Introduction to OpenTelemetry with Flask

FlaskCon 2024

elastic

Jessica Garson

@JessicaGarson

Senior Developer Advocate

elastic

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

elastic

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.


elastic

Agenda

1. Observability fundamentals
2. What is OpenTelemetry?
3. Why use it?
4. Demo
5. State of adoption
6. Next steps

elastic

Observability fundamentals

elastic

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

elastic

elastic

What isn’t operating as expected

elastic

Beyond detecting problems

elastic

What is an agent?

elastic

An observability agent is a software component or tool that collects, processes, and sends telemetry data from a system or application to an observability platform for analysis.

elastic

What is OpenTelemetry (OTel)?

elastic

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.

OpenTelemetary enables comprehensive monitoring across services and platforms.

elastic

There are many OpenTelemetry components

Main OpenTelemetry components

OpenTelemetry consists of the following major 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

OpenTelemetry is used by a wide variety of libraries, services and apps that have OpenTelemetry integrated to provide observability by default.

OpenTelemetry is supported by numerous vendors, many of whom provide commercial support for OpenTelemetry and contribute to the project directly.

Why?


Logs alone are simply not
enough sometimes

Um, what?

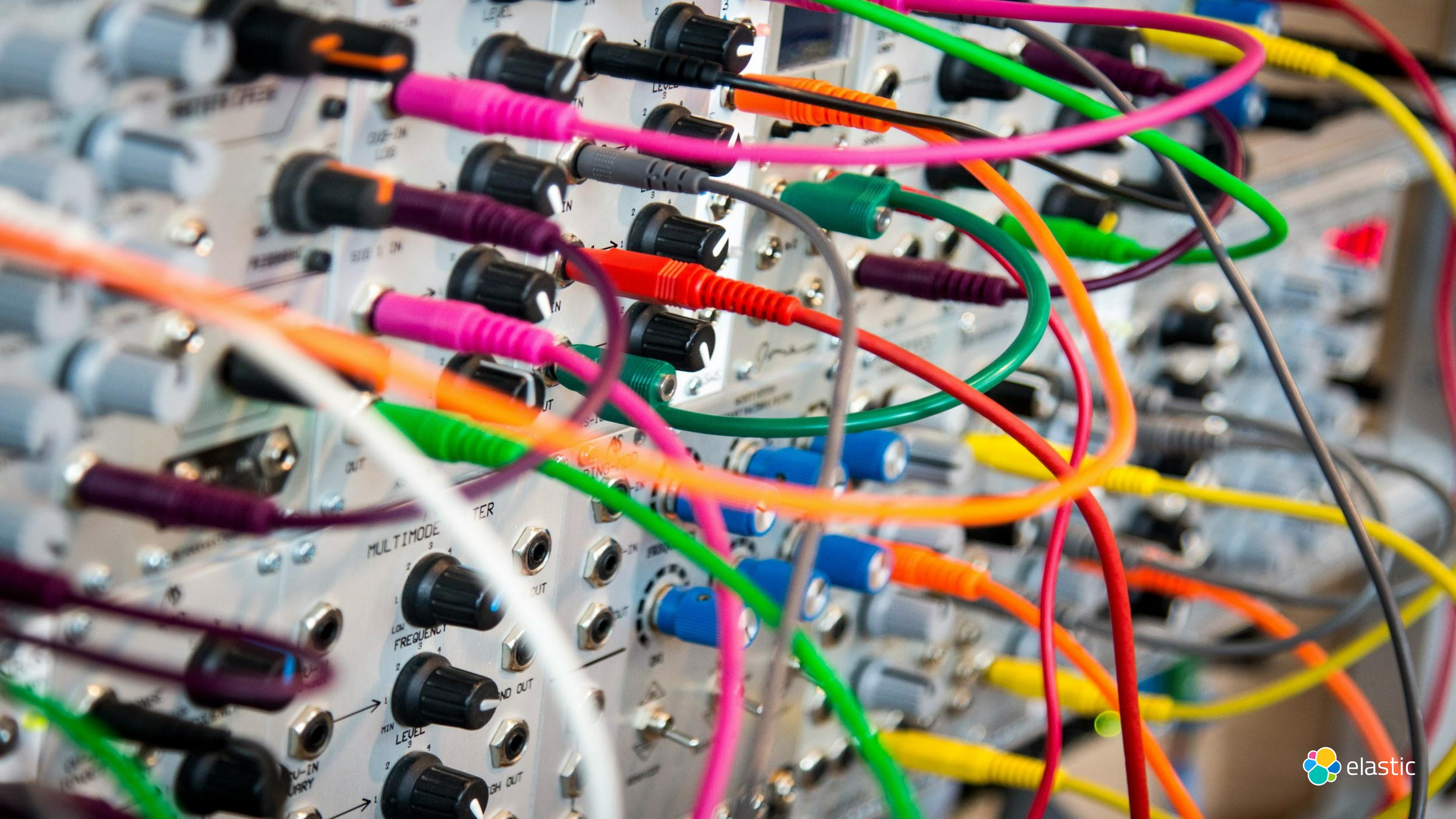
```
> * 2024-05-10 08:00:06.117   55   54440319   ...   http://api.nasa.gov/neo/rest/v1/neo/sentry/544...
> * 2024-05-10 08:00:06.117   56   54440642   ...                                                 NaN
v * 2024-05-10 08:00:06.117   57   54440644   ...   http://api.nasa.gov/neo/rest/v1/neo/sentry/544...
```

