Hands-on-Lab: Automated Instrumentation with OpenTelemetry

Estimated Time Needed:45 mins

Getting Started

In this lab, you will learn how to automate instrumentation using OpenTelemetry. OpenTelemetry is an observability framework, an Application programming interface (API), Software Development Kit (SDK), and tools openTelementry aids in generating and collecting application telemetry data such as metrics, logs, and traces.

Developers can use OpenTelemetry tracing to create spans representing a timed code block. Each span includes key-value pairs, attributes that help describe what the span means, links to other spans, and events that represent timestamps within the span. Python agent, which can be attached to any Python application, is used for automatic instrumentation. It dynamically injects bytecode to capture telemetry from many popular libraries and frameworks.

> Note: The working knowledge of Python is not needed to complete this lab, as you already get the code. This lab is an example of how to instrument any application code with OpenTelemetry automatically.

Learning Objectives:

After completing this lab, you will be able to:

- Create a Python virtual environment
- Instrument the app automatically using OpenTelemetry libraries

About Skills Network Cloud IDE

Skills Network Cloud integrated development environment (based on Theia) provides an environment for hands-on labs for the course and project-related labs. Theia is an open-source integrated development environment (IDE) that can be run on the desktop or cloud. To complete this lab, you will use the Cloud IDE (based on Theia and MongoDB) running in a Docker container.

Important Notice about this Lab Environment

Please note that you cannot save or persist the sessions for this lab environment. Every time you connect to this lab, a new environment is created for you. Any data you may have saved in the earlier session would be lost. Plan to complete both exercises given in the lab in a single session to avoid losing your data.

Exercise 1: Creating a Python Virtual Environment

A virtual environment for Python is an isolated working copy of the language that enables working on a specific project without affecting others.

As a result, it is a tool that allows multiple Python installations to run concurrently, one for each project.

1. Open a terminal window from the menu in the editor: Terminal > New Terminal.

2. In the terminal, you will run all the commands to complete the lab.

3. In the terminal, run the command below to install the Python virtual environment

   ```
   pip install virtualenv
   ```

4. Run the command below to check the version of your installation.

   ```
   virtualenv --version
   ```

5. Create a virtual environment now by running the below command.

```
virtualenv telemetryenv
```

6. Activate the virtual environment using the below command.

```
source telemetryenv/bin/activate
```

Exercise 2: Instrumenting the App Automatically Using OpenTelemetry Libraries

Follow the steps below to instrument your Flask application automatically using OpenTelemetry libraries:

1. Install the opentelemetry-instrumentation-flask and flask packages using your Python package manager. You can use pip for this:

```
pip install opentelemetry-instrumentation-flask flask
```

2. Create app.py with the command

```
touch app.py
```

In the explorer, you can view the app.py file. You need to click and open it.

3. To instrument app.py python code, add the following OpenTelemetry setup to create traces and spans, and then export them to the console. Add this code to the top of your Python app.

```
from opentelemetry import trace
from flask import Flask, request
from opentelemetry.sdk.resources import Resource
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor
from random import randint
from opentelemetry.sdk.trace.export import ConsoleSpanExporter
app = Flask(__name__)
```

```
provider = TracerProvider()
processor = BatchSpanProcessor(ConsoleSpanExporter())
provider.add_span_processor(processor)
trace.set_tracer_provider(provider)
tracer = trace.get_tracer(__name__)
```

4. Three concepts are seen here: provider, processor, and tracer.

- A provider: The API entry point that holds configuration. In this case, it is a TracingProvider.
- A processor: The method of sending the created elements or spans onward.
- A tracer: An actual object that creates the spans.

The code creates a provider, adds a processor, then configures the local tracing environment to use them.

Add the code below after installing the setup code in app.py

```
@app.route("/roll")
def roll():
  with tracer.start_as_current_span(
    "server_request",
    attributes={ "endpoint": "/roll" }
  ):
    sides = int(request.args.get('sides'))
    rolls = int(request.args.get('rolls'))
    return roll_sum(sides,rolls)
def roll_sum(sides, rolls):
  span = trace.get_current_span()
  sum = 0
  for r in range(0,rolls):
    result = randint(1,sides)
    span.add_event( "log", {
      "roll.sides": sides,
      "roll.result": result,
    })
    sum += result
  return  str(sum)
```

5. Your final app.py should look like this:

```
from opentelemetry import trace
from flask import Flask,request
from opentelemetry.sdk.resources import Resource
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor
from random import randint
from opentelemetry.sdk.trace.export import ConsoleSpanExporter
app = Flask(__name__)
provider = TracerProvider()
processor = BatchSpanProcessor(ConsoleSpanExporter())
provider.add_span_processor(processor)
trace.set_tracer_provider(provider)
tracer = trace.get_tracer(__name__)
@app.route("/roll")
def roll():
  with tracer.start_as_current_span(
    "server_request",
    attributes={ "endpoint": "/roll" }
  ):
    sides = int(request.args.get('sides'))
    rolls = int(request.args.get('rolls'))
    return roll_sum(sides,rolls)
def roll_sum(sides, rolls):
  span = trace.get_current_span()
  sum = 0
  for r in range(0,rolls):
    result = randint(1,sides)
    span.add_event( "log", {
      "roll.sides": sides,
      "roll.result": result,
    })
    sum += result
  return  str(sum)
```

Now, click the File from the menu and save the app.py file. See instructions below on how to complete this.

6. You must also ensure you have the opentelemetry-distro Python package installed by running the following command.

Note: The opentelemetry-distro can pull in the SDK and the API and make the opentelemetry-bootstrap and opentelemetry-instrument commands available.

```
pip install opentelemetry-distro \
  opentelemetry-exporter-otlp
opentelemetry-bootstrap -a install
```

7. Run the opentelemetry-instrument command with the appropriate parameters to enable auto-instrumentation. In this example, you will use Flask run as the method to run the script:

```
opentelemetry-instrument --traces_exporter console flask run
```

The –traces_exporter flag specifies the exporter where the traces will go. In this case, you are using the console exporter, which outputs the traces to the console. By modifying the command accordingly, you can replace it with your desired exporter, such as Jaeger or another OpenTelemetry processor.

8. Send requests to your Flask application endpoints using curl or web browser tools.

Open a split terminal window using the editor's menu: Terminal > Split Terminal

```
curl -X GET "http://localhost:5000/roll?sides=6&rolls=3"
```

9. Go back to the previous terminal to view the output.

Observe the automatically generated tracing output. The output will include a trace ID, span ID, timestamps, span kind, parent ID, status, attributes, events, links, and resource information.

The attributes will provide information about the HTTP request, such as the HTTP method, server name, scheme, host, target, user agent, peer IP and port, request route, and status code.

By following these steps, you can instrument your Flask application automatically using OpenTelemetry, and you will be able to gather distributed tracing information without manually adding instrumentation code.

Summary

Congratulations! You just completed the Automated Instrumentation with OpenTelemetry.

You learned how to create a Python virtual environment. You also learned to instrument the app automatically using OpenTelemetry libraries. You also explored how OpenTelemetry, an observability framework, aids in generating and collecting application telemetry data such as metrics, logs, and traces.

Author(s)
Shivam Kumar

Contributor
Pallavi