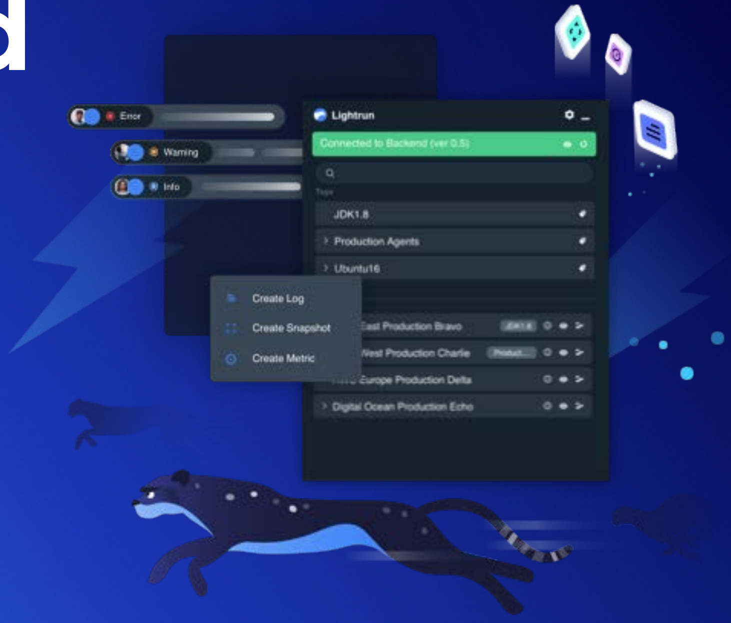
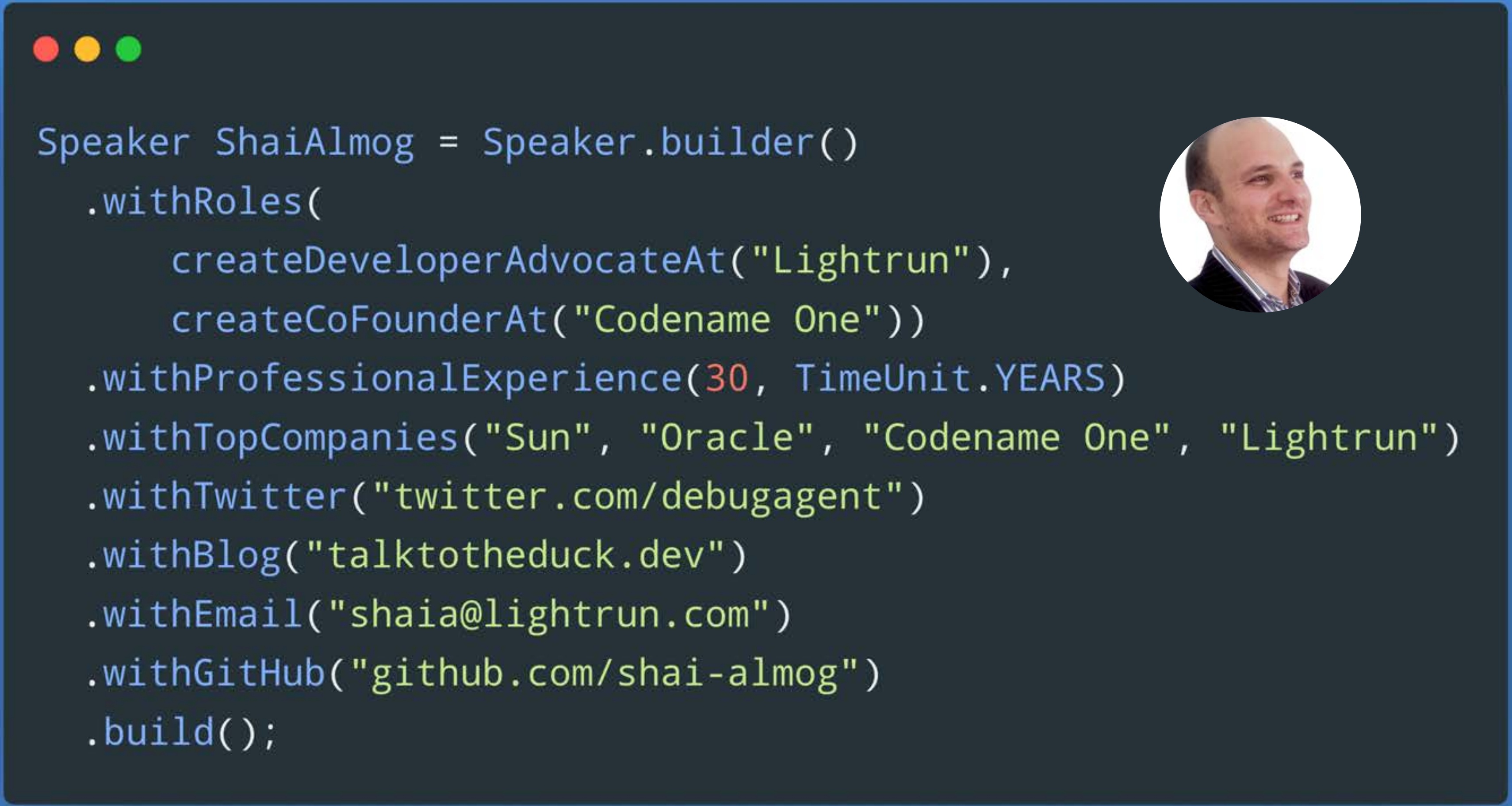
Lightrun

The Polyglot Cloud Native Debugger - Going Beyond APM

Shai Almog

January 2022

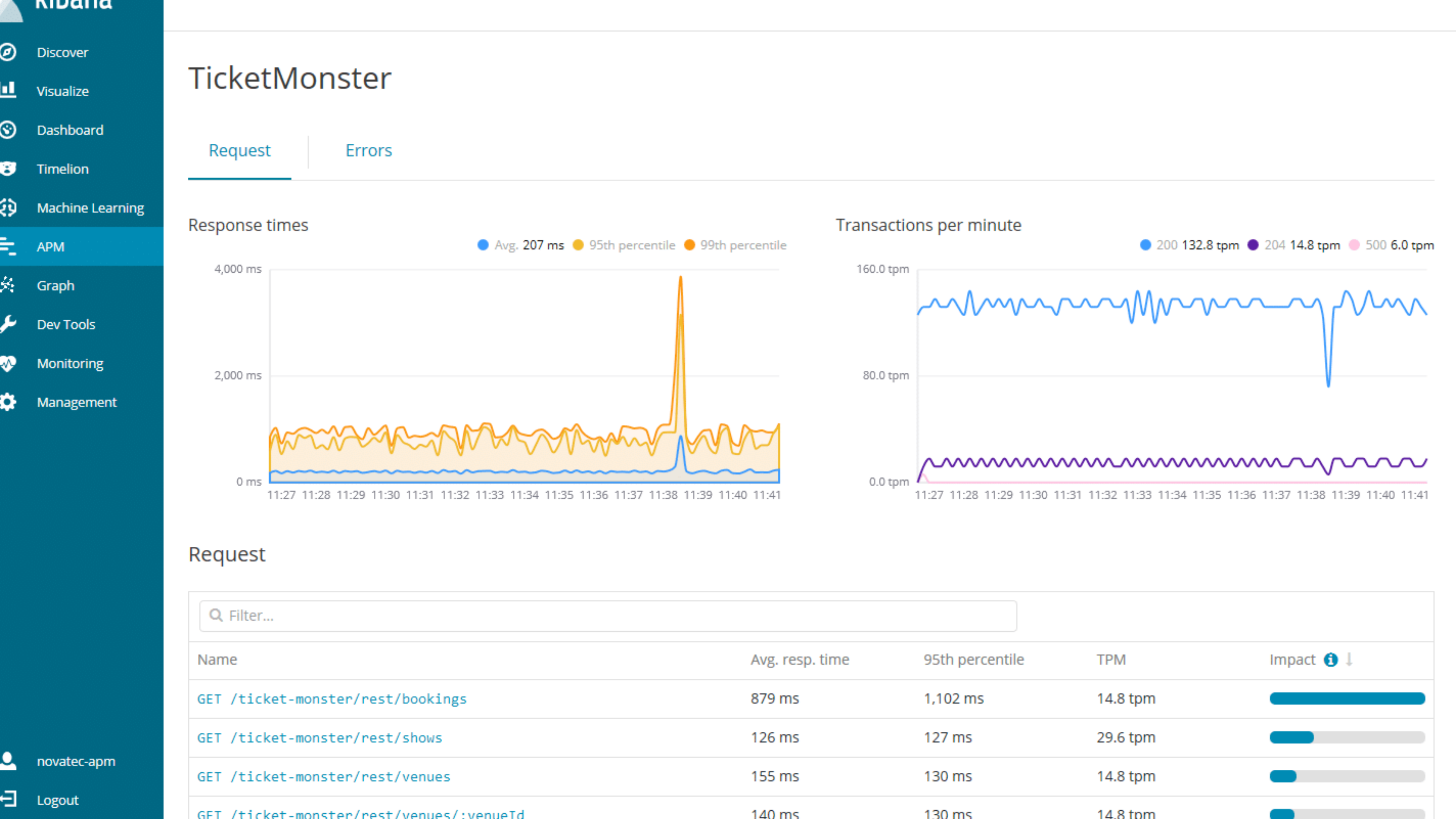




```
Speaker ShaiAlmog = Speaker.builder()
  .withRoles(
      createDeveloperAdvocateAt("Lightrun"),
      createCoFounderAt("Codename One"))
  .withProfessionalExperience(30, TimeUnit.YEARS)
  .withTopCompanies("Sun", "Oracle", "Codename One", "Lightrun")
  .withTwitter("twitter.com/debugagent")
  .withBlog("talktotheduck.dev")
  .withEmail("shaia@lightrun.com")
  .withGitHub("github.com/shai-almog")
  .build();
```

I APMs

Lightrun



TicketMonster

Request | Errors

Response times

Transactions per minute

Request

Filter...

Name	Avg. resp. time	95th percentile	TPM	Impact ℹ ↓
GET /ticket-monster/rest/bookings	879 ms	1,102 ms	14.8 tpm	
GET /ticket-monster/rest/shows	126 ms	127 ms	29.6 tpm	
GET /ticket-monster/rest/venues	155 ms	130 ms	14.8 tpm	
GET /ticket-monster/rest/venues/:venueId	140 ms	130 ms	14.8 tpm	