Copy ▾ | Similar entries ▾ | Expand nested fields | Hide log summary


```
{
  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]
```

elastic


MULTIMODE
FREQUENCY
LEVEL
OUT
elastic

Future proof

elastic

Benefits

elastic

- 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.

elastic

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?!?!

elastic


elastic

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

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

elastic

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"}
```

elastic

Root span

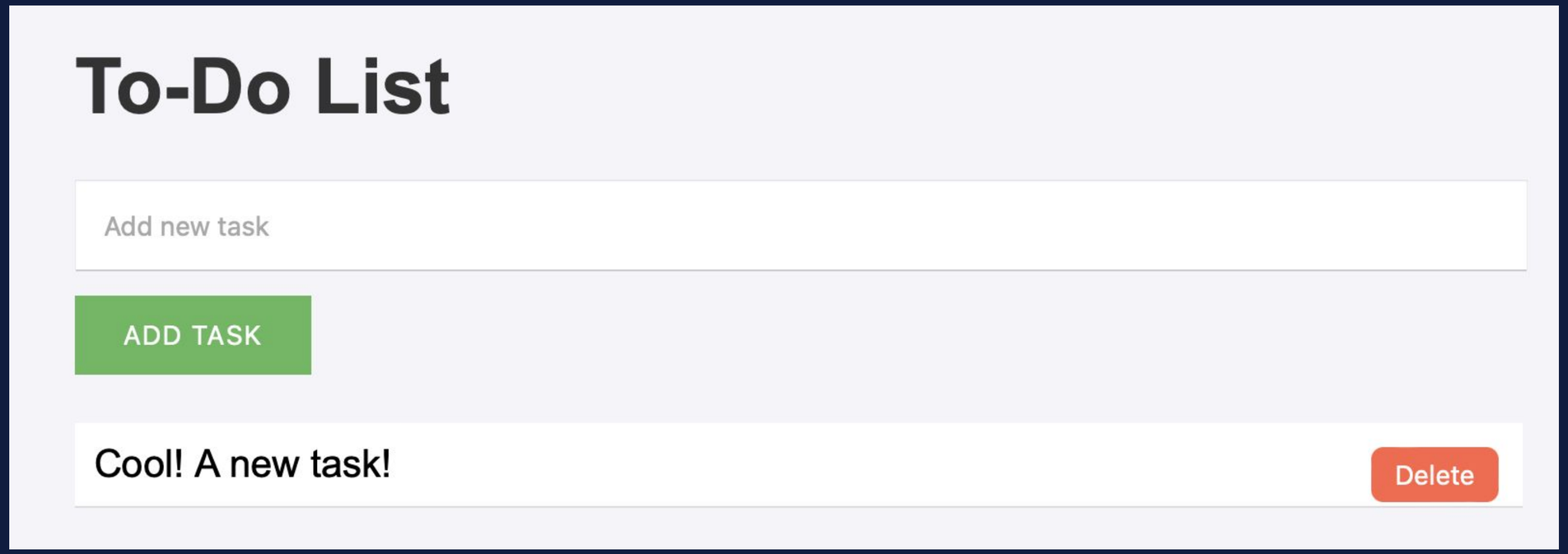
```
Span #1
    Trace ID          : 4e6b8d4dcd44d9245285a9459a0114e6
    Parent ID         :
    ID                : 6c2171f87b96401f
    Name              : /
    Kind              : Server
    Start time        : 2024-05-14 19:07:24.894406 +0000 UTC
    End time          : 2024-05-14 19:07:24.897374 +0000 UTC
