Uygulamalı Laboratuvar: OpenTelemetry ile Otomatik Enstrümantasyon

Tahmini Gerekli Süre: 45 dk

Başlarken

Bu laboratuvarda, OpenTelemetry kullanarak enstrümantasyonu otomatikleştirmeyi öğreneceksiniz. OpenTelemetry, gözlemlenebilirlik çerçevesi, bir Uygulama Programlama Arayüzü (API), Yazılım Geliştirme Kiti (SDK) ve uygulama telemetri verilerini, örneğin metrikler, günlükler ve izler, oluşturma ve toplama konusunda yardımcı olan araçlardır.

Geliştiriciler, zamanlanmış bir kod bloğunu temsil eden span'ler oluşturmak için OpenTelemetry izleme özelliğini kullanabilirler. Her span, span'in ne anlama geldiğini tanımlamaya yardımcı olan anahtar-değer çiftlerini, diğer span'lere bağlantıları ve span içindeki zaman damgalarını temsil eden olayları içerir. Herhangi bir Python uygulamasına eklenebilen Python ajanı, otomatik enstrümantasyon için kullanılır. Bu ajan, birçok popüler kütüphane ve çerçeveden telemetri yakalamak için dinamik olarak bytecode enjekte eder.

> Not: Bu laboratuvarı tamamlamak için Python bilgisine sahip olmanız gerekmiyor, çünkü kodu zaten alıyorsunuz. Bu laboratuvar, OpenTelemetry ile herhangi bir uygulama kodunu otomatik olarak nasıl enstrümante edeceğinize dair bir örnektir.

Öğrenme Hedefleri:

Bu laboratuvarı tamamladıktan sonra şunları yapabileceksiniz:

- Python sanal ortamı oluşturmak
- OpenTelemetry kütüphanelerini kullanarak uygulamayı otomatik olarak enstrümante etmek

Skills Network Cloud IDE Hakkında

Skills Network Cloud entegre geliştirme ortamı (Theia tabanlı) kurs ve proje ile ilgili laboratuvarlar için uygulamalı laboratuvarlar sunar. Theia, masaüstünde veya bulutta çalıştırılabilen açık kaynaklı bir entegre geliştirme ortamıdır. Bu laboratuvarı tamamlamak için, Docker konteynerinde çalışan Cloud IDE'yi (Theia ve MongoDB tabanlı) kullanacaksınız.

Bu Laboratuvar Ortamı Hakkında Önemli Bilgi

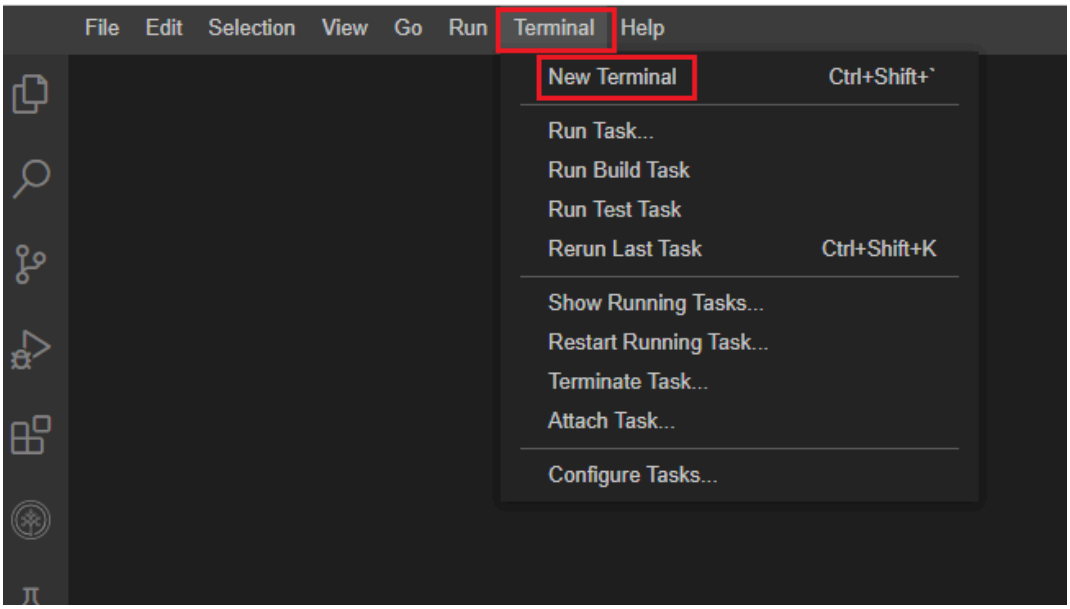
Lütfen bu laboratuvar ortamı için oturumları kaydedemeyeceğinizi veya sürdüremeyeceğinizi unutmayın. Bu laboratuvara her bağlandığınızda, sizin için yeni bir ortam oluşturulur. Önceki oturumda kaydettiğiniz veriler kaybolacaktır. Verilerinizi kaybetmemek için laboratuvar içinde verilen her iki egzersizi de tek bir oturumda tamamlamayı planlayın.