Lightrun

Developers

Lightrun

Developers

DevOps

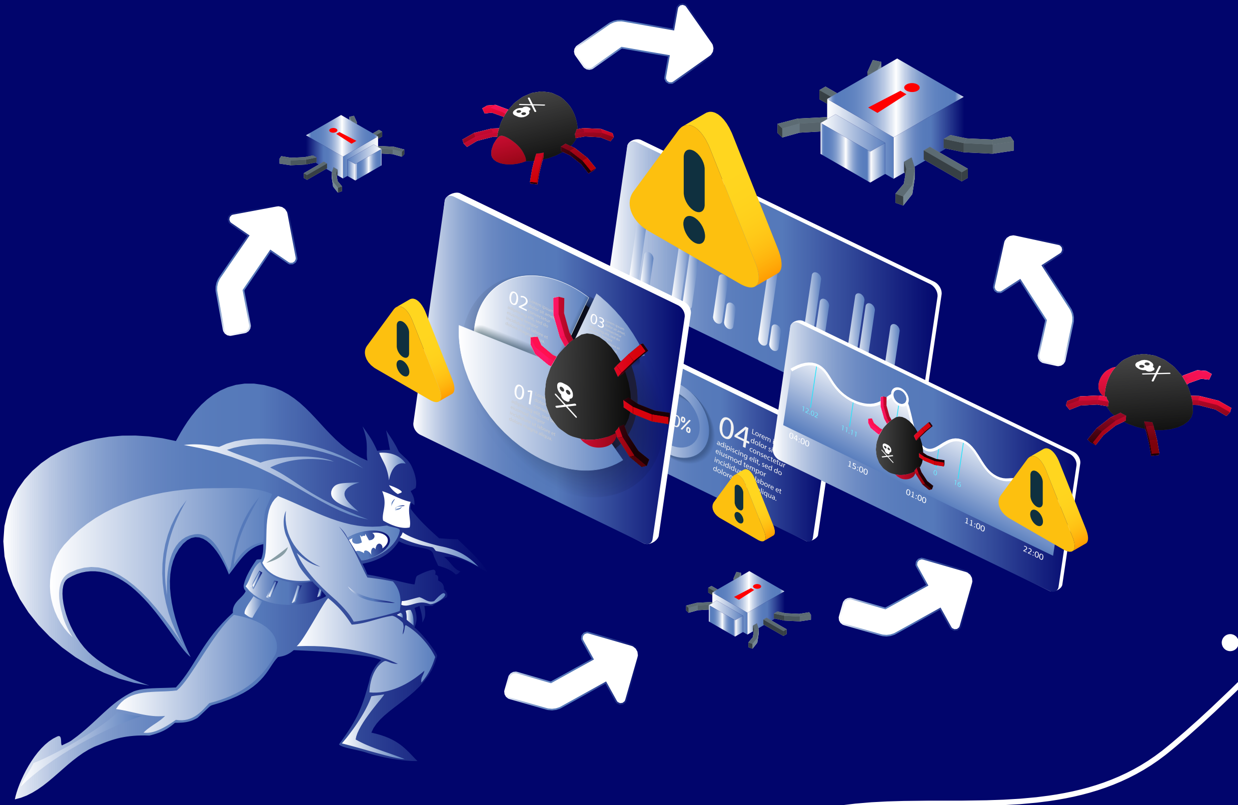
Lightrun

Developers

DevOps

Lightrun

Lightrun



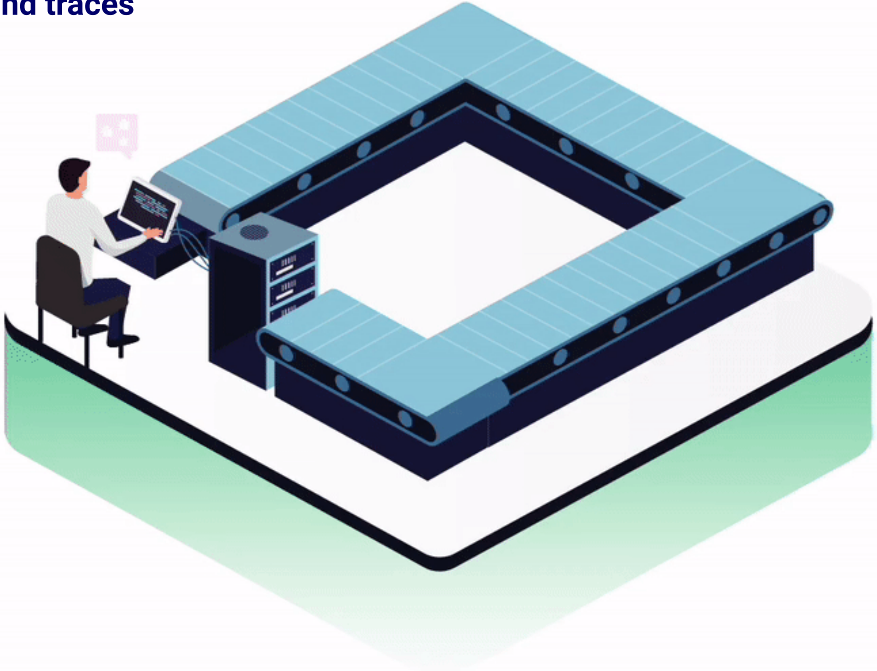
Lightrun

The Needle isn’t Necessarily Here

Production issues in 2021

1

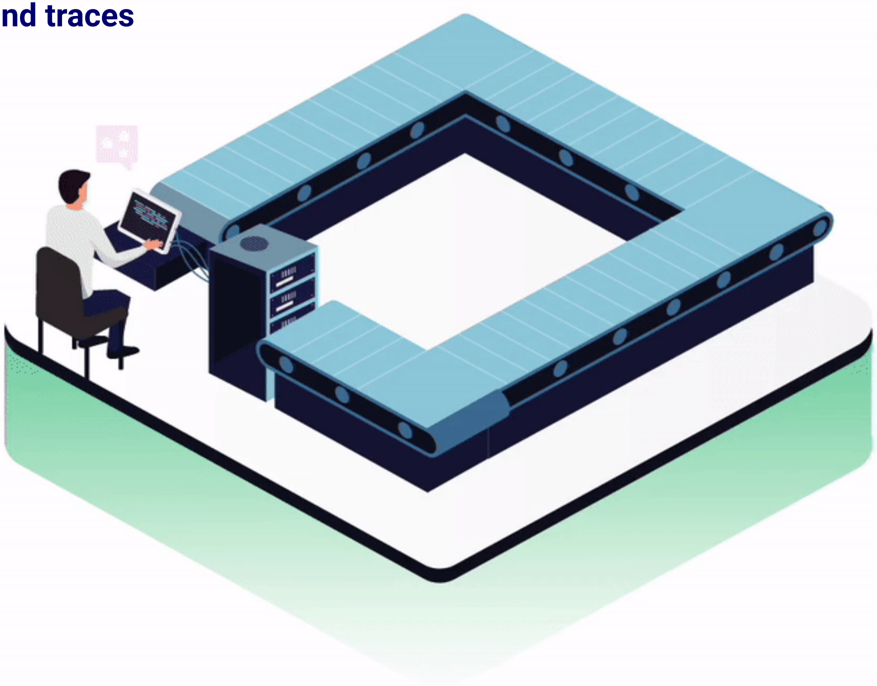
Add new logs, metrics and traces



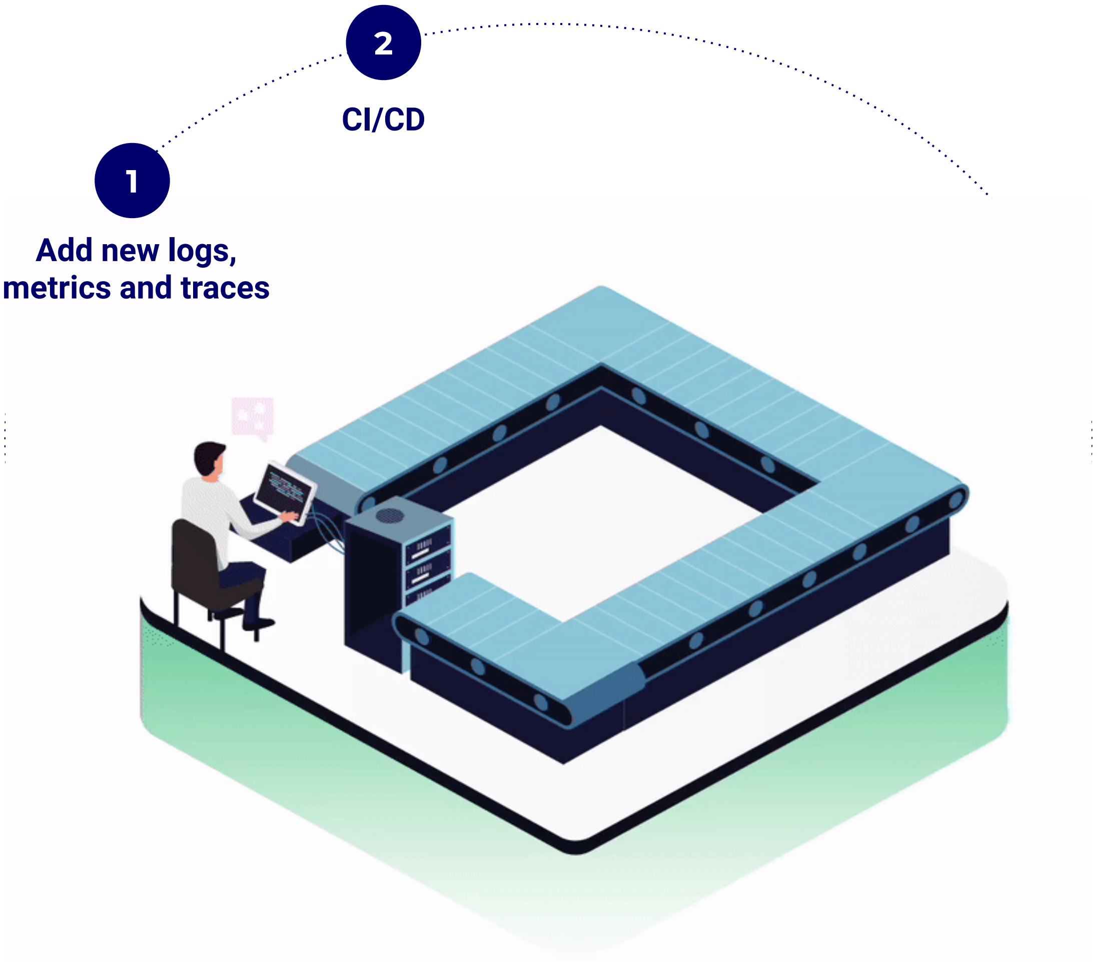
Production issues in 2021

1

Add new logs, metrics and traces



Production issues in 2021



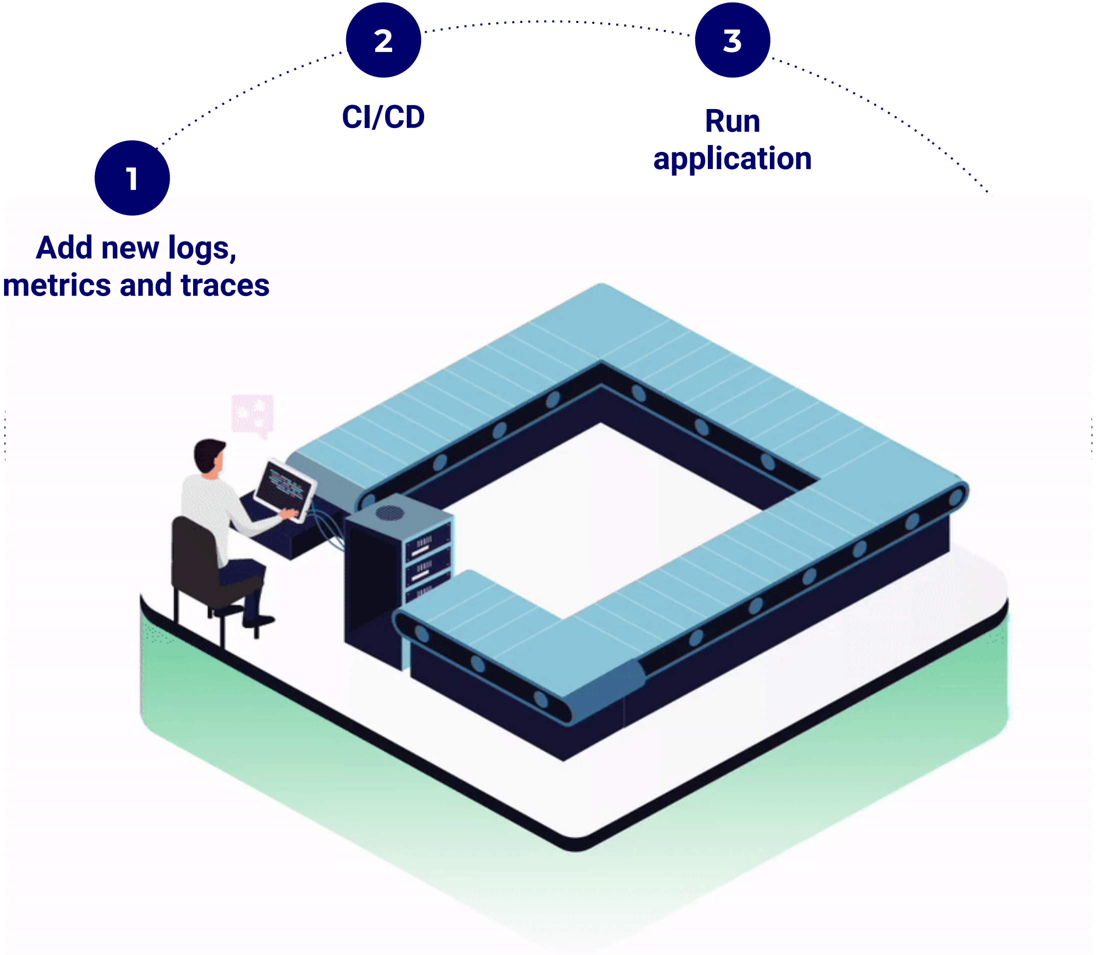
1

Add new logs, metrics and traces

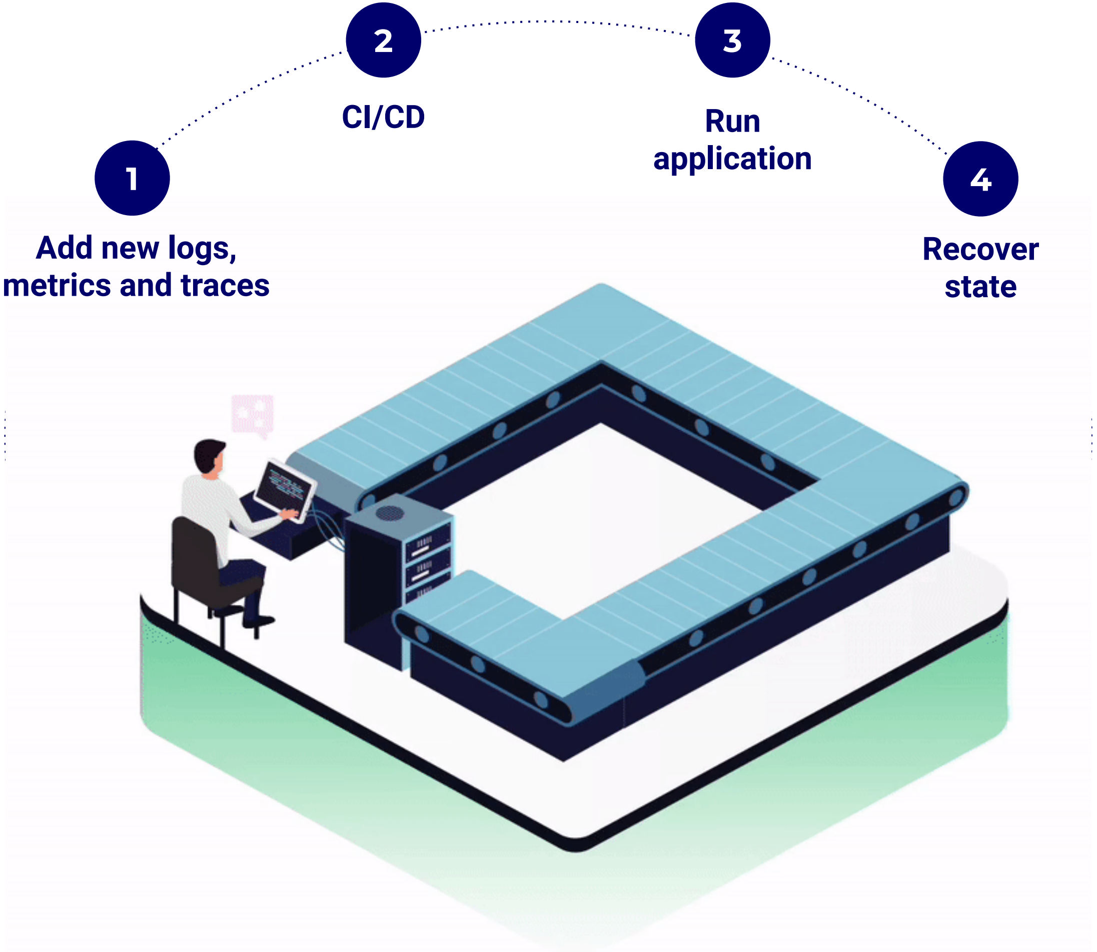
2

CI/CD

Production issues in 2021



Production issues in 2021



1 Add new logs, metrics and traces

2 CI/CD

3 Run application

4 Recover state

There has to be a Better Way!

Lightrun

We don’t know what we’ll run into

If Only we Could Debug...

Lightrun

But Debuggers aren’t the Right Tool

They Can’t Cross Server Boundaries

Lightrun

They can’t handle Different Languages

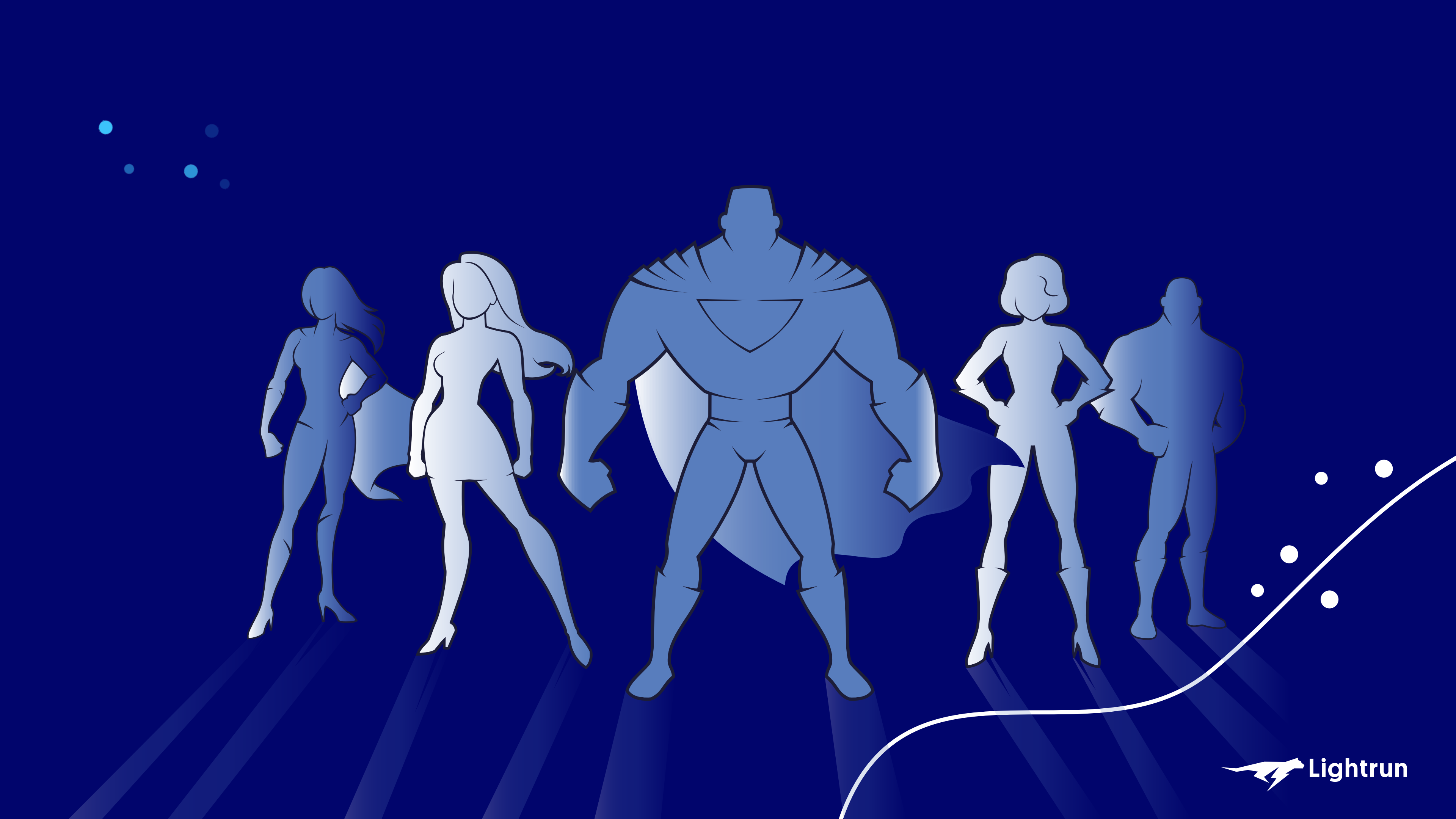
Lightrun

I APMs

Why Do APMs Impact Performance?