    Status code       : Unset
    Status message :
```

elastic

Demo in Flask

elastic

Let’s start with the classic Flask example



elastic

Let’s instrument our app

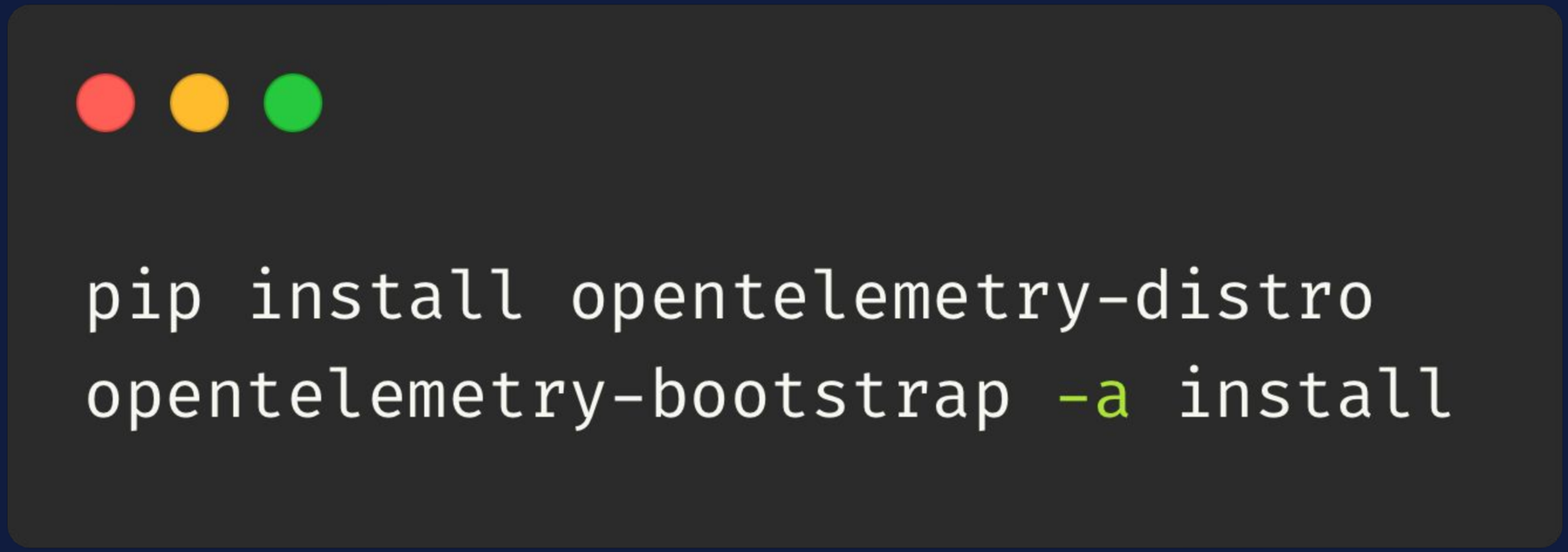
elastic

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

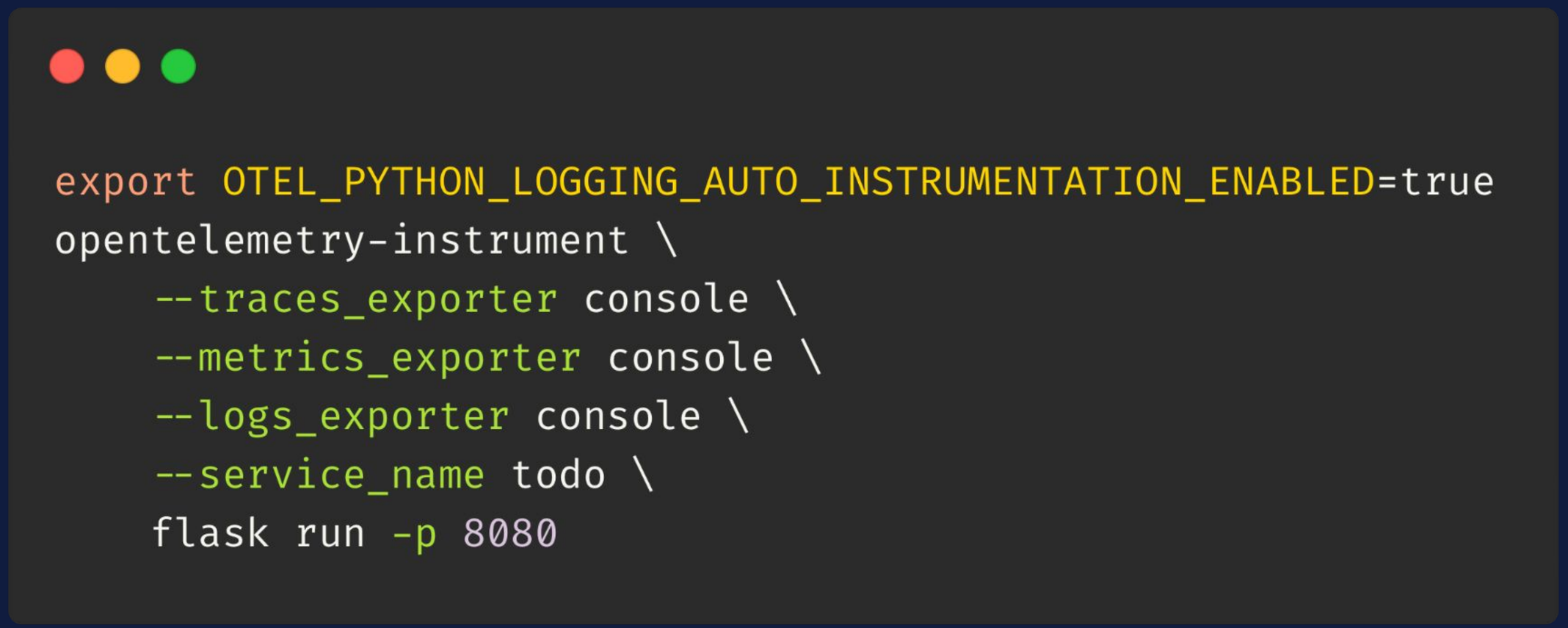
elastic

The command line is just one of the ways you can instrument your application. Other options include serverless functions and kubernetes.

elastic


pip install opentelemetry-distro
opentelemetry-bootstrap -a install

elastic


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

elastic

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

elastic

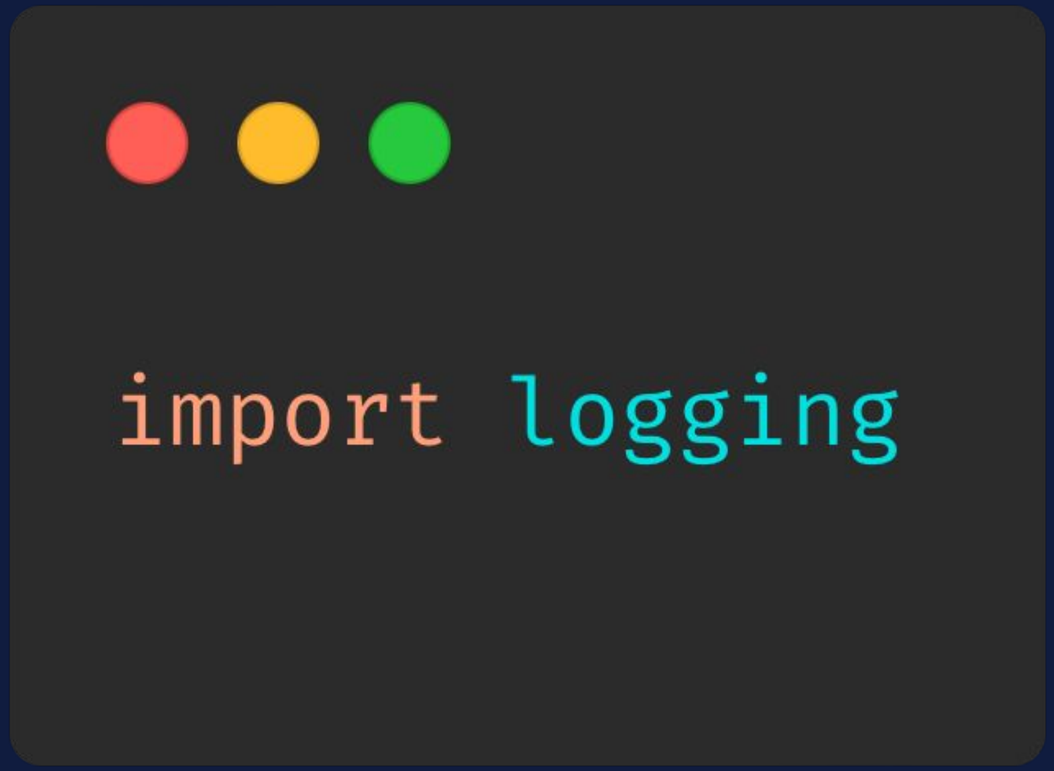
So when would you want to use manual instrumentation?

Manually instrumenting our application

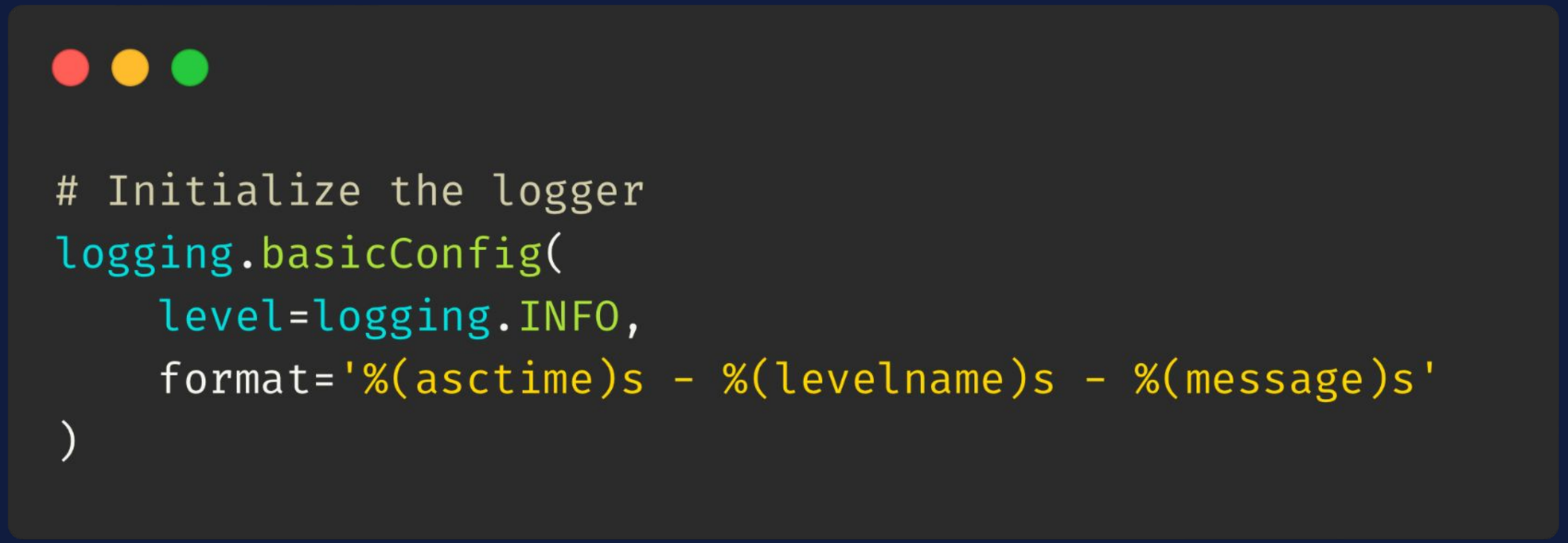
elastic

Logs

elastic



elastic

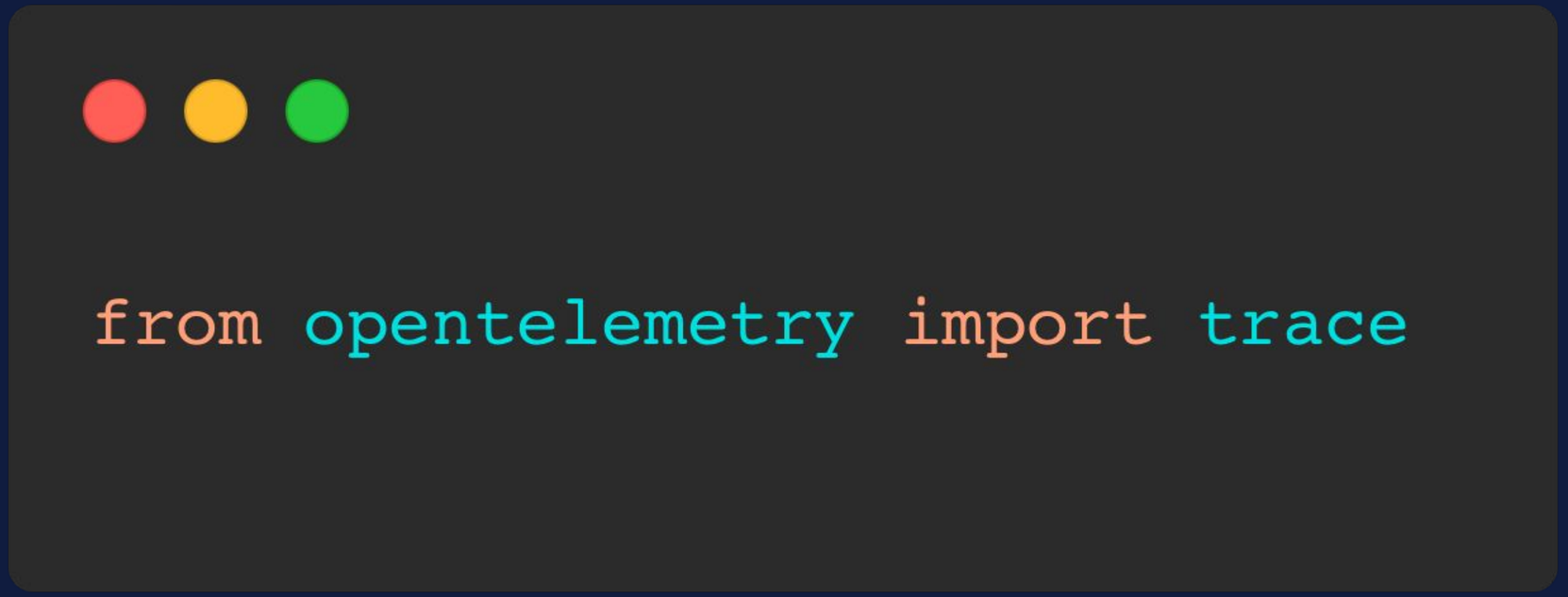