Alıştırma 1: Python Sanal Ortamı Oluşturma

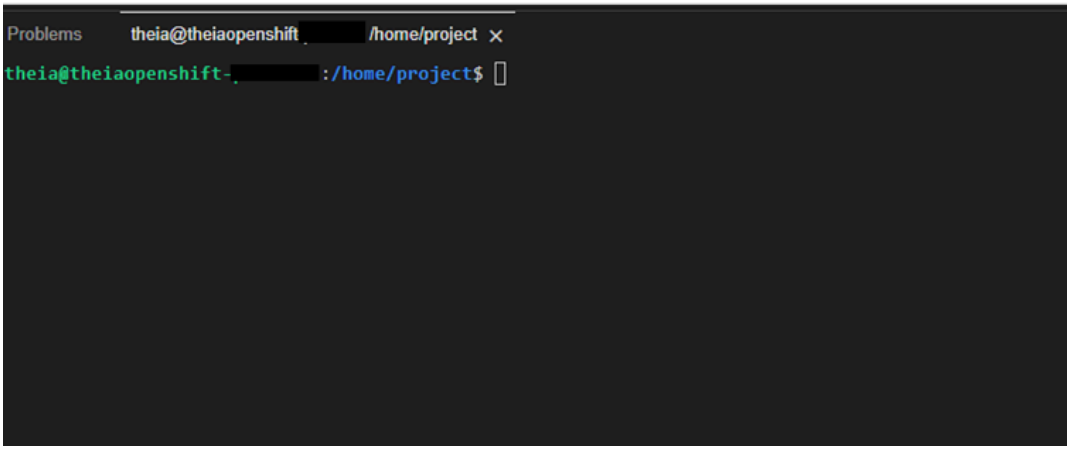
Python için bir sanal ortam, dilin izole bir çalışma kopyasıdır ve belirli bir proje üzerinde çalışmayı, diğerlerini etkilemeden gerçekleştirmenizi sağlar.

Sonuç olarak, her proje için bir tane olmak üzere birden fazla Python kurulumunun aynı anda çalışmasına olanak tanıyan bir araçtır.

