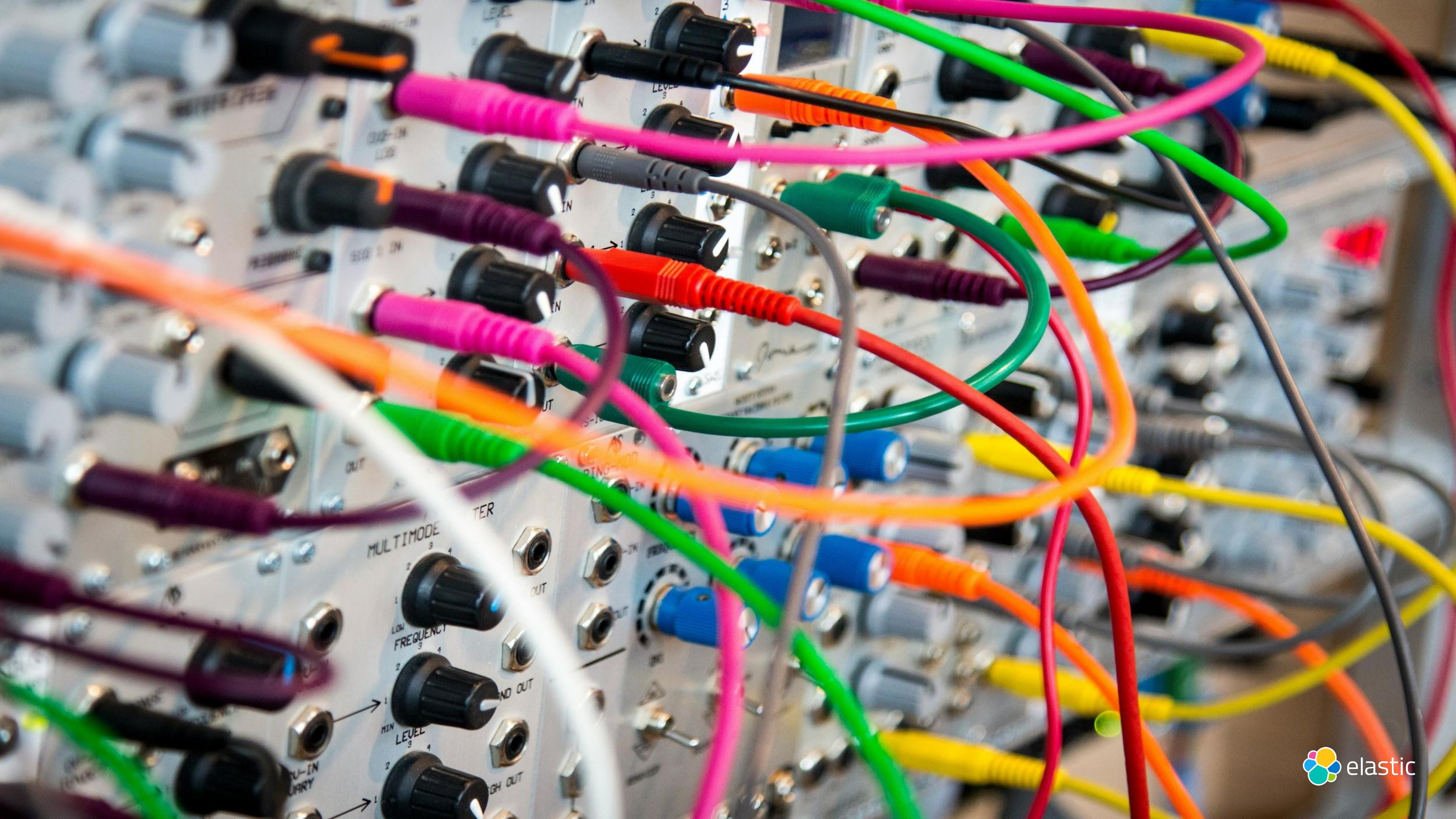
Why?

A photograph of a stack of logs in a forest. The logs are cut trees, showing their circular cross-sections. They are piled up in a somewhat haphazard manner, with some logs leaning against each other. The background is filled with dense forest trees, and the ground is covered with fallen branches and needles. The lighting suggests it might be late afternoon or early morning.

Logs alone are simply not
enough sometimes

Um, what?