```
# Initialize the logger
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)
```

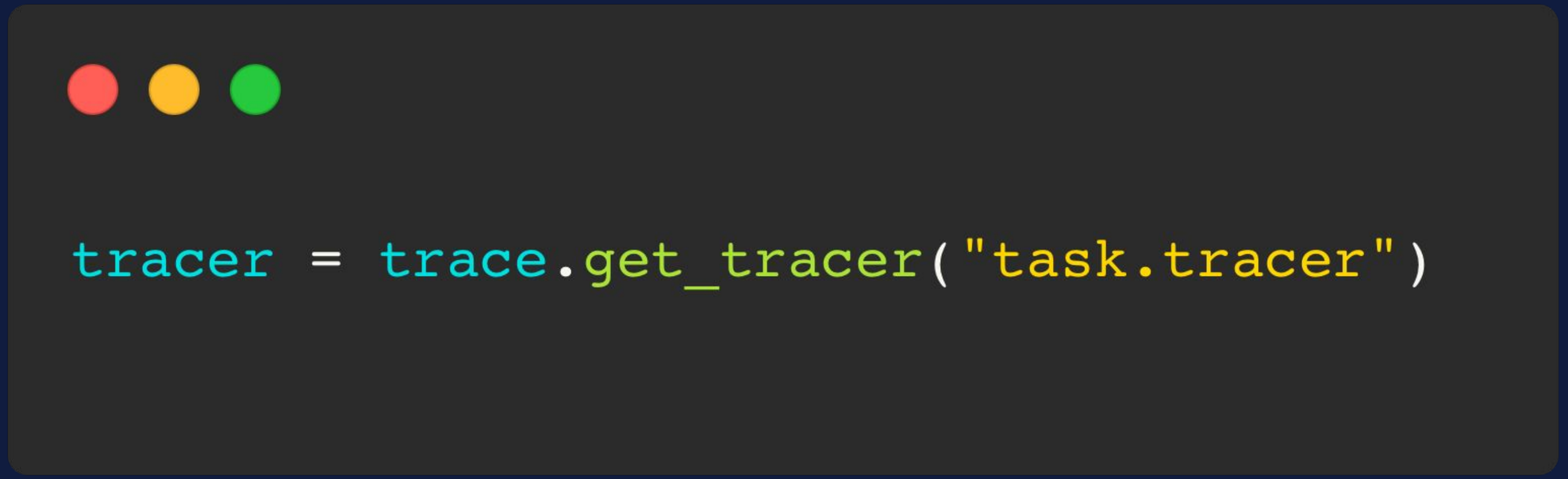
```python
@app.route('/', methods=['GET'])
def home():
    tasks = Task.query.all()
    logging.info(f'Fetched {len(tasks)} tasks from the database')
    return render_template_string(HOME_HTML, tasks=tasks)