Lightrun

This isn’t Too Bad

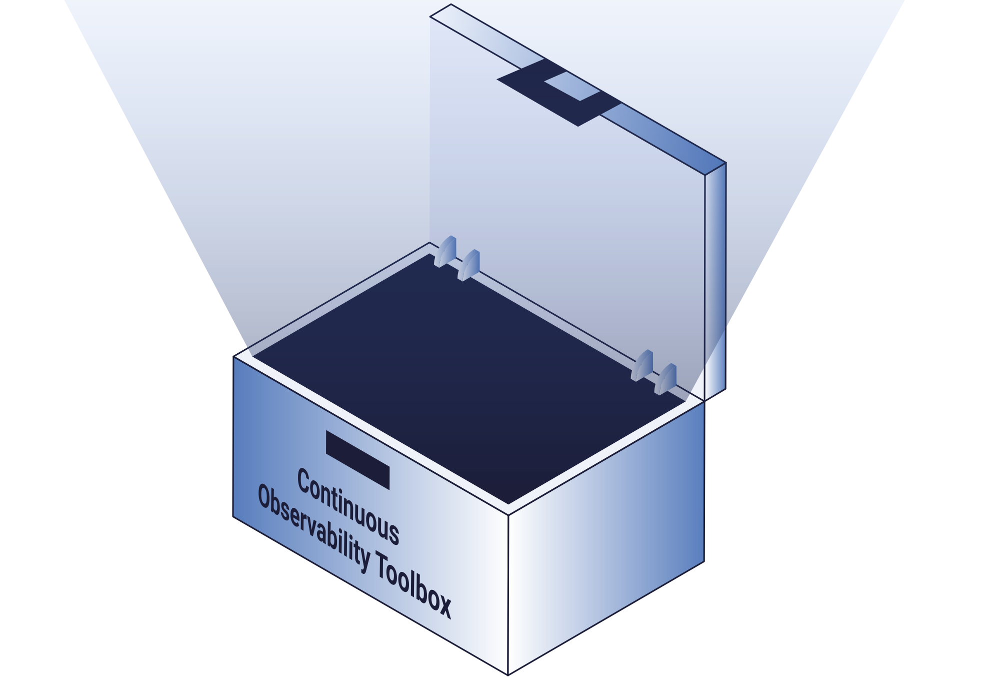

Lightrun

Developers

DevOps

Production Debugging

Lightrun



Continuous Observability?

Continuous Observability?

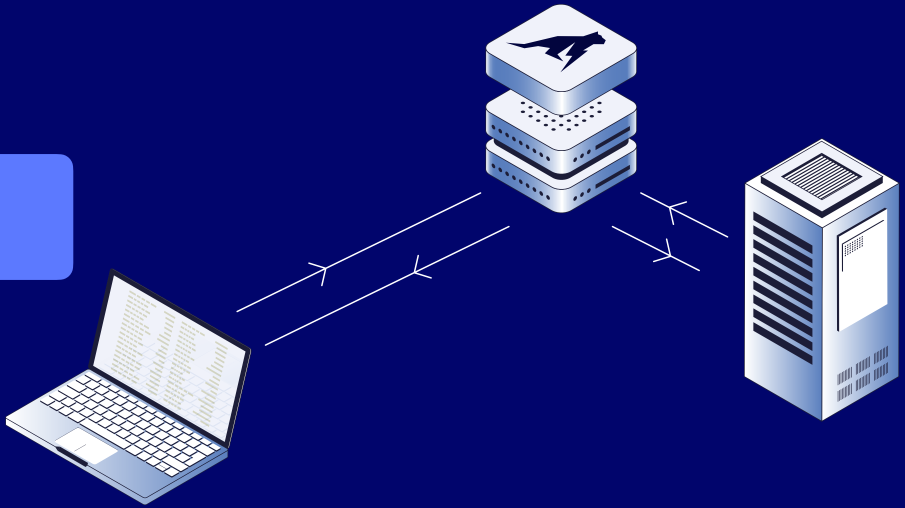
Continuous Observability?

Continuous Observability is a streamlined process of constantly asking new questions and getting immediate answers.

Lightrun

How it Works

Developer

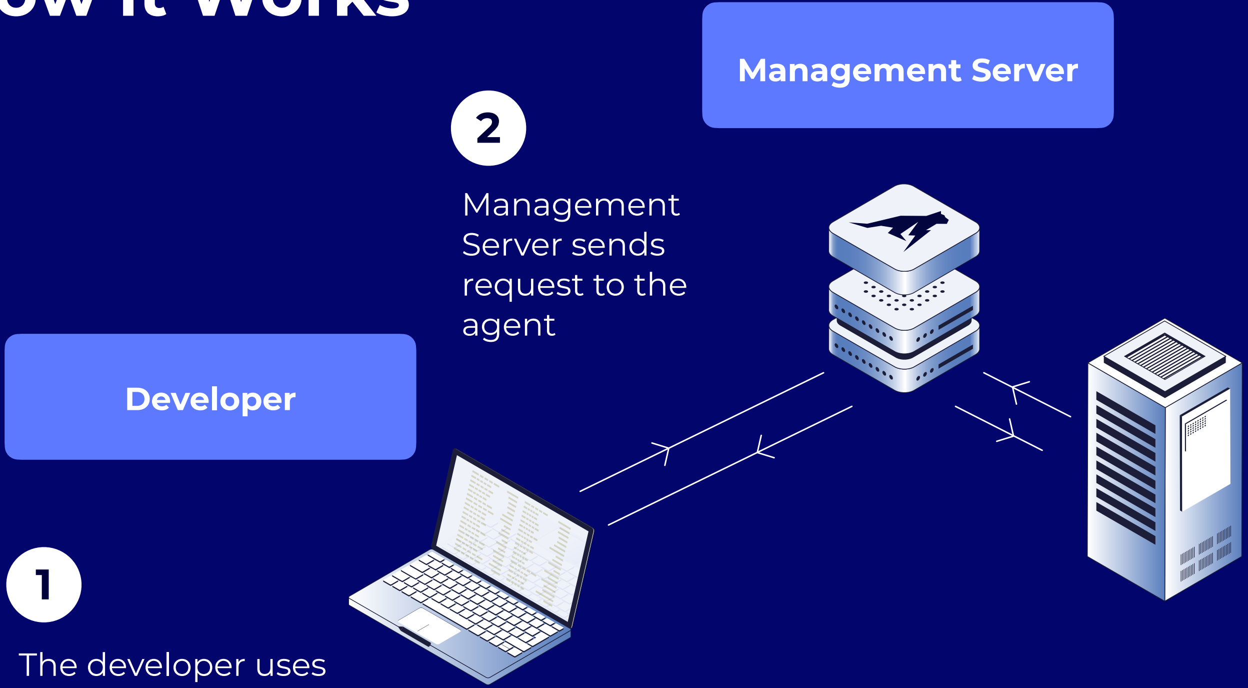


1

The developer uses Lightrun's IDE plugin to add an action in example.java line 100

Lightrun

How it Works



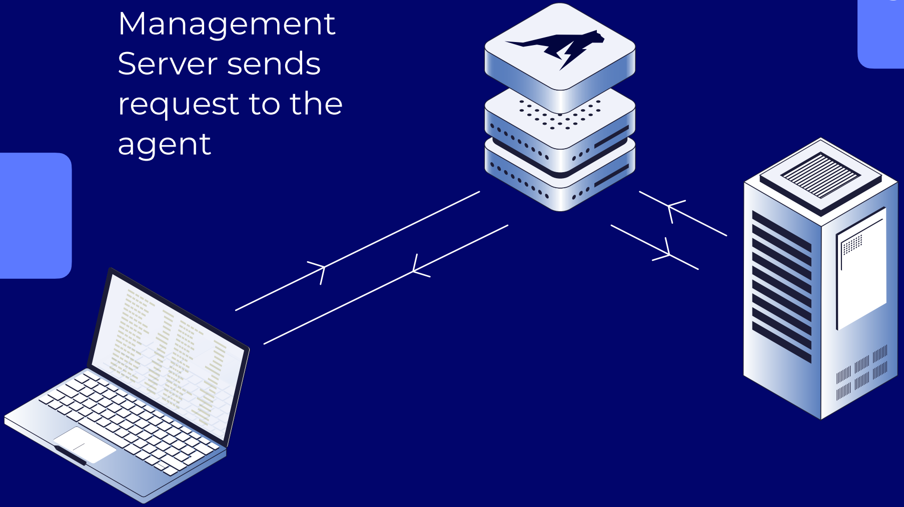
2

Management Server sends request to the agent

1

The developer uses Lightrun's IDE plugin to add an action in example.java line 100

How it Works



Developer

1

The developer uses Lightrun's IDE plugin to add an action in example.java line 100

2

Management Server sends request to the agent

Management Server

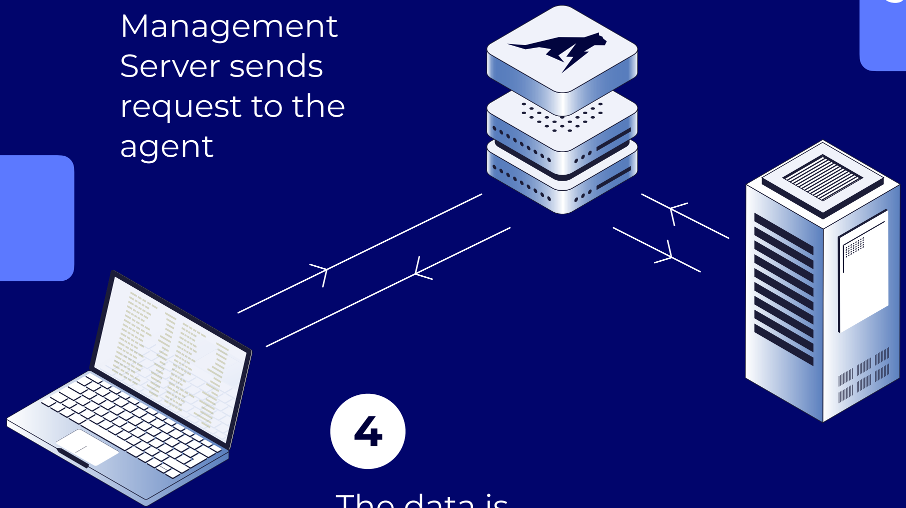
Service Running with Lightrun's Agent

3

Agent inserts the actions at the specific location at runtime

Lightrun

How it Works



Management Server

Service Running with Lightrun's Agent

Developer

1

The developer uses Lightrun's IDE plugin to add an action in example.java line 100

2

Management Server sends request to the agent

3

Agent inserts the actions at the specific location at runtime

4

The data is transferred to the developer's IDE, through the Server

Lightrun

It’s Very Low Overhead?

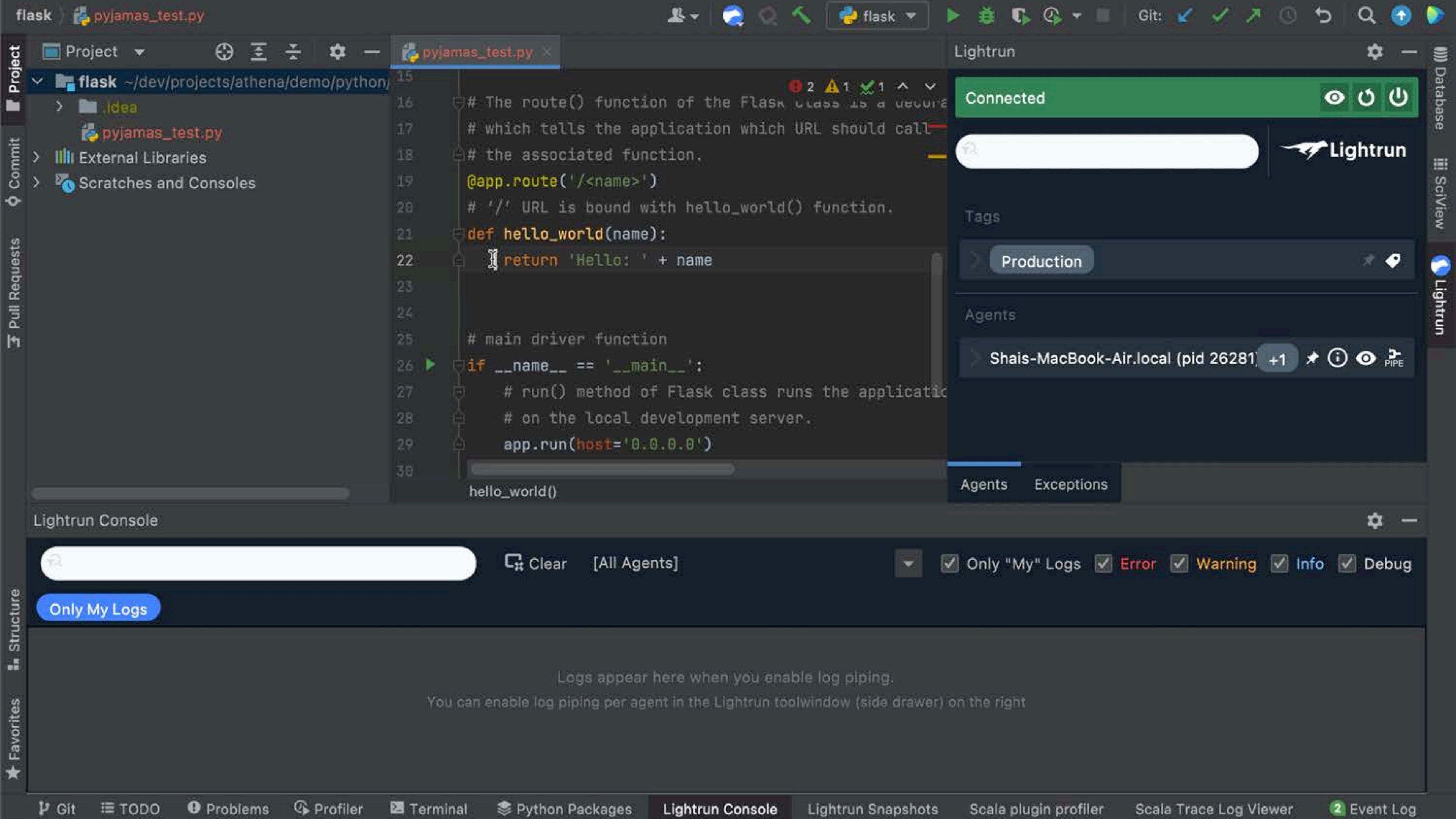
It isn’t an APM

What if I Observe Something Central?