1. Editördeki menüden bir terminal penceresi açın: Terminal > Yeni Terminal.

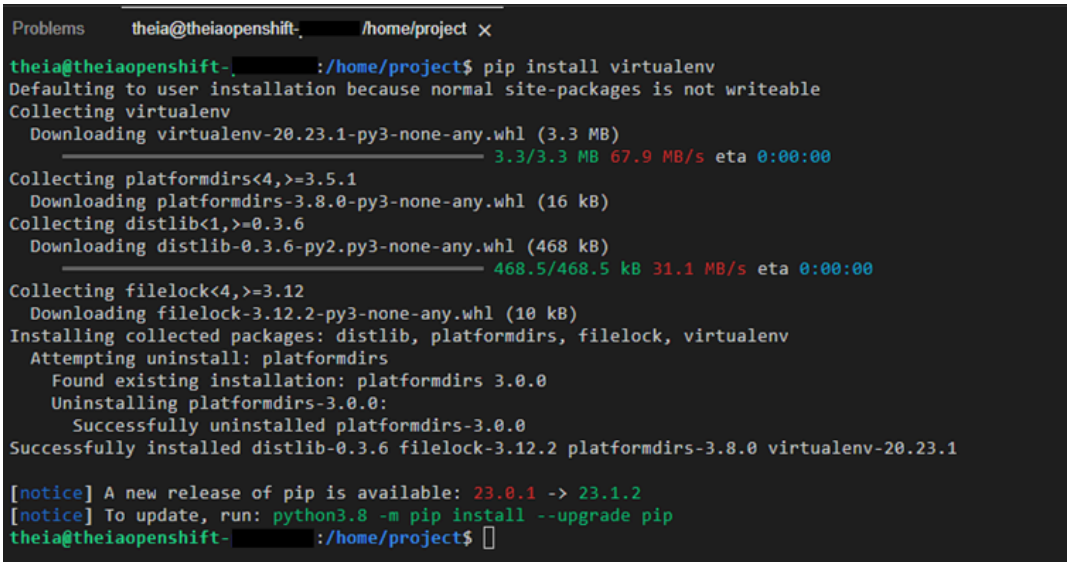


2. Terminalde, laboratuvarı tamamlamak için tüm komutları çalıştıracaksınız.



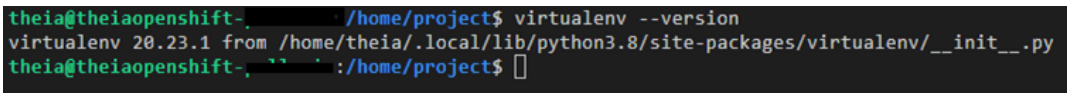
3. Terminalde, Python sanal ortamını kurmak için aşağıdaki komutu çalıştırın.

```
pip install virtualenv
```



4. Kurulumunuzun sürümünü kontrol etmek için aşağıdaki komutu çalıştırın.

```
virtualenv --version
```



5. Aşağıdaki komutu çalıştırarak şimdi bir sanal ortam oluşturun.

```
virtualenv telemetryenv
```

```
theia@theiaopenshift-[redacted]:/home/project$ virtualenv virtualenv_name
created virtual environment CPython3.8.0.final.0-64 in 1333ms
  creator CPython3Posix(dest=/home/project/virtualenv_name, clear=False, no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=/home/theia/.local/share/virtualenv)
    added seed packages: pip==23.1.2, setuptools==67.8.0, wheel==0.40.0
  activators BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator
```