@app.route('/add', methods=['POST'])
def add():
    task_description = request.form['task']
    new_task = Task(description=task_description)
    db.session.add(new_task)
    db.session.commit()
    logging.info(f'Added new task with description: {task_description}')
    return redirect(url_for('home'))
```

Traces

elastic



elastic

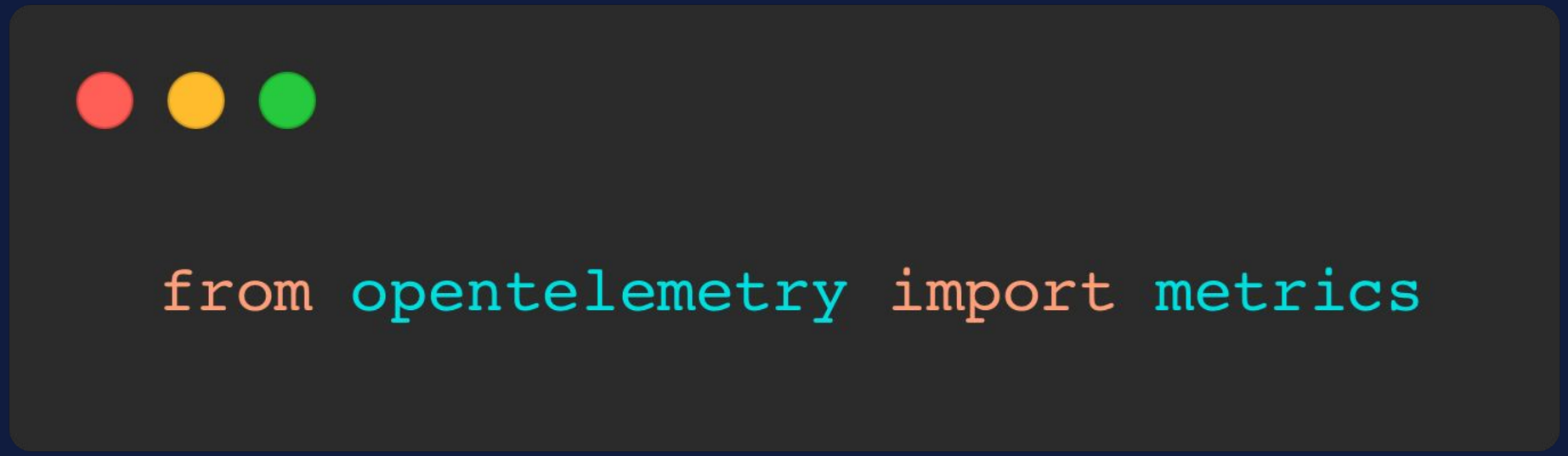


elastic