Lightrun

Lightrun

Demo



```
# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def hello_world(name):
    return 'Hello: ' + name


# main driver function
if __name__ == '__main__':
    # run() method of Flask class runs the applicatio
    # on the local development server.
    app.run(host='0.0.0.0')
```

Connected

Tags

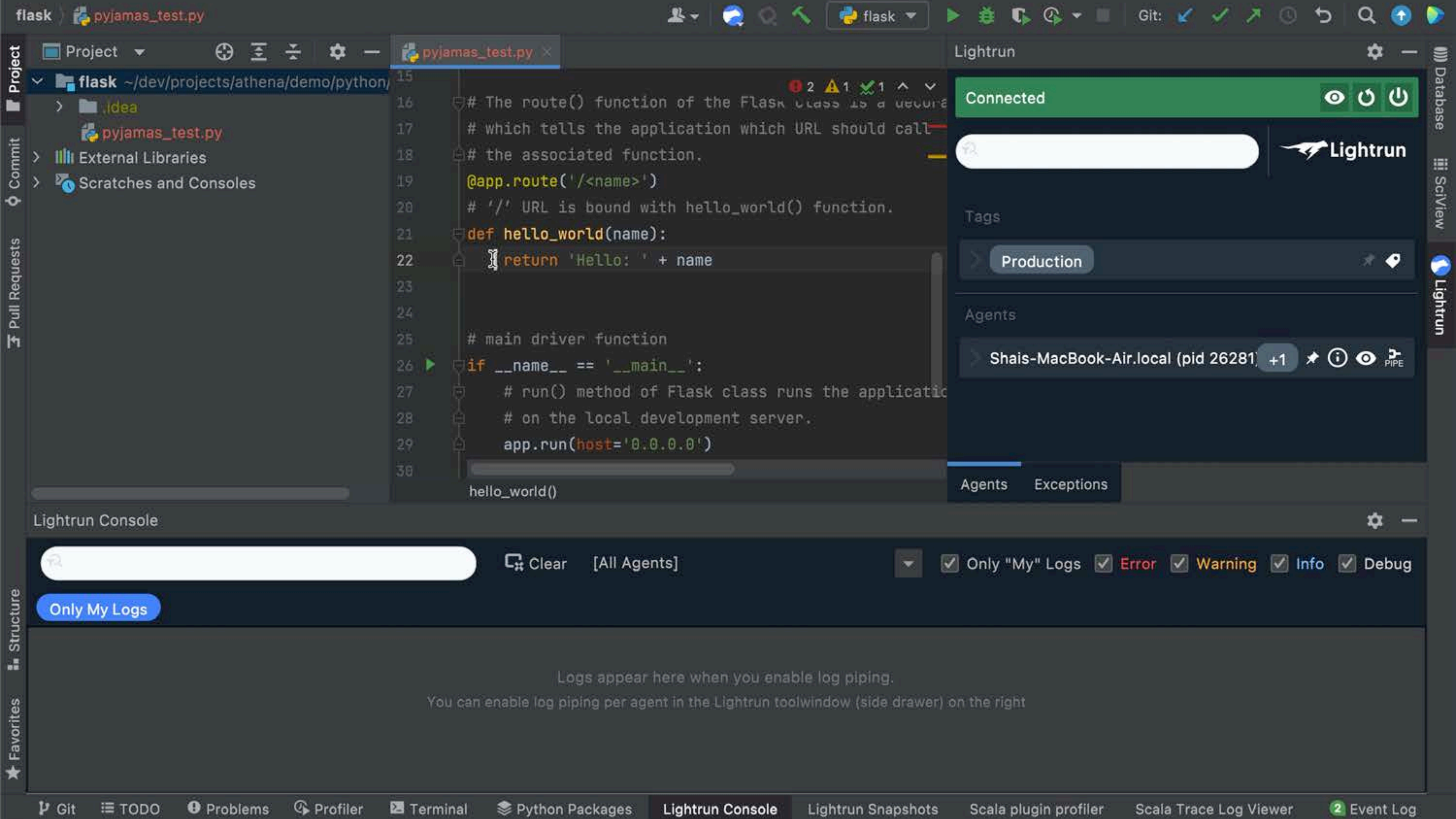
Production

Agents

Shais-MacBook-Air.local (pid 26281) +1

☑ Only "My" Logs ☑ Error ☑ Warning ☑ Info ☑ Debug

Logs appear here when you enable log piping.
You can enable log piping per agent in the Lightrun toolwindow (side drawer) on the right



```
# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def hello_world(name):
    return 'Hello: ' + name


# main driver function
if __name__ == '__main__':
    # run() method of Flask class runs the applicatio
    # on the local development server.
    app.run(host='0.0.0.0')
```

Lightrun

Connected

Lightrun

Tags

Production

Agents

Shais-MacBook-Air.local (pid 26281) +1

Agents Exceptions

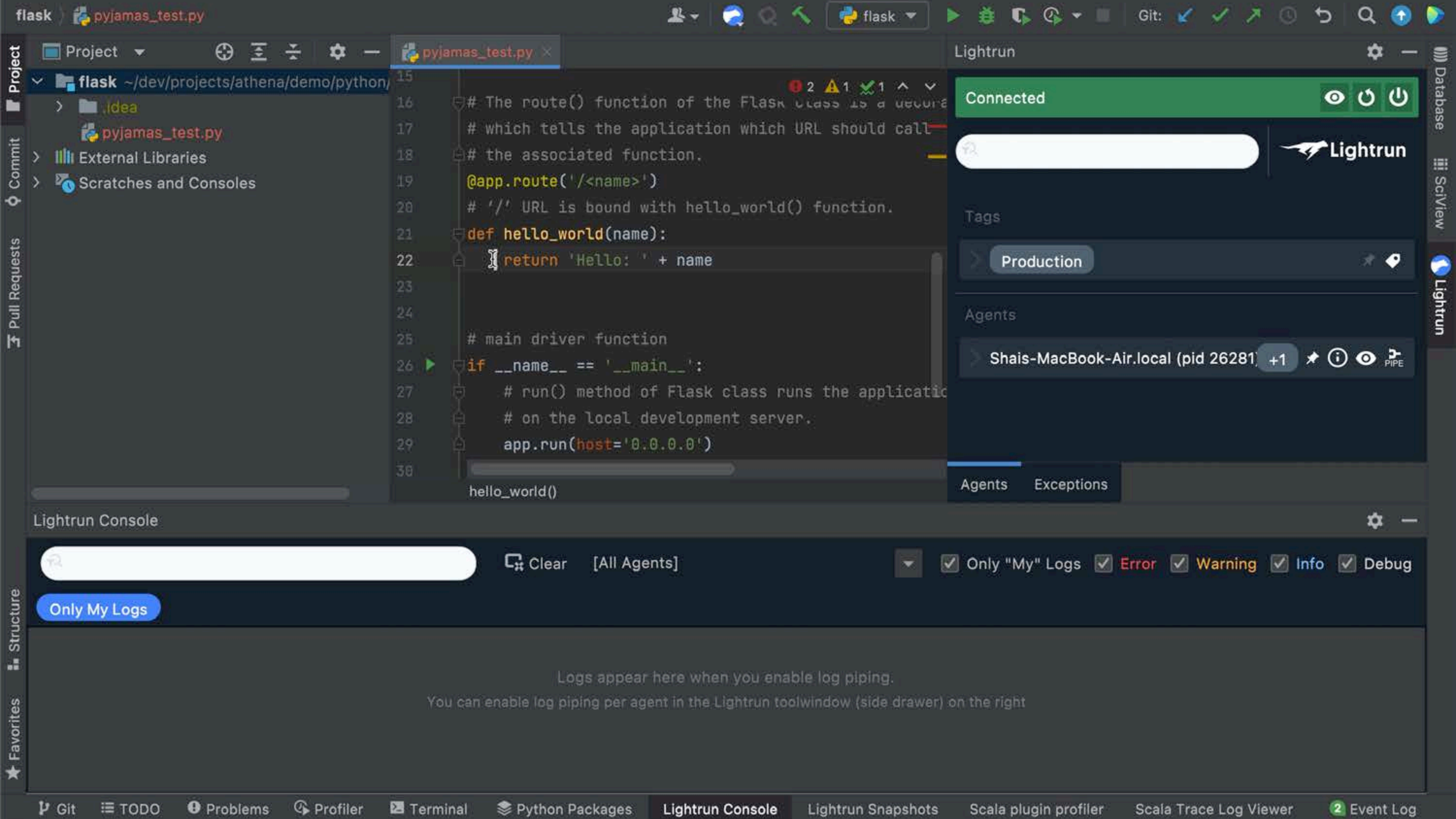
Lightrun Console

Clear [All Agents] ☑ Only "My" Logs ☑ Error ☑ Warning ☑ Info ☑ Debug

Only My Logs

Logs appear here when you enable log piping.

You can enable log piping per agent in the Lightrun toolwindow (side drawer) on the right



```
# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def hello_world(name):
    return 'Hello: ' + name


# main driver function
if __name__ == '__main__':
    # run() method of Flask class runs the applicatio
    # on the local development server.
    app.run(host='0.0.0.0')
```

Connected

Tags

Production

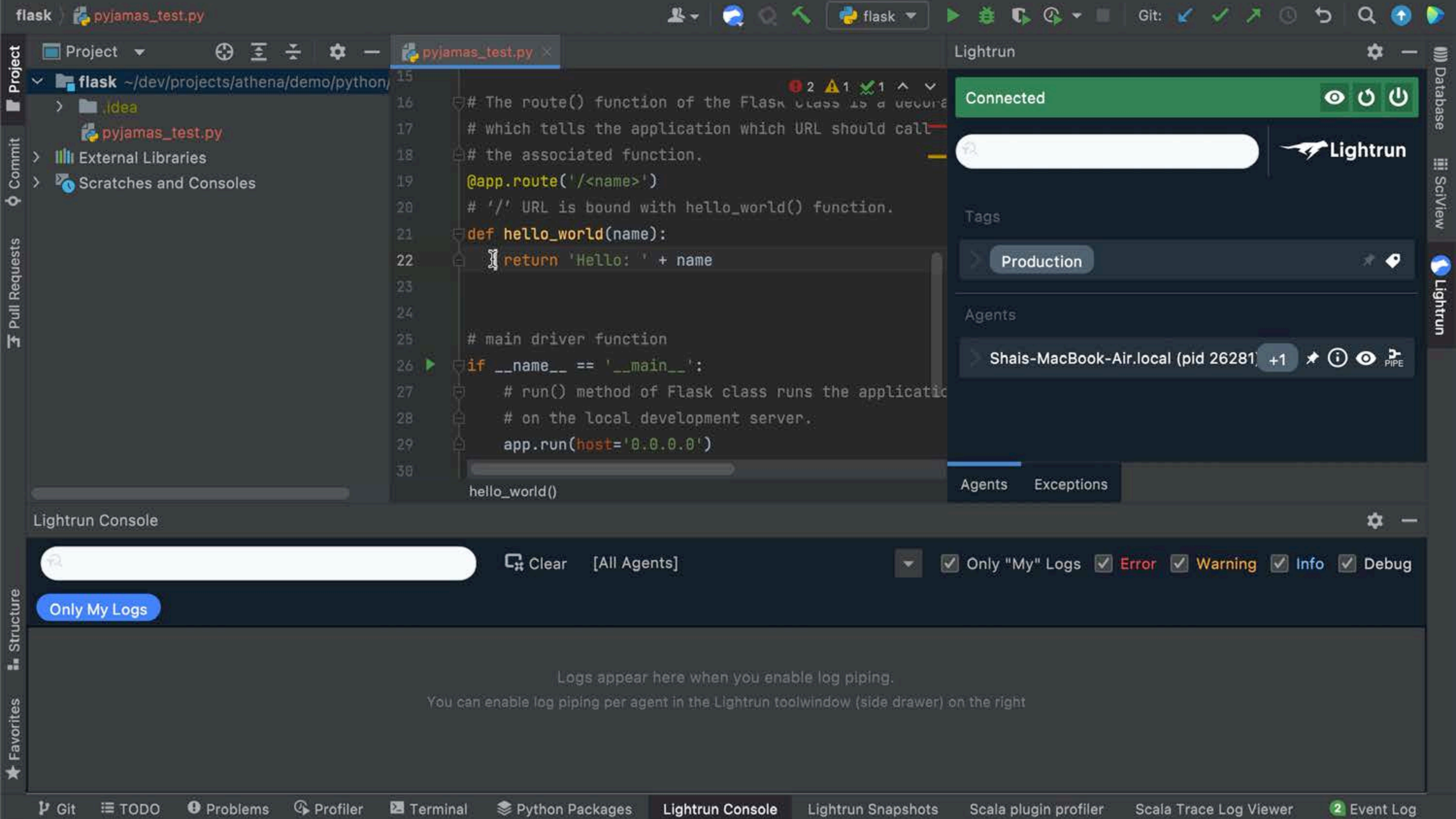
Agents

Shais-MacBook-Air.local (pid 26281) +1

☑ Only "My" Logs ☑ Error ☑ Warning ☑ Info ☑ Debug

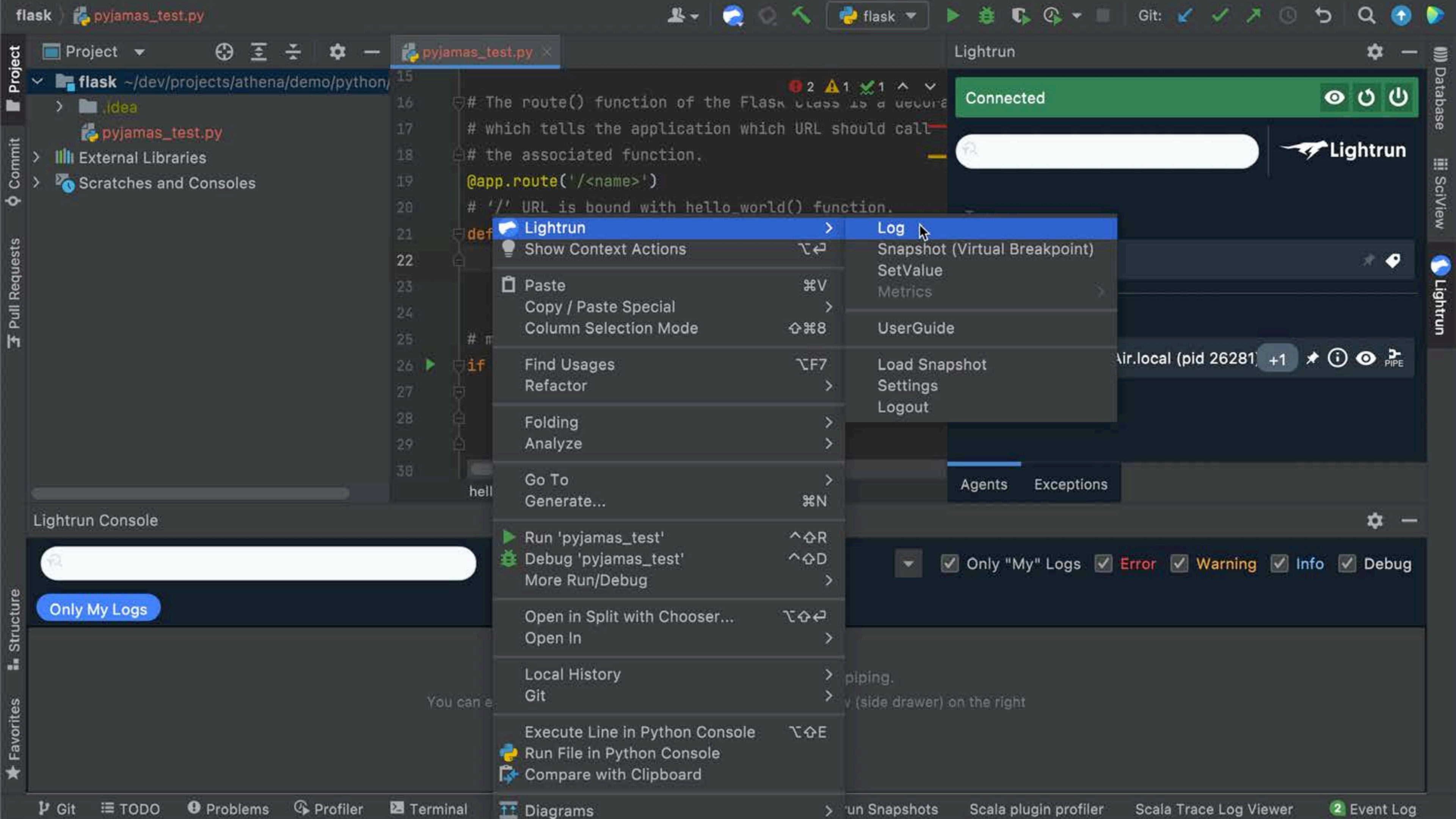
Only My Logs

Logs appear here when you enable log piping.
You can enable log piping per agent in the Lightrun toolwindow (side drawer) on the right