6. Aşağıdaki komutu kullanarak sanal ortamı etkinleştirin.

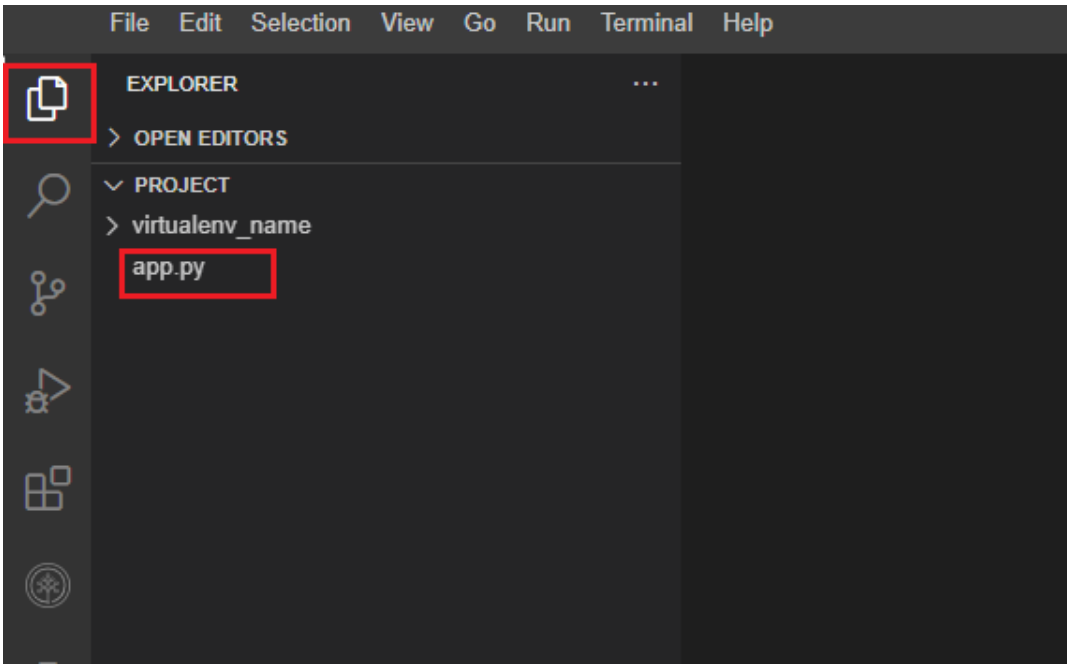
```
source telemetryenv/bin/activate
```

```
<img src="https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-CD0223EN-SkillsNetwork/labs/module4/images/opentelemetry6.png" width="75%">
::page{title="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:
```bash
pip install opentelemetry-instrumentation-flask flask
```

```
(virtualenv_name) theia@theiaopenshift-[redacted]:/home/project$ pip install opentelemetry-instrumentation-flask flask
Collecting opentelemetry-instrumentation-flask
  Downloading opentelemetry_instrumentation_flask-0.39b0-py3-none-any.whl (13 kB)
Collecting flask
  Downloading Flask-2.3.2-py3-none-any.whl (96 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 96.9/96.9 kB 15.3 MB/s eta 0:00:00
Collecting opentelemetry-api~=1.12 (from opentelemetry-instrumentation-flask)
  Downloading opentelemetry_api-1.18.0-py3-none-any.whl (57 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 57.3/57.3 kB 9.3 MB/s eta 0:00:00
Collecting opentelemetry-instrumentation-wsgi==0.39b0 (from opentelemetry-instrumentation-flask)
  Downloading opentelemetry_instrumentation_wsgi-0.39b0-py3-none-any.whl (12 kB)
Collecting opentelemetry-instrumentation==0.39b0 (from opentelemetry-instrumentation-flask)
  Downloading opentelemetry_instrumentation-0.39b0-py3-none-any.whl (24 kB)
Collecting opentelemetry-semantic-conventions==0.39b0 (from opentelemetry-instrumentation-flask)
  Downloading opentelemetry_semantic_conventions-0.39b0-py3-none-any.whl (26 kB)
Collecting opentelemetry-util-http==0.39b0 (from opentelemetry-instrumentation-flask)
  Downloading opentelemetry_util_http-0.39b0-py3-none-any.whl (6.7 kB)
Requirement already satisfied: setuptools>=16.0 in ./virtualenv_name/lib/python3.8/site-packages (from opentelemetry-instrumentation==0.39b0->opentel
etry-instrumentation-flask) (67.8.0)
Collecting wrapt<2.0.0,>=1.0.0 (from opentelemetry-instrumentation==0.39b0->opentelemetry-instrumentation-flask)
  Downloading wrapt-1.15.0-cp38-cp38-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (81 kB)
```

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.