```python
@app.route('/', methods=['GET'])
def home():
    with tracer.start_as_current_span("querying_tasks"):
        tasks = Task.query.all()
    return render_template_string(HOME_HTML, tasks=tasks)

@app.route('/add', methods=['POST'])
def add():
    task_description = request.form['task']
    new_task = Task(description=task_description)
    with tracer.start_as_current_span("adding_task"):
        db.session.add(new_task)
        db.session.commit()
    return redirect(url_for('home'))
```

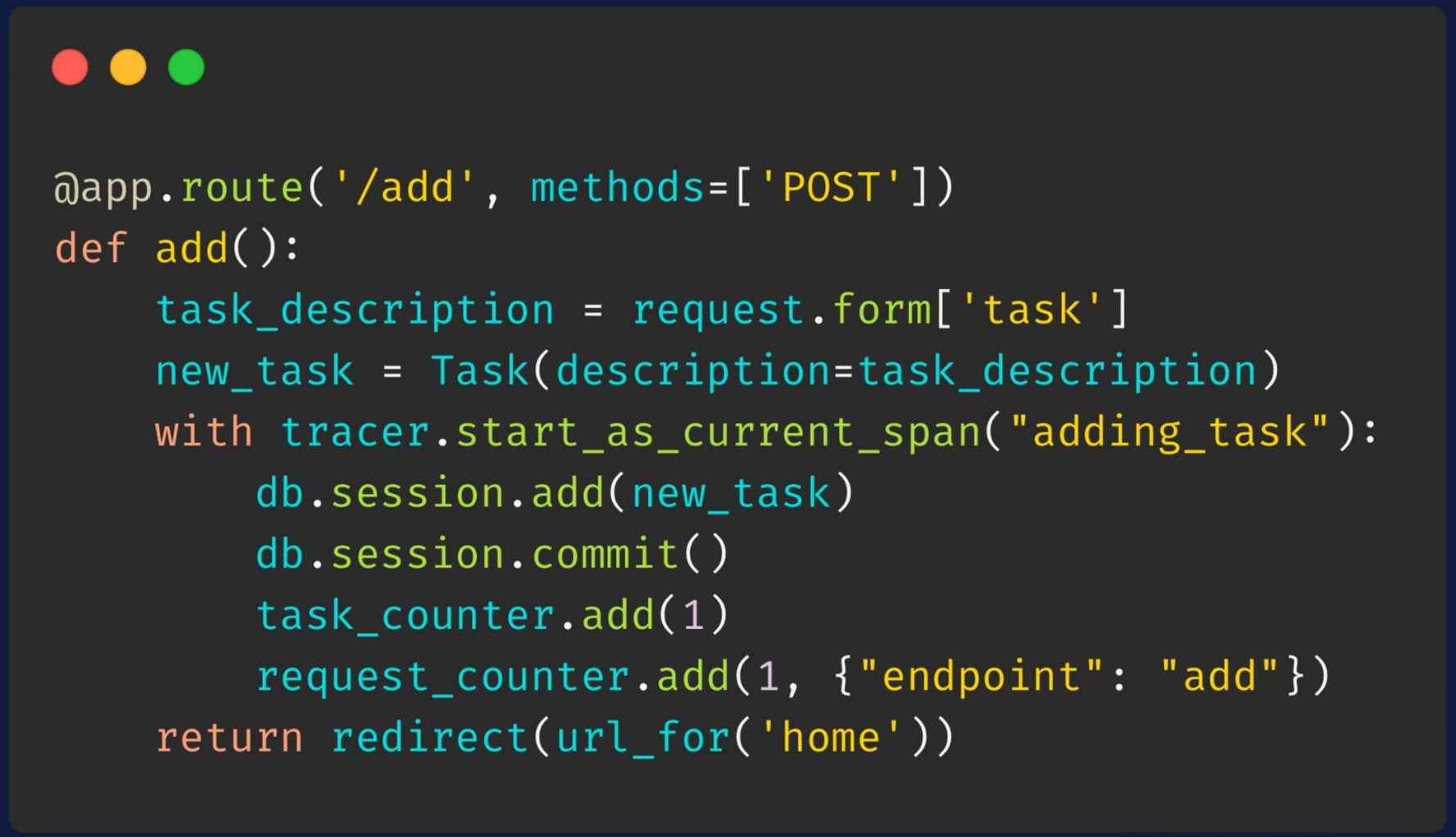
Metrics

elastic



elastic