>	*	2024-05-10	08:00:06.117	55	54440319	...	http://api.nasa.gov/neo/rest/v1/neo/sentry/544...
>	*	2024-05-10	08:00:06.117	56	54440642	...	NaN
▼	*	2024-05-10	08:00:06.117	57	54440644	...	http://api.nasa.gov/neo/rest/v1/neo/sentry/544...
<div>Copy Similar entries Expand nested fields Hide log summary</div>							
▼	{						
	insertId: "663e0c460001c9a60e5e65c3"						
▶	labels: {2}						
	logName: "projects/ncav-293119/logs/run.googleapis.com%2Fstdout"						
	receiveTimestamp: "2024-05-10T12:00:06.338017672Z"						
▶	resource: {2}						
	textPayload: "57 54440644 ... http://api.nasa.gov/neo/rest/v1/neo/sentry/544..."						
	timestamp: "2024-05-10T12:00:06.117158Z"						
	}						
>	*	2024-05-10	08:00:06.117	[43 rows x 18 columns]			



Future proof



Benefits

- Enhanced visibility into the execution flow and call timing
- Metrics on how users utilize services to pinpoint unnecessary ones
- The ability to detect patterns, such as peak usage times, which aids in production planning and scaling.

Logs

Logs are records of events in a system, documenting operations, errors, and activities to aid in troubleshooting, monitoring, and compliance.

Metrics

Metrics are quantitative measurements that track the performance and health of a system

Traces

Traces track the path and interactions of a request through a system.

Spans within traces? What?!?!



Spans are the building blocks of traces.
They include the following information:

- Name
- Parent span ID (empty for root spans)

Ohhhh! A span within a trace!

Span #1

```
Trace ID      : 4e6b8d4dcd44d9245285a9459a0114e6
Parent ID     : 6c2171f87b96401f
ID            : d56a8dc2bb6f2dbb
Name          : jinja2.render
Kind          : Internal
Start time    : 2024-05-14 19:07:24.897193 +0000 UTC
End time      : 2024-05-14 19:07:24.897258 +0000 UTC
Status code   : Unset
Status message :
```

Attributes:

```
-> jinja2.template_name: Str(<memory>)
    {"kind": "exporter", "data_type": "traces", "name": "debug"}
```

Root span

Span #1

Trace ID	:	4e6b8d4dcd44d9245285a9459a0114e6
Parent ID	:	
ID	:	6c2171f87b96401f
Name	:	/
Kind	:	Server
Start time	:	2024-05-14 19:07:24.894406 +0000 UTC
End time	:	2024-05-14 19:07:24.897374 +0000 UTC
Status code	:	Unset
Status message	:	

Demo in Flask

Let's start with the classic Flask example

To-Do List

ADD TASK

Cool! A new task!

Delete

A very simple application



Let's instrument our app

Instrumenting refers to the process of adding observability features to your application to collect telemetry data, such as traces, metrics, and logs.



```
pip install opentelemetry-distro  
opentelemetry-bootstrap -a install
```



```
export OTEL_PYTHON_LOGGING_AUTO_INSTRUMENTATION_ENABLED=true
opentelemetry-instrument \
    --traces_exporter console \
    --metrics_exporter console \
    --logs_exporter console \
    --service_name todo \
flask run -p 8080
```

You didn't add any code to
your file. What?!?!

So when would you want to use manual instrumentation?