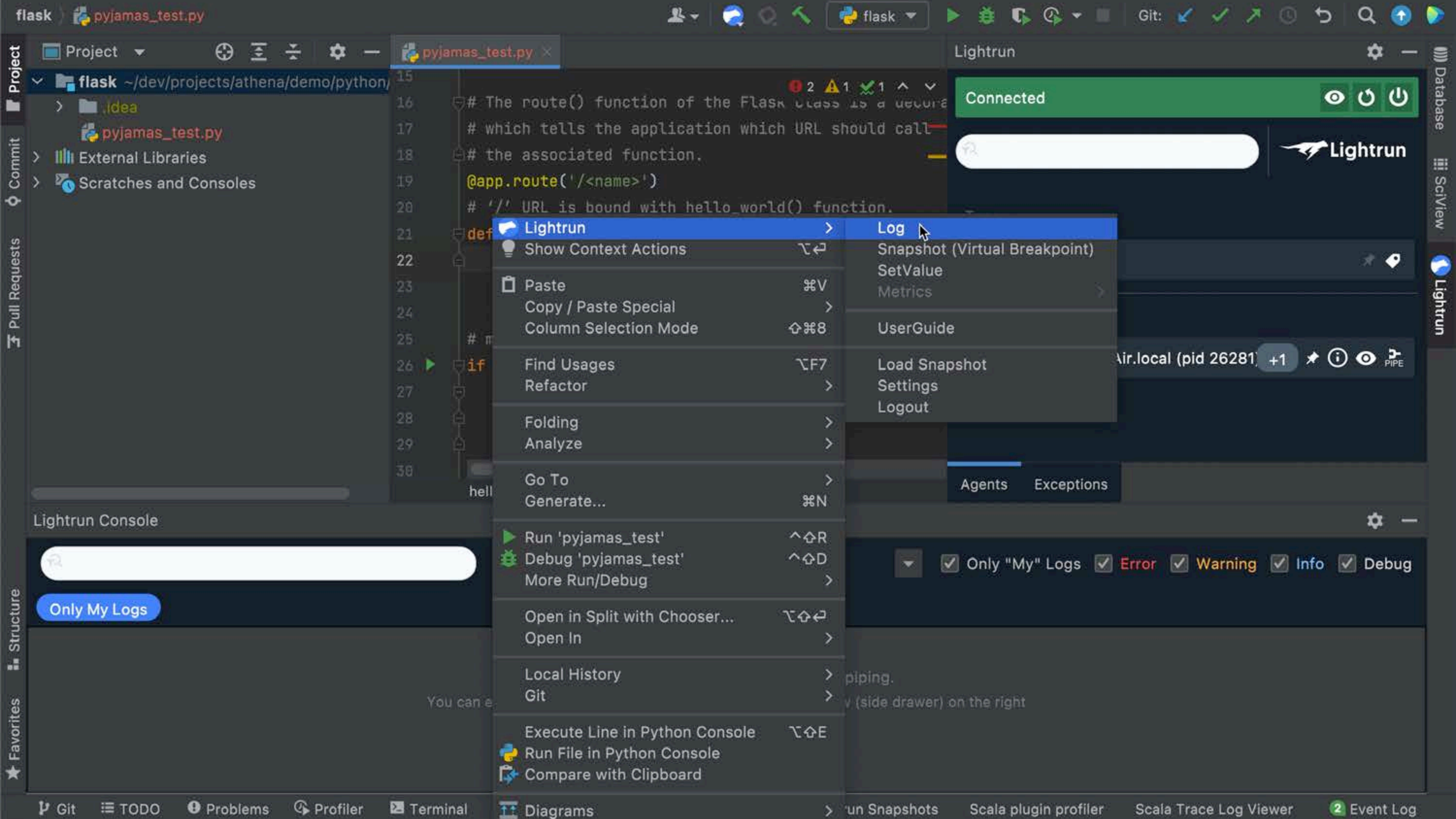


```
# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def hello_world(name):
    return 'Hello: ' + name


# main driver function
if __name__ == '__main__':
    # run() method of Flask class runs the applicatio
    # on the local development server.
    app.run(host='0.0.0.0')
```

Logs appear here when you enable log piping.
You can enable log piping per agent in the Lightrun toolwindow (side drawer) on the right



```
# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def


# m
if
```



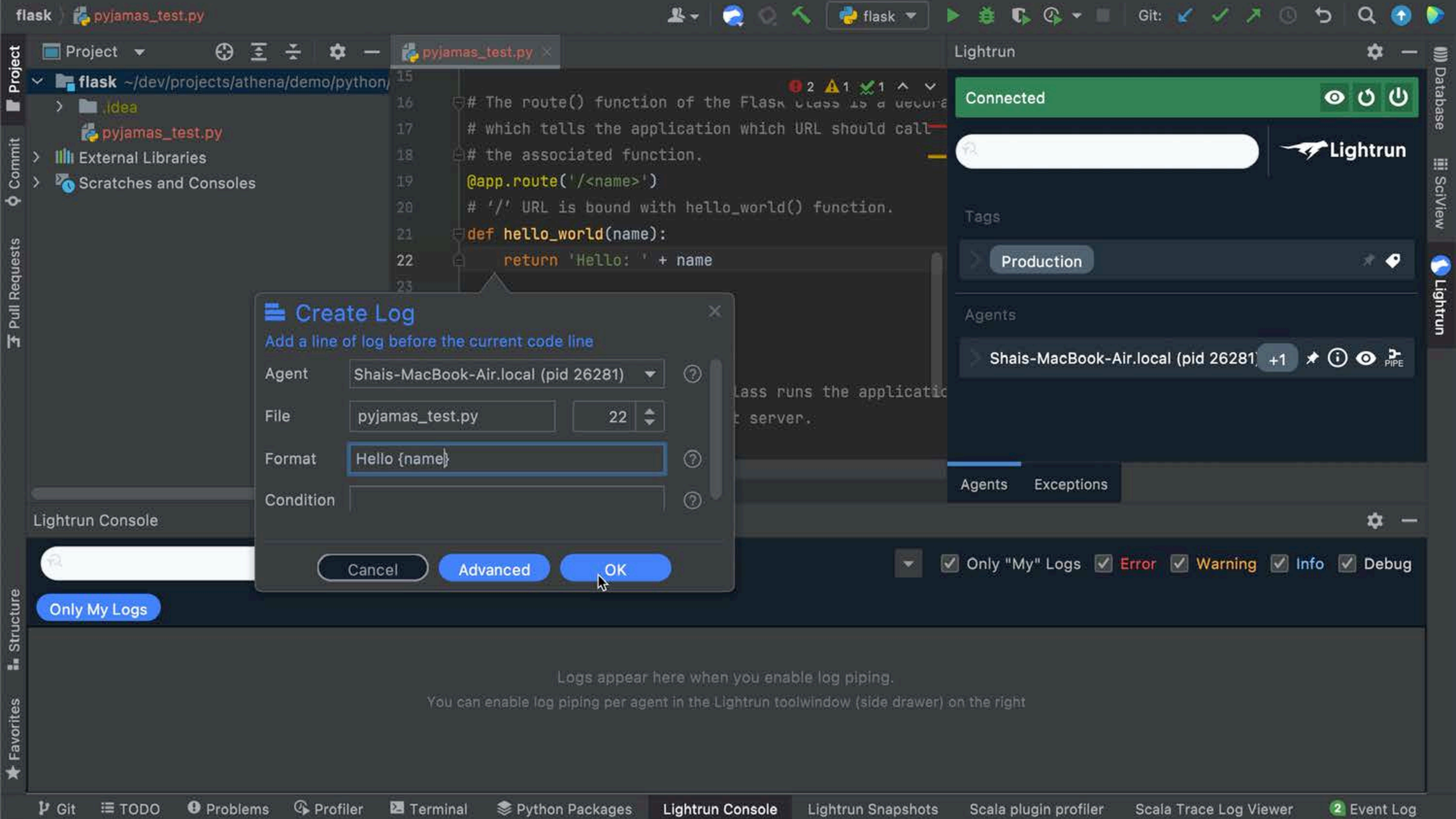
```
# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def
```

```
# m
if
```

hell

You can e… piping.

…(side drawer) on the right



```
# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def hello_world(name):
    return 'Hello: ' + name
```

```
lass runs the applicati
t server.
```

Create Log

Add a line of log before the current code line

Agent: Shais-MacBook-Air.local (pid 26281)

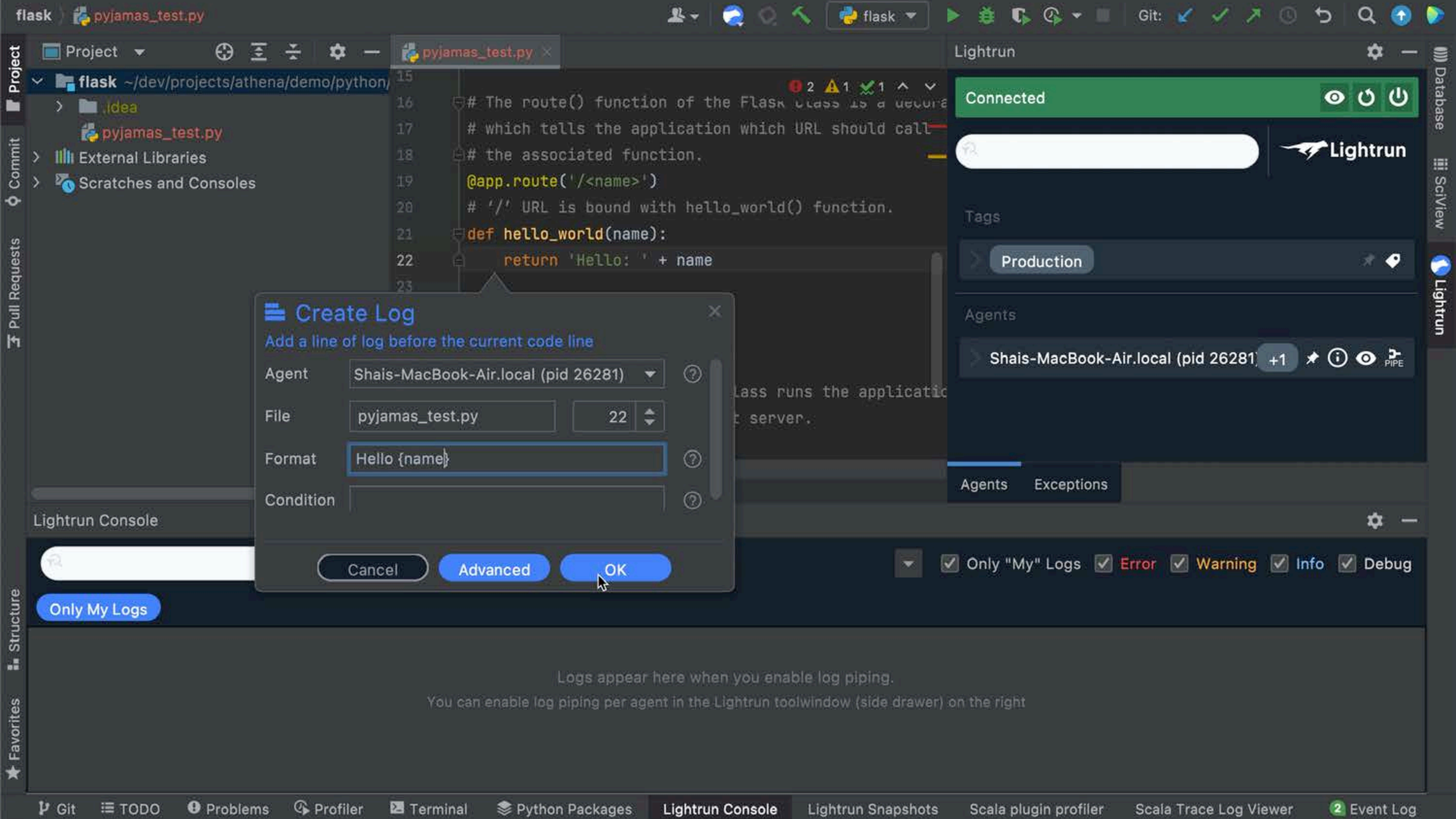
File: pyjamas_test.py 22

Format: Hello {name}

Condition:

Cancel | Advanced | OK

Logs appear here when you enable log piping.
You can enable log piping per agent in the Lightrun toolwindow (side drawer) on the right



```
# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def hello_world(name):
    return 'Hello: ' + name

                                lass runs the applicatio
                                t server.
```

Create Log

Add a line of log before the current code line

Agent: Shais-MacBook-Air.local (pid 26281)

File: pyjamas_test.py 22

Format: Hello {name}

Condition:

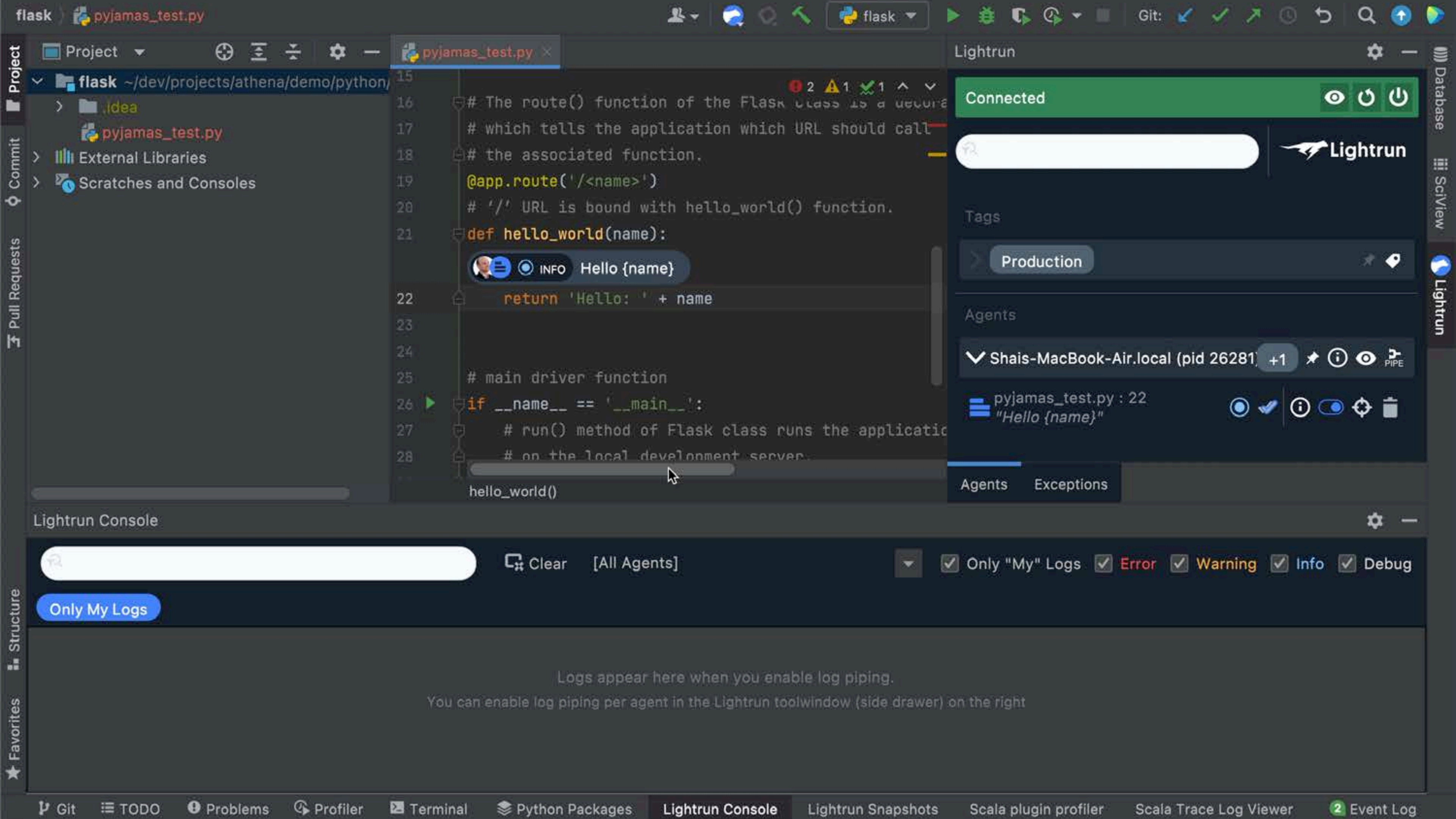
Connected

Tags: Production

Agents: Shais-MacBook-Air.local (pid 26281) +1

☑ Only "My" Logs ☑ Error ☑ Warning ☑ Info ☑ Debug

Logs appear here when you enable log piping.
You can enable log piping per agent in the Lightrun toolwindow (side drawer) on the right



```
# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def hello_world(name):
    INFO Hello {name}
    return 'Hello: ' + name


# main driver function
if __name__ == '__main__':
    # run() method of Flask class runs the applicatio
    # on the local development server.
```

hello_world()