```
```python
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. Burada üç kavram görülmektedir: sağlayıcı, işlemci ve izleyici.

- Bir sağlayıcı: Yapılandırmayı tutan API giriş noktasıdır. Bu durumda, bir TracingProvider'dır.
- Bir işlemci: Oluşturulan öğeleri veya span'leri iletme yöntemidir.
- Bir izleyici: Span'leri oluşturan gerçek nesnedir.

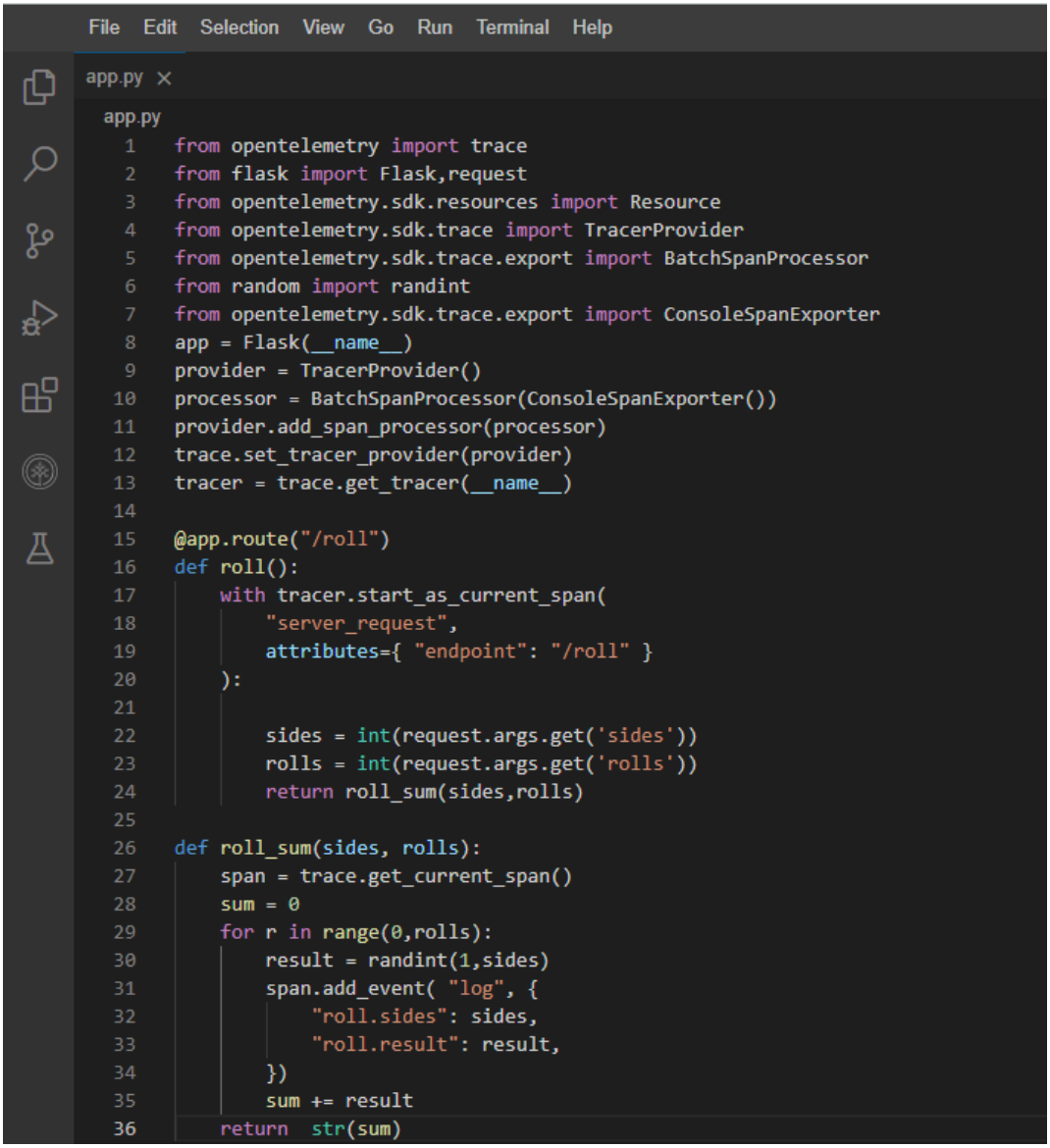
Kod, bir sağlayıcı oluşturur, bir işlemci ekler ve ardından yerel izleme ortamını bunları kullanacak şekilde yapılandırır.