Manually instrumenting our application

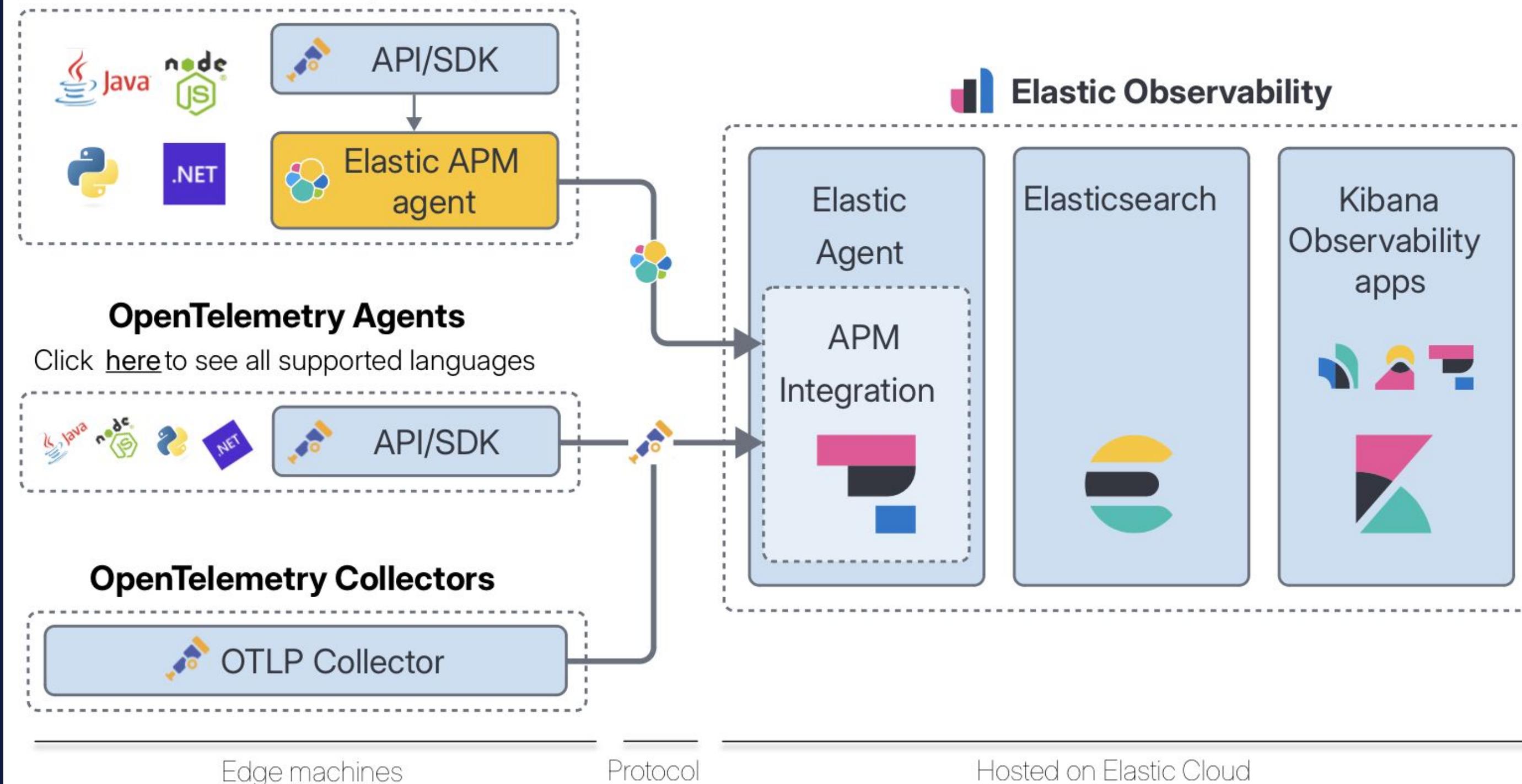
How does Elastic fit into this?

What is Elastic?

The Elastic Stack natively supports the OpenTelemetry protocol (OTLP). This means trace data and metrics collected from your applications and infrastructure can be sent directly to the Elastic Stack.

OpenTelemetry API/SDK with Elastic APM agents

Available in Java, .NET, Node.js, and Python



Sending to a collector on Docker

A collector is a component that collects, processes, and exports telemetry data like metrics, logs, and traces, facilitating observability across different systems and platforms.

**You can configure your collector via a
YAML file**

**This is also one method we can
use to connect to Elastic**



```
docker run -p 4317:4317 \
-v $(pwd)/otelcollector-config.yaml:/etc/otel-collector-config.yaml \
otel/opentelemetry-collector:latest \
--config=/etc/otel-collector-config.yaml
```





```
pip install opentelemetry-exporter-otlp
```



```
export OTEL_PYTHON_LOGGING_AUTO_INSTRUMENTATION_ENABLED=true  
opentelemetry-instrument --logs_exporter otlp flask run -p 8080
```

Agents vs APM Client

The Elastic APM OpenTelemetry bridge allows you to create Elastic APM Transactions and Spans, using the OpenTelemetry API.

Elastic APM agents translate OpenTelemetry API calls to Elastic APM API calls. This allows you to reuse your existing instrumentation to create Elastic APM transactions and spans.

Elastic supports both OpenTelemetry and custom agents for sending data to APM Server. You can choose based on your needs.

State of adoption

WORK in PROGRESS

<https://opentelemetry.io/status/>

Closing thoughts

It's very easy to start instrumenting your code.

It is highly configurable and extensible

It scales well from large applications to
small applications

Next steps



Resources

<https://github.com/JessicaGarson/Introduction-to-OpenTelemetry-with-Flask>

**Let me know if this talk inspires you to build
anything. I'm @JessicaGarson on most
platforms.**

Thank you!