Connected

Tags

Production

Agents

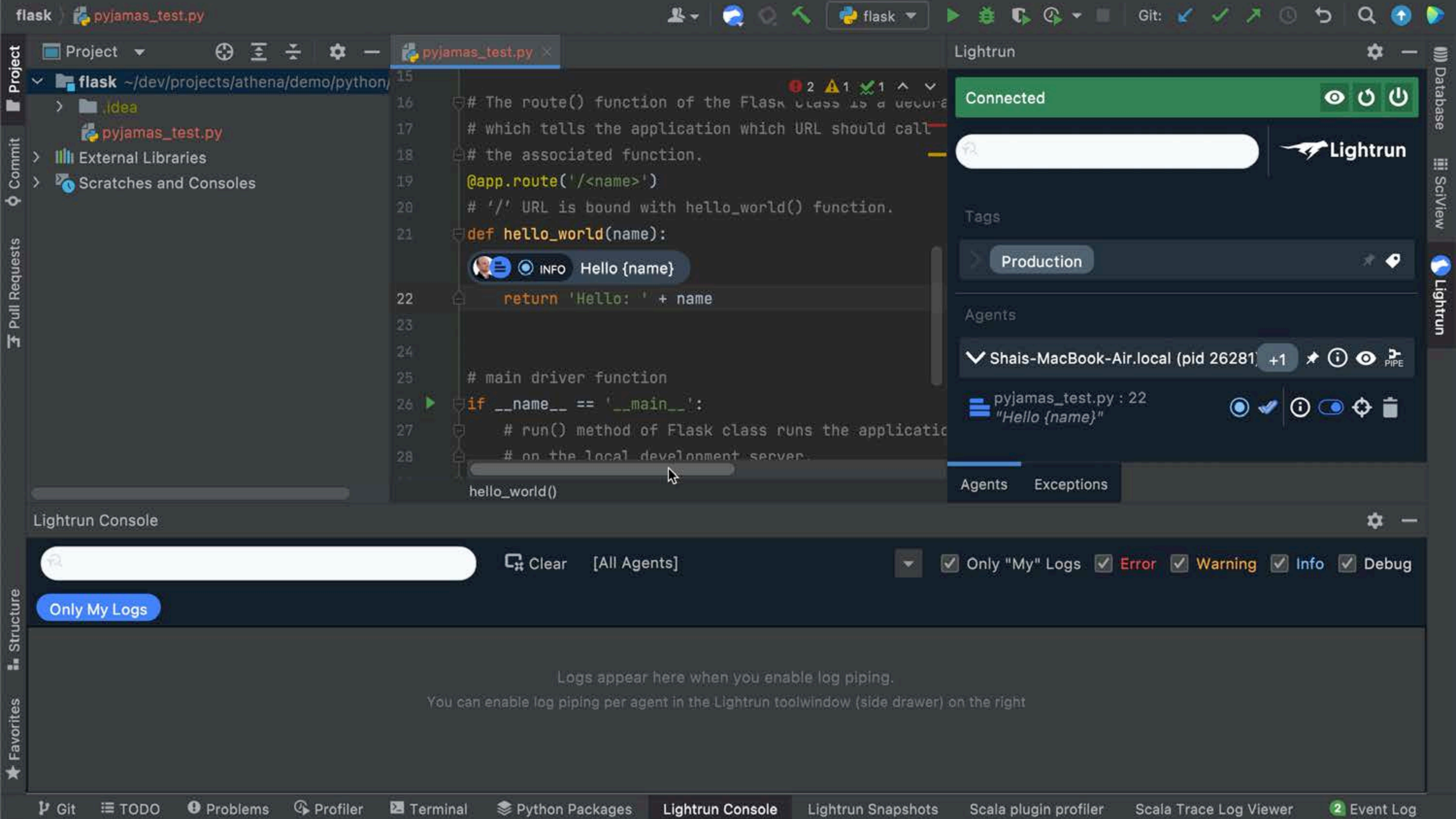
Shais-MacBook-Air.local (pid 26281) +1

pyjamas_test.py : 22
"Hello {name}"

☑ Only "My" Logs ☑ Error ☑ Warning ☑ Info ☑ Debug

Only My Logs

Logs appear here when you enable log piping.
You can enable log piping per agent in the Lightrun toolwindow (side drawer) on the right



```
# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def hello_world(name):
    INFO  Hello {name}
    return 'Hello: ' + name


# main driver function
if __name__ == '__main__':
    # run() method of Flask class runs the applicatio
    # on the local development server.
```

hello_world()

Connected

Tags

Production

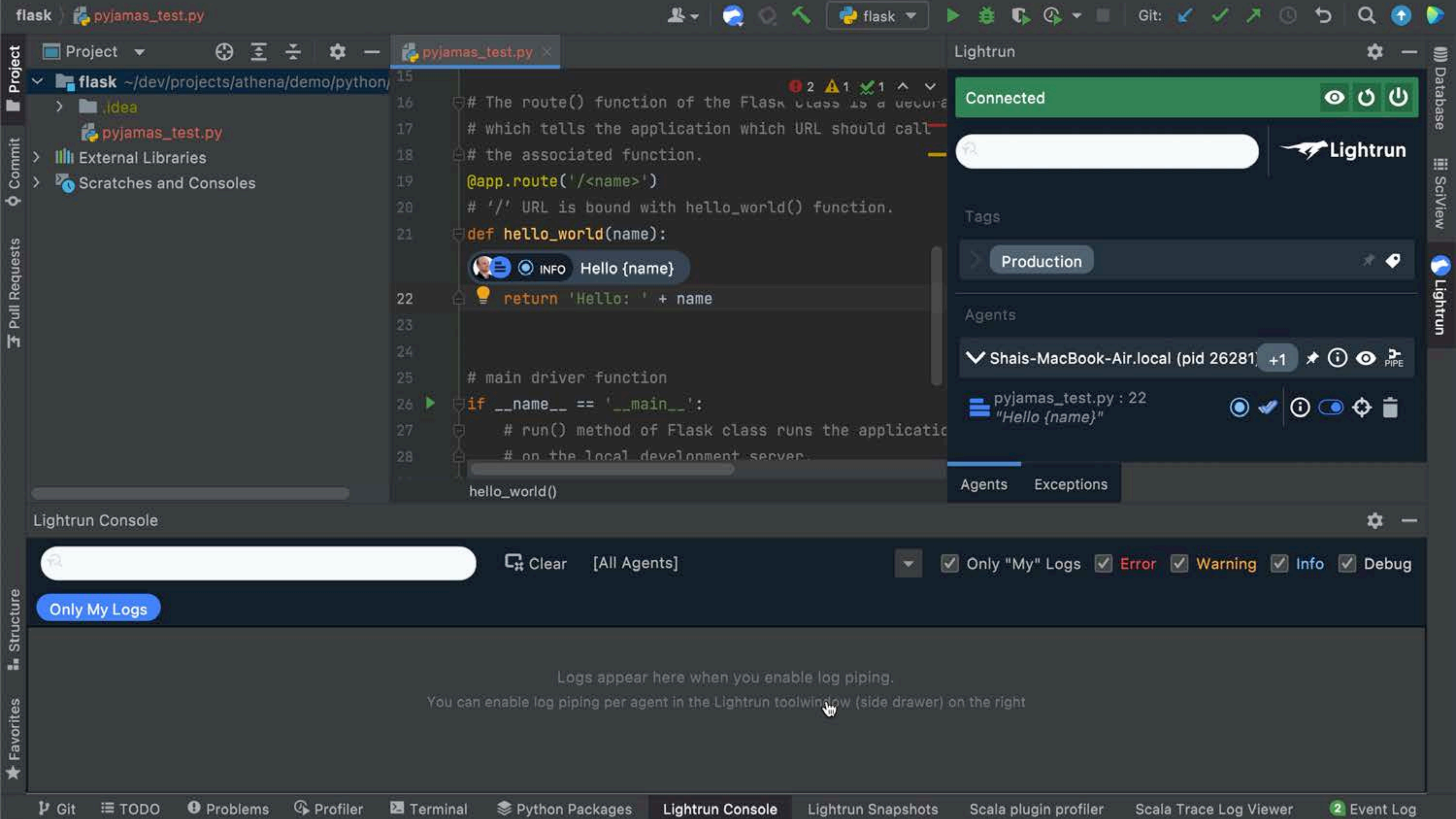
Agents

Shais-MacBook-Air.local (pid 26281) +1

pyjamas_test.py : 22
"Hello {name}"

☑ Only "My" Logs ☑ Error ☑ Warning ☑ Info ☑ Debug

Logs appear here when you enable log piping.
You can enable log piping per agent in the Lightrun toolwindow (side drawer) on the right



```
# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def hello_world(name):
    INFO Hello {name}
    return 'Hello: ' + name


# main driver function
if __name__ == '__main__':
    # run() method of Flask class runs the applicatio
    # on the local development server.
```

hello_world()

Connected

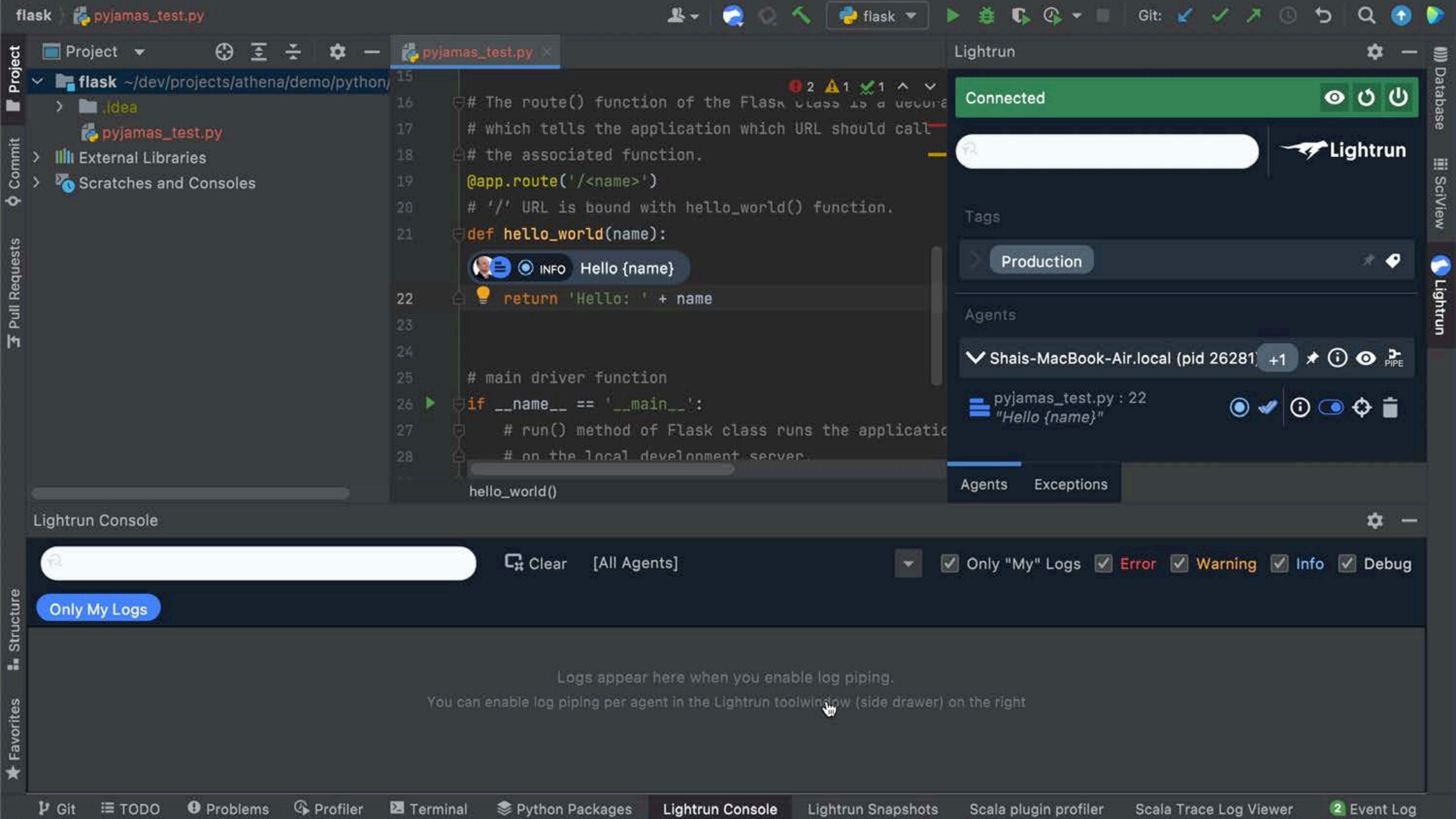
Production

Shais-MacBook-Air.local (pid 26281) +1

pyjamas_test.py : 22
"Hello {name}"

☑ Only "My" Logs ☑ Error ☑ Warning ☑ Info ☑ Debug

Logs appear here when you enable log piping.
You can enable log piping per agent in the Lightrun toolwindow (side drawer) on the right



```
# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def hello_world(name):
    return 'Hello: ' + name


# main driver function
if __name__ == '__main__':
    # run() method of Flask class runs the applicatic
    # on the local development server.
```