app.py dosyasındaki kurulum kodunu ekledikten sonra aşağıdaki kodu ekleyin.

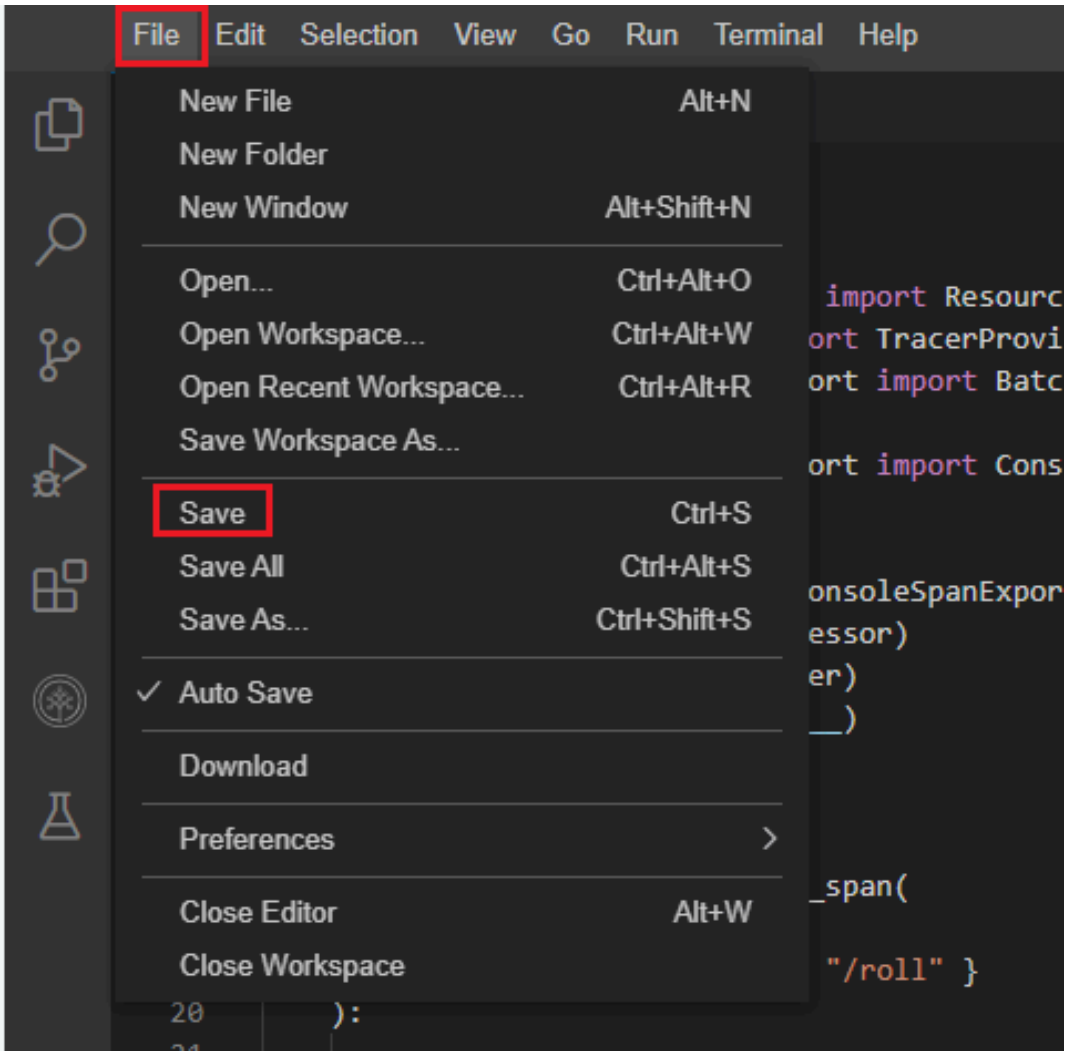
```
@app.route("/roll")
def roll():
  with tracer.start_as_current_span(
    "server_request",
    attributes={ "endpoint": "/roll" }
  ):
    sides = int(request.args.get('sides'))
    rolls = int(request.args.get('rolls'))
    return roll_sum(sides,rolls)
def roll_sum(sides, rolls):
  span = trace.get_current_span()
  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)
```
```