```python
meter = metrics.get_meter("task.meter")

request_counter = meter.create_counter(
    "flask_app_requests_total",
    description="Total number of requests",
)

task_counter = meter.create_counter(
    "flask_app_tasks_total",
    description="Total number of tasks created",
)
```



```python
@app.route('/add', methods=['POST'])
def add():
    task_description = request.form['task']
    new_task = Task(description=task_description)
    with tracer.start_as_current_span("adding_task"):
        db.session.add(new_task)
        db.session.commit()
        task_counter.add(1)
        request_counter.add(1, {"endpoint": "add"})
    return redirect(url_for('home'))
```

How does Elastic fit into this?

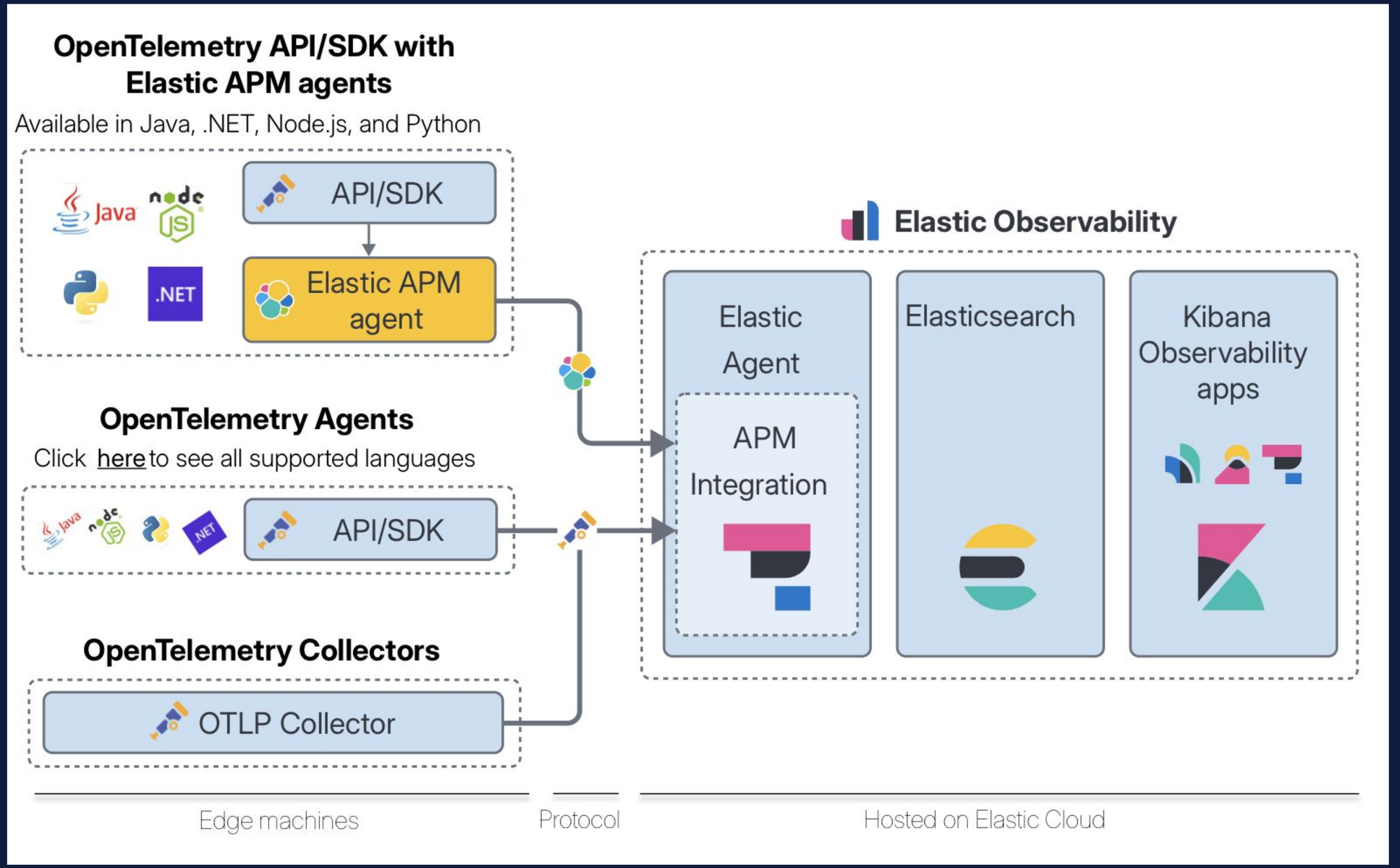
elastic

What is Elastic?

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.

elastic


OpenTelemetry API/SDK with Elastic APM agents
Available in Java, .NET, Node.js, and Python
Java
node
.NET
API/SDK
Elastic APM agent
OpenTelemetry Agents
Click here to see all supported languages
API/SDK
OpenTelemetry Collectors
OTLP Collector
Elastic Observability
Elastic Agent
APM Integration
Elasticsearch
Kibana Observability apps
Edge machines
Protocol
Hosted on Elastic Cloud

elastic

Sending to a collector on Docker

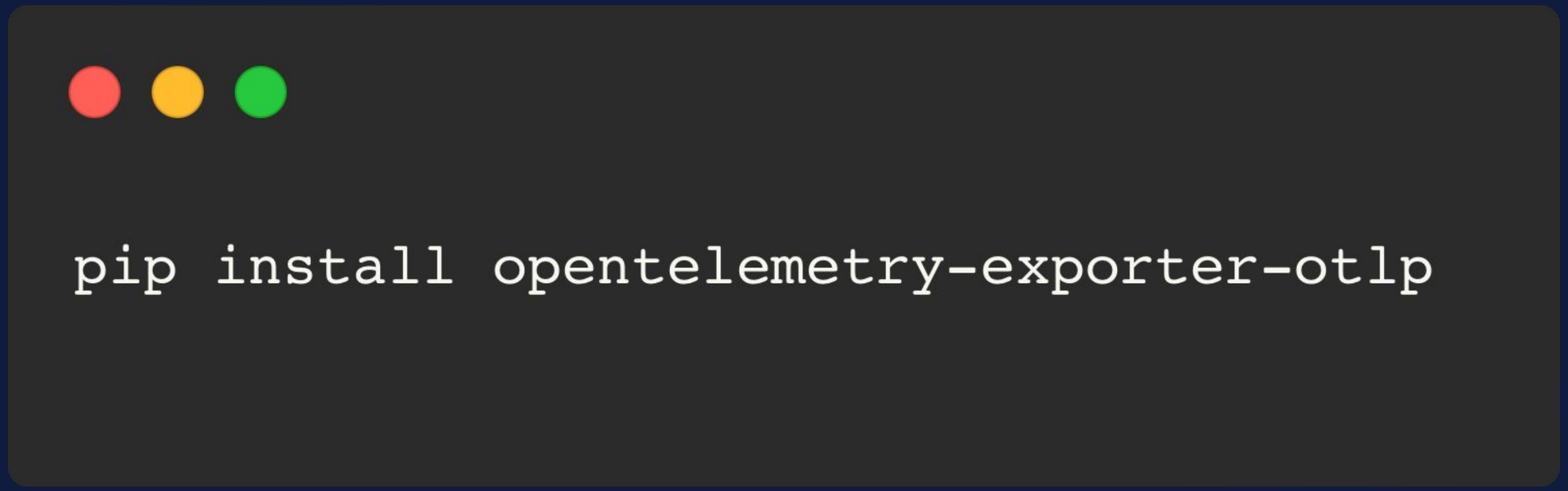
elastic

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