Connected

Production

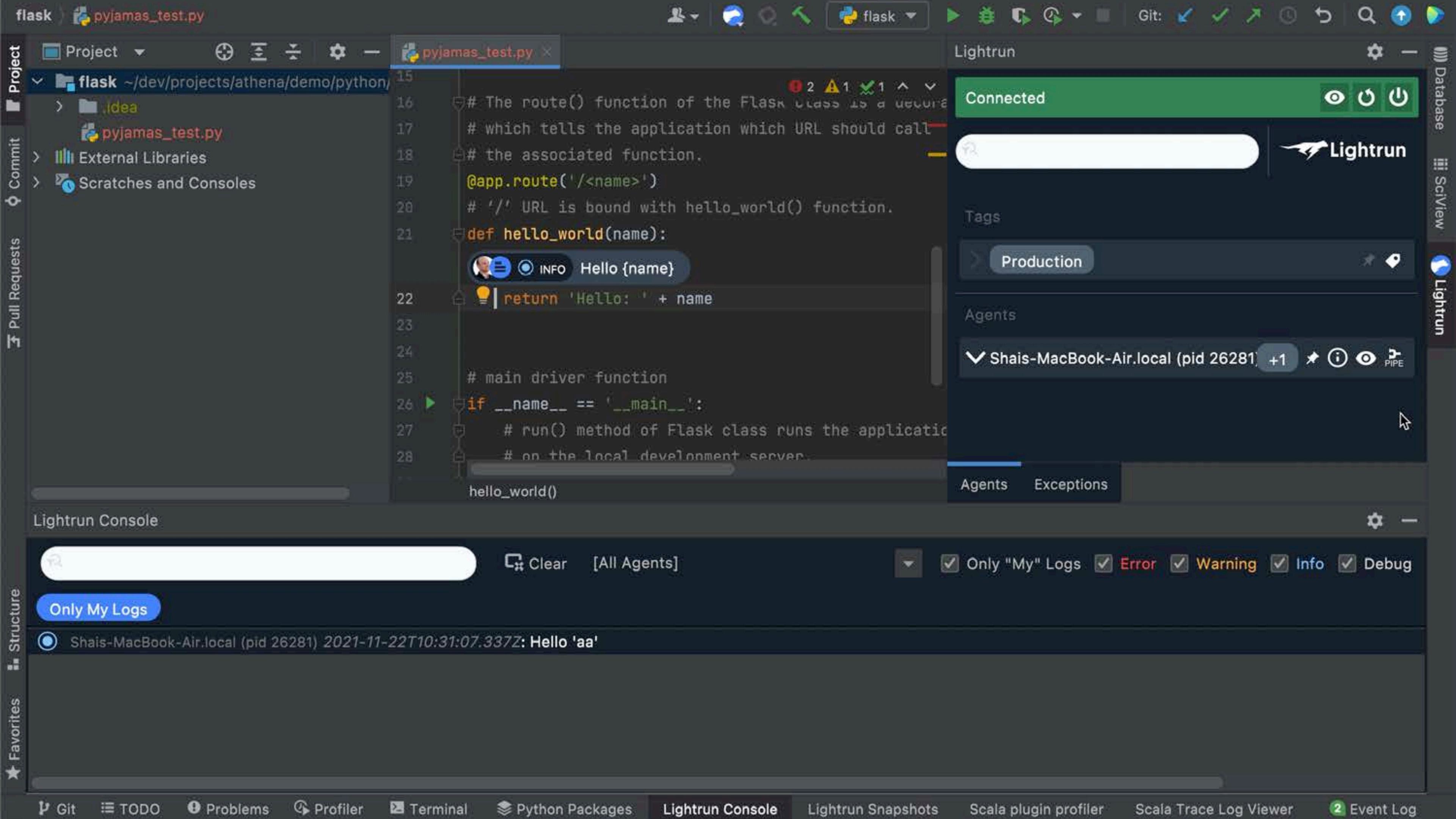
Shais-MacBook-Air.local (pid 26281) +1

pyjamas_test.py : 22
"Hello {name}"

☑ Only "My" Logs ☑ Error ☑ Warning ☑ Info ☑ Debug

Only My Logs

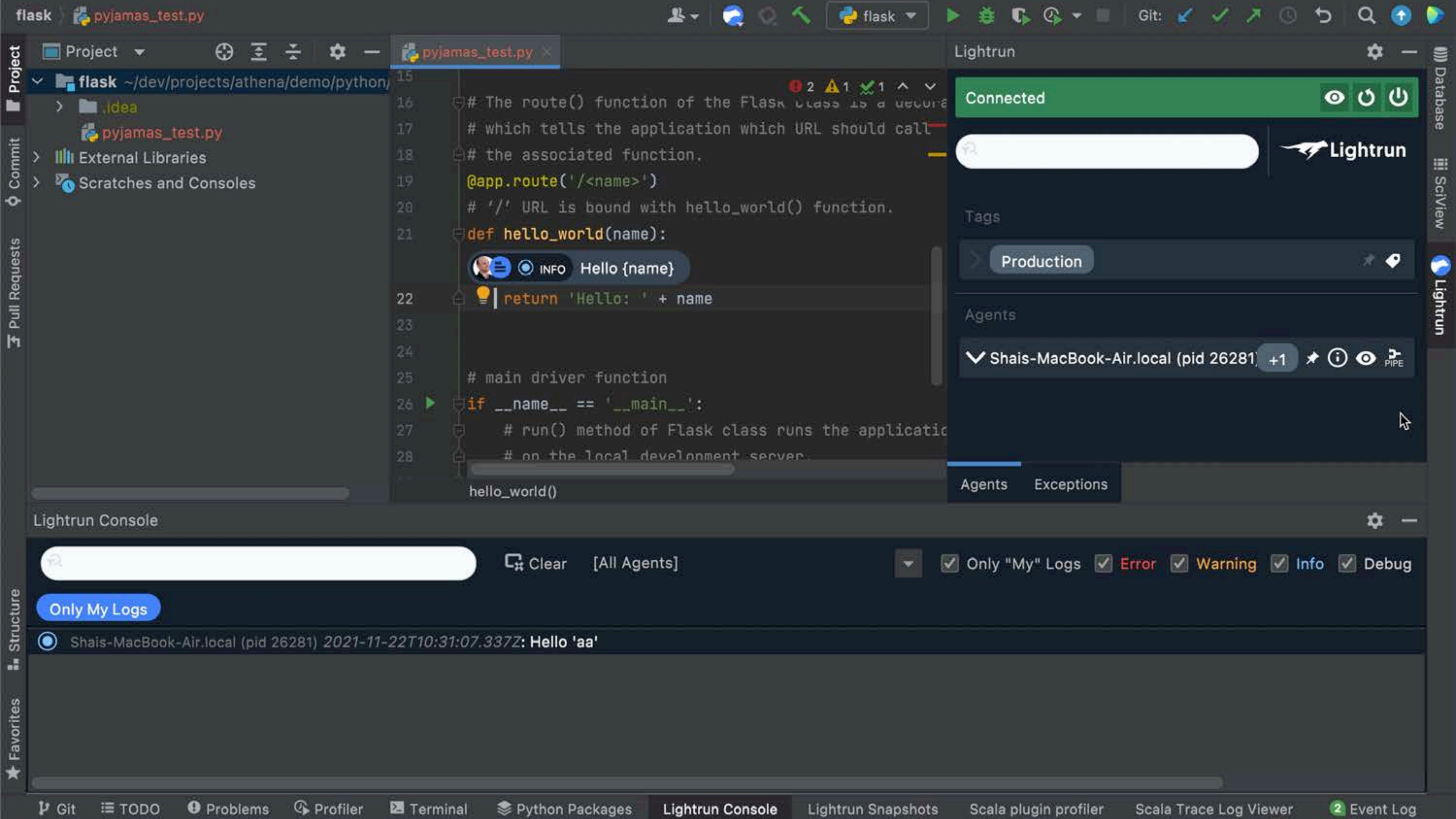
Logs appear here when you enable log piping.
You can enable log piping per agent in the Lightrun toolwindow (side drawer) on the right



```
# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def hello_world(name):
    INFO Hello {name}
    return 'Hello: ' + name


# main driver function
if __name__ == '__main__':
    # run() method of Flask class runs the applicatio
    # on the local development server.
```

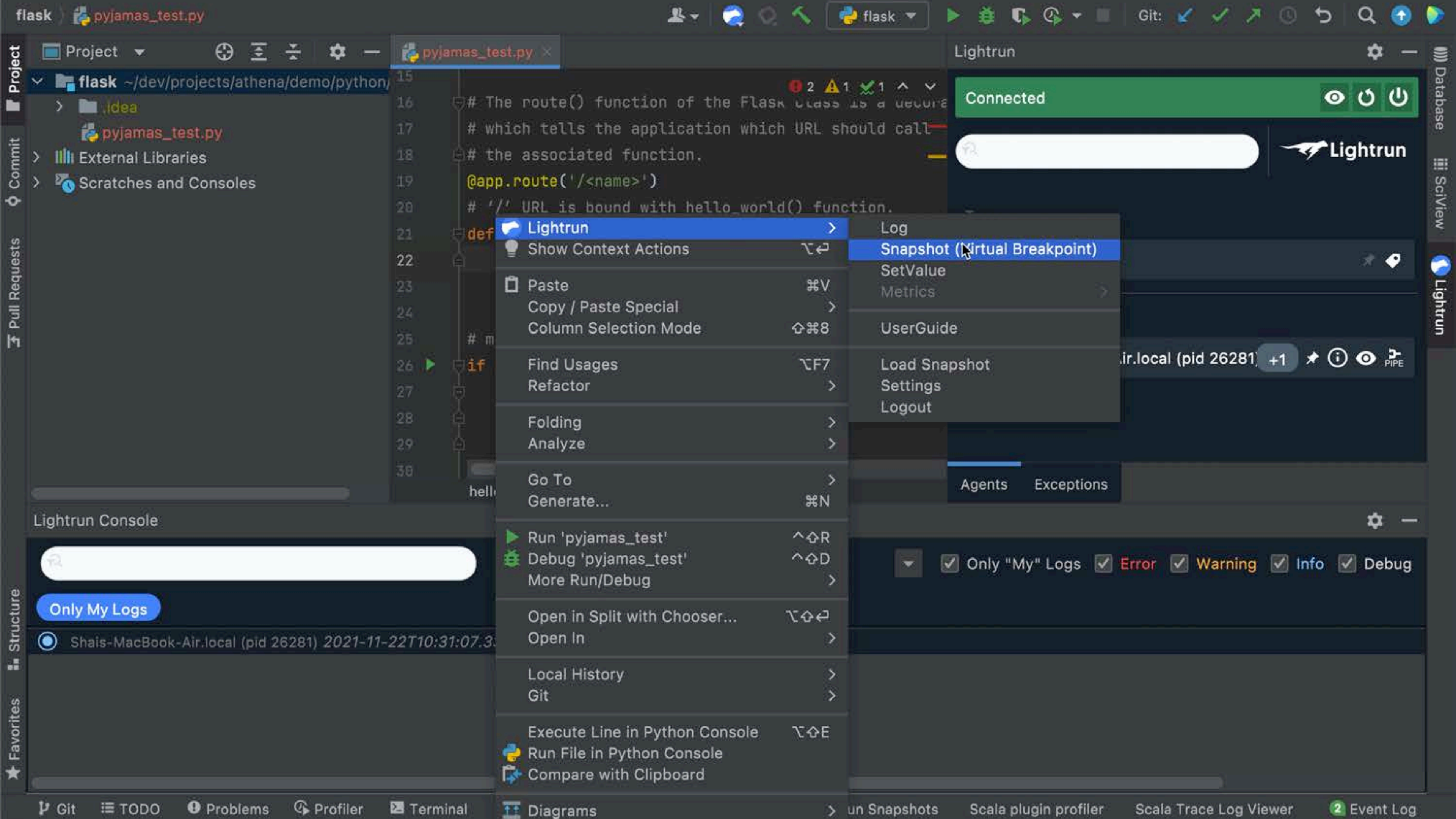
☑ Only "My" Logs ☑ Error ☑ Warning ☑ Info ☑ Debug

```
Shais-MacBook-Air.local (pid 26281) 2021-11-22T10:31:07.337Z: Hello 'aa'
```



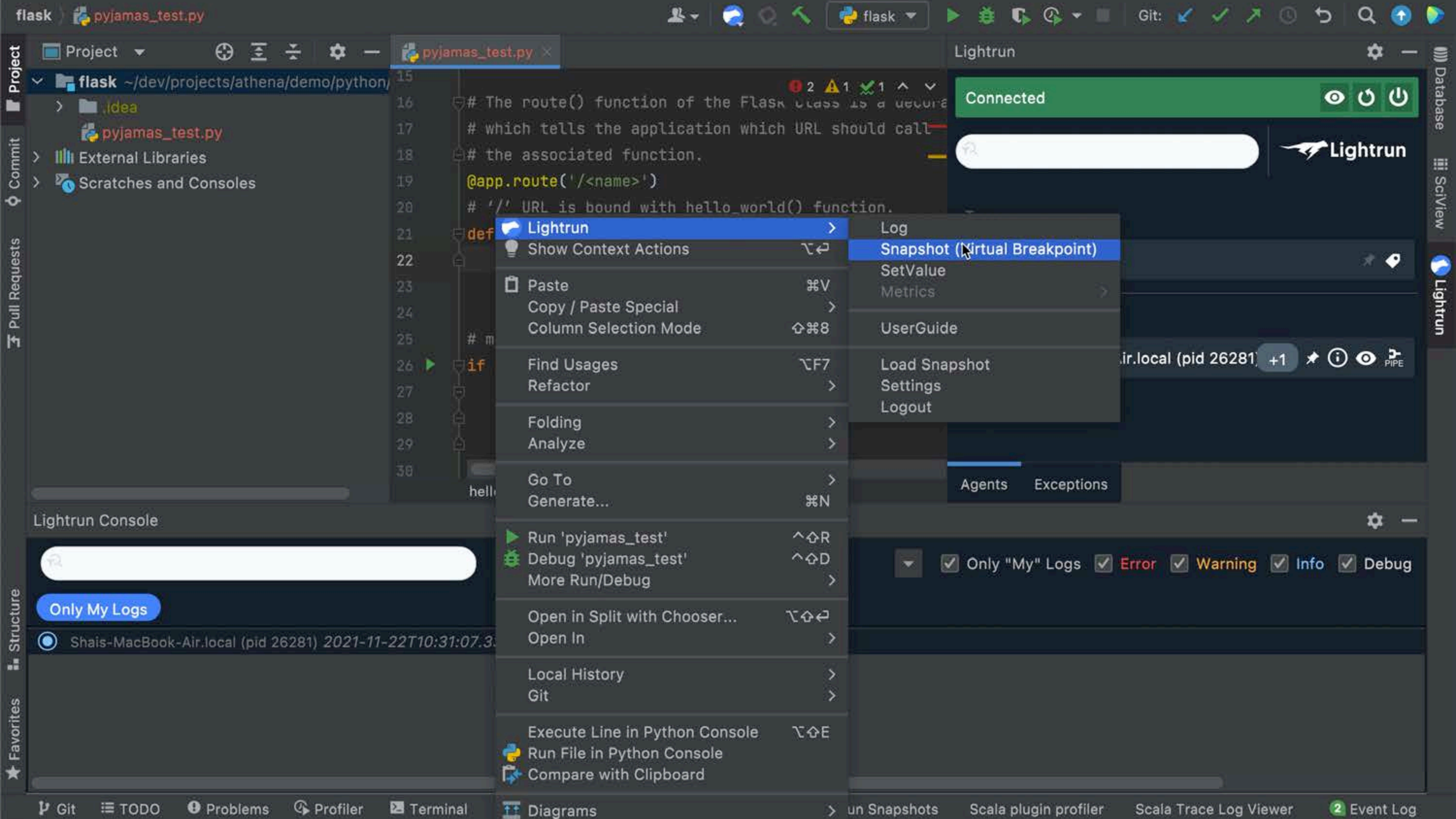
```
# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def hello_world(name):
    return 'Hello: ' + name


# main driver function
if __name__ == '__main__':
    # run() method of Flask class runs the applicatic
    # on the local development server.
```

```
Shais-MacBook-Air.local (pid 26281) 2021-11-22T10:31:07.337Z: Hello 'aa'
```



```
# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def



# m
if



hell
```