Şimdi, menüden Dosya'ya tıklayın ve app.py dosyasını kaydedin. Aşağıda bunu tamamlamak için talimatları görebilirsiniz.



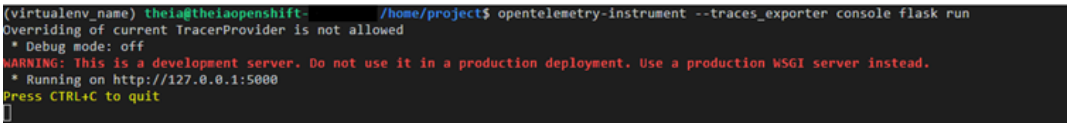
6. Ayrıca, aşağıdaki komutu çalıştırarak opentelemetry-distro Python paketinin yüklü olduğundan emin olmalısınız.

Not: opentelemetry-distro, SDK ve API'yi çekebilir ve opentelemetry-bootstrap ve opentelemetry-instrument komutlarını kullanılabilir hale getirebilir.

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

7. Auto-enstrümantasyonu etkinleştirmek için uygun parametrelerle `opentelemetry-instrument` komutunu çalıştırın. Bu örnekte, betiği çalıştırmak için Flask run yöntemini kullanacaksınız:

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



–traces_exporter bayrağı, izlerin gideceği ihracatı belirtir. Bu durumda, izleri konsola çıktısını veren konsol ihracatçısını kullanıyorsunuz. Komutu buna göre değiştirerek, istediğiniz ihracatçı ile değiştirebilirsiniz, örneğin Jaeger veya başka bir OpenTelemetry işlemcisi.

8. Flask uygulamanızın uç noktalarına curl veya web tarayıcı araçları kullanarak istek gönderin.

Editörün menüsünden bir bölünmüş terminal penceresi açın: Terminal > Split Terminal

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

```
theia@theiaopenshift-[redacted]:/home/project$ curl -X GET "http://localhost
:5000/roll?sides=6&rolls=3"
13theia@theiaopenshift-[redacted]:/home/project$
```

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

```
127.0.0.1 - - [29/Jun/2023 01:10:45] "GET /roll?sides=6&rolls=3 HTTP/1.1" 200 -
{
    "name": "server_request",
    "context": {
        "trace_id": "0xf857eafa570e4c4874092a2708abb9e0",
        "span_id": "0xfffcbdcba70db7c6",
        "trace_state": "[]"
    },
    "kind": "SpanKind.INTERNAL",
    "parent_id": "0x0d253270eea1a8c1",
    "start_time": "2023-06-29T05:10:45.037621Z",
    "end_time": "2023-06-29T05:10:45.037745Z",
    "status": {
        "status_code": "UNSET"
    },
    "attributes": {
        "endpoint": "/roll"
    },
    "events": [
        {
            "name": "log",
            "timestamp": "2023-06-29T05:10:45.037710Z",
            "attributes": {
                "roll.sides": 6,
                "roll.result": 6
            }
        },
        {
            "name": "log",
            "timestamp": "2023-06-29T05:10:45.037725Z",
            "attributes": {
                "roll.sides": 6,
                "roll.result": 1
            }
```

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