```yaml
 # /tmp/otel-collector-config.yaml
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318

processors:
  batch:
  memory_limiter:
    check_interval: 1s
    limit_mib: 2000

exporters:
  debug:
    verbosity: detailed
  otlp/elastic:
    # Elastic APM server https endpoint without the "https://" prefix
    endpoint: "${env:ELASTIC_APM_SERVER_ENDPOINT}"
    headers:
      # Elastic APM Server secret token
      Authorization: "Bearer ${env:ELASTIC_APM_SECRET_TOKEN}"

service:
  pipelines:
    traces:
      receivers: [otlp]
      exporters: [debug, otlp/elastic]
    metrics:
      receivers: [otlp]
      exporters: [debug, otlp/elastic]
    logs:
      receivers: [otlp]
      exporters: [debug, otlp/elastic]
```

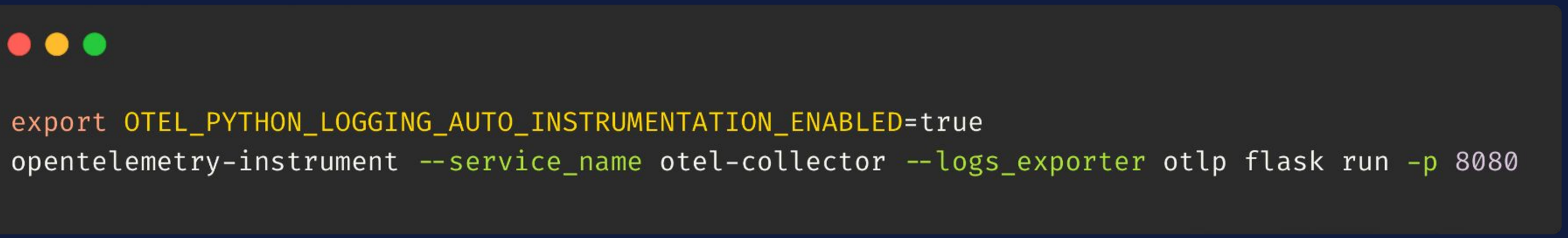
You can configure your collector via a YAML file

elastic



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
```



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

elastic

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

elastic

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

elastic

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.

State of adoption

elastic

https://opentelemetry.io/status/

Closing thoughts

elastic

OpenTelemetry (OTel) is an open-source, vendor-neutral observability framework designed to integrate with any backend system. It offers standardized APIs, libraries, and tools to gather telemetry data, including metrics, logs, and traces.

elastic

It's very easy to start instrumenting your code.

It is highly configurable and extensible

It scales well from large applications to small applications

elastic

Next steps

elastic

Resources

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

elastic

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


Thank you!
elastic