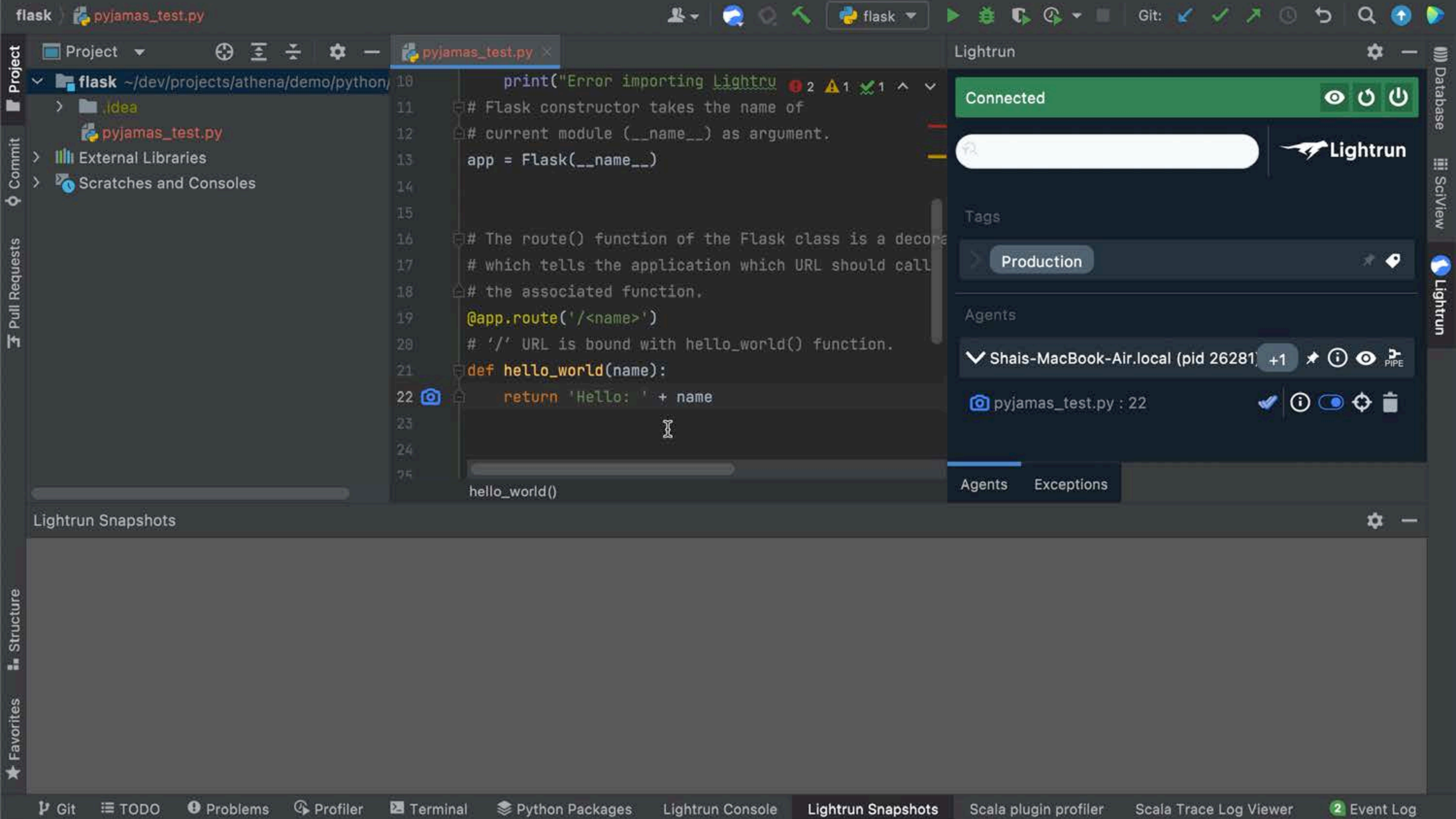
```
Shais-MacBook-Air.local (pid 26281) 2021-11-22T10:31:07.3
```



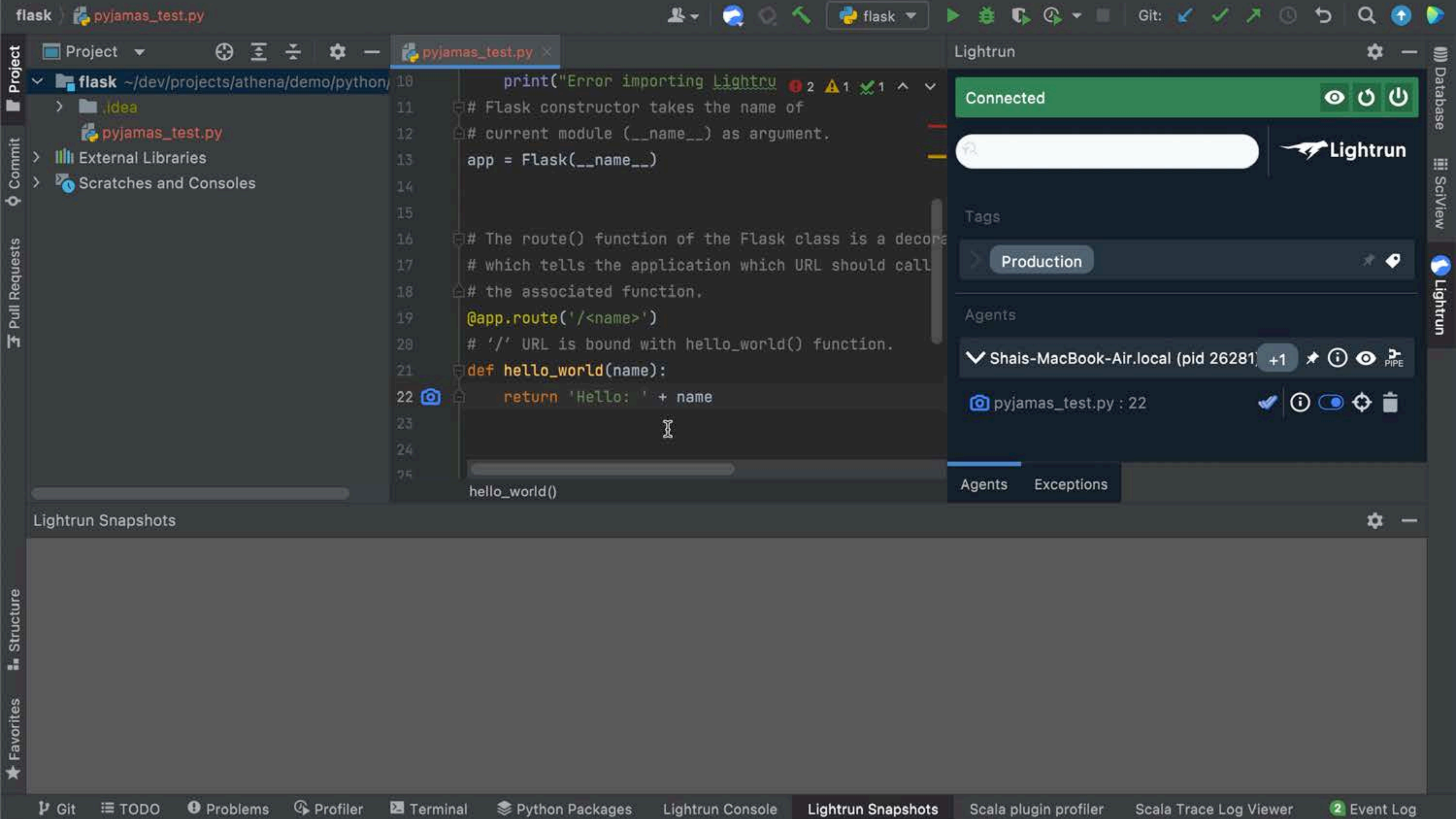
```
# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def
```

```
# m
if
```

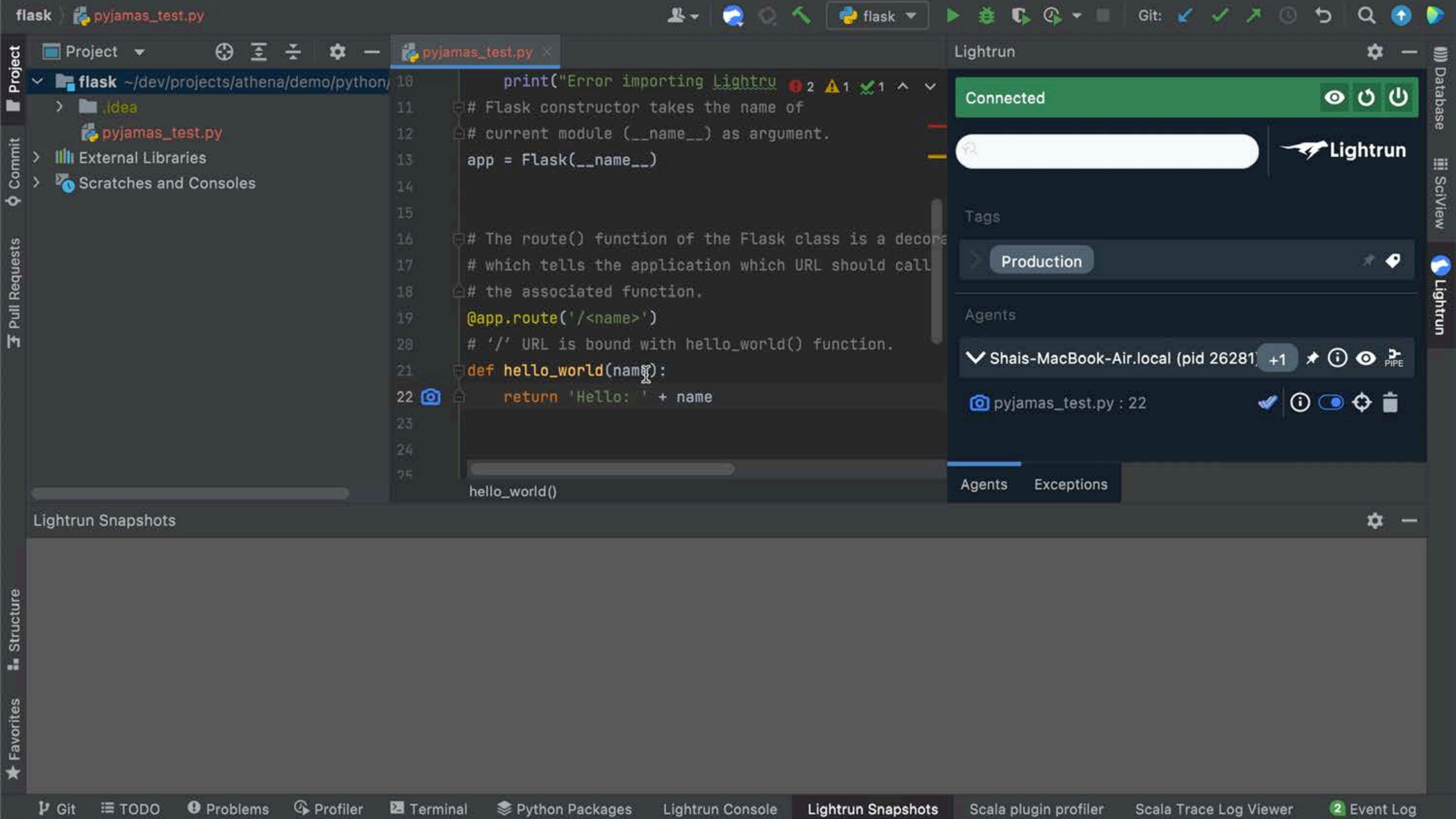
```
Shais-MacBook-Air.local (pid 26281) 2021-11-22T10:31:07.3
```



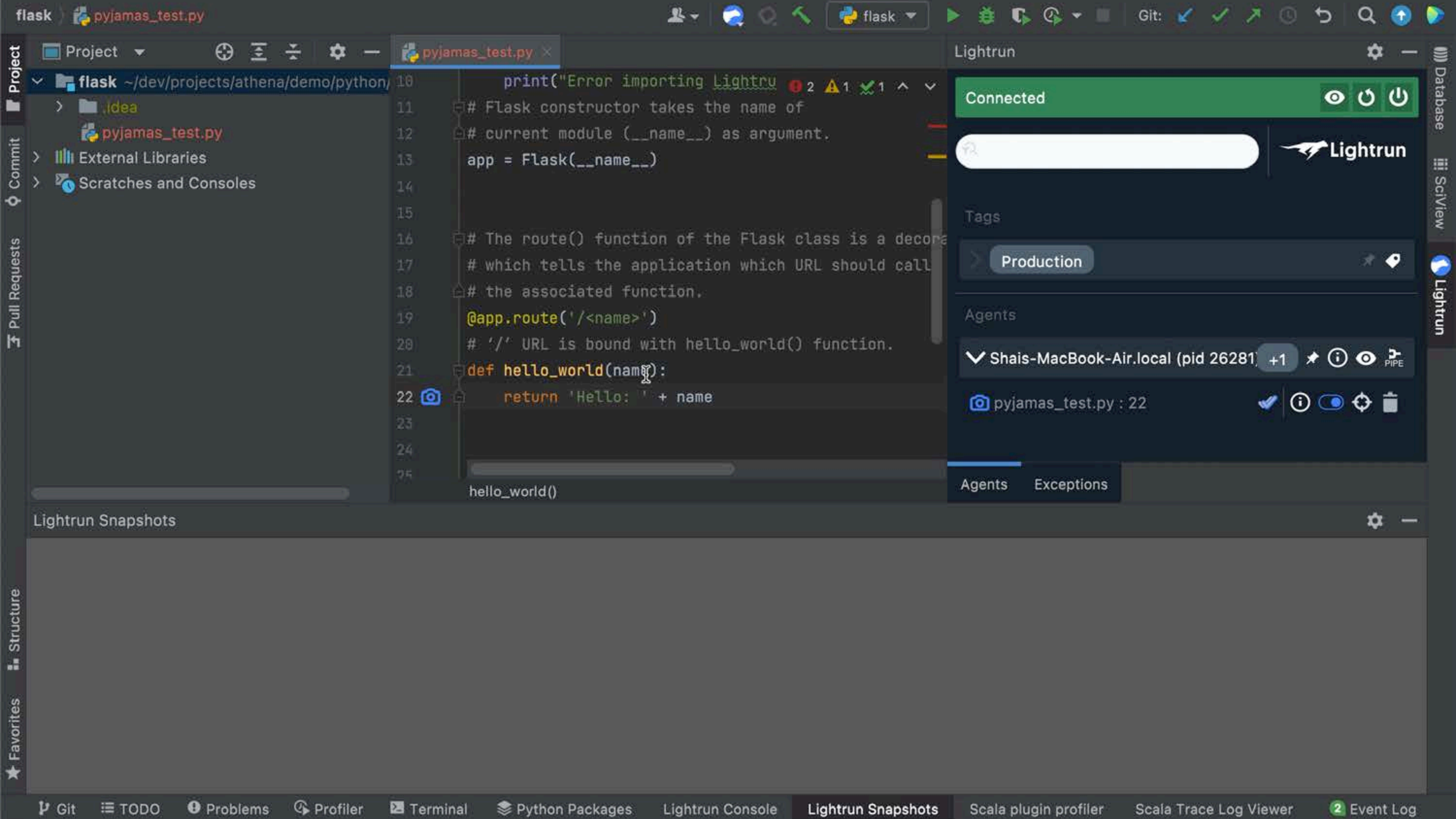
```
    print("Error importing Lightru
# Flask constructor takes the name of
# current module (__name__) as argument.
app = Flask(__name__)


# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def hello_world(name):
    return 'Hello: ' + name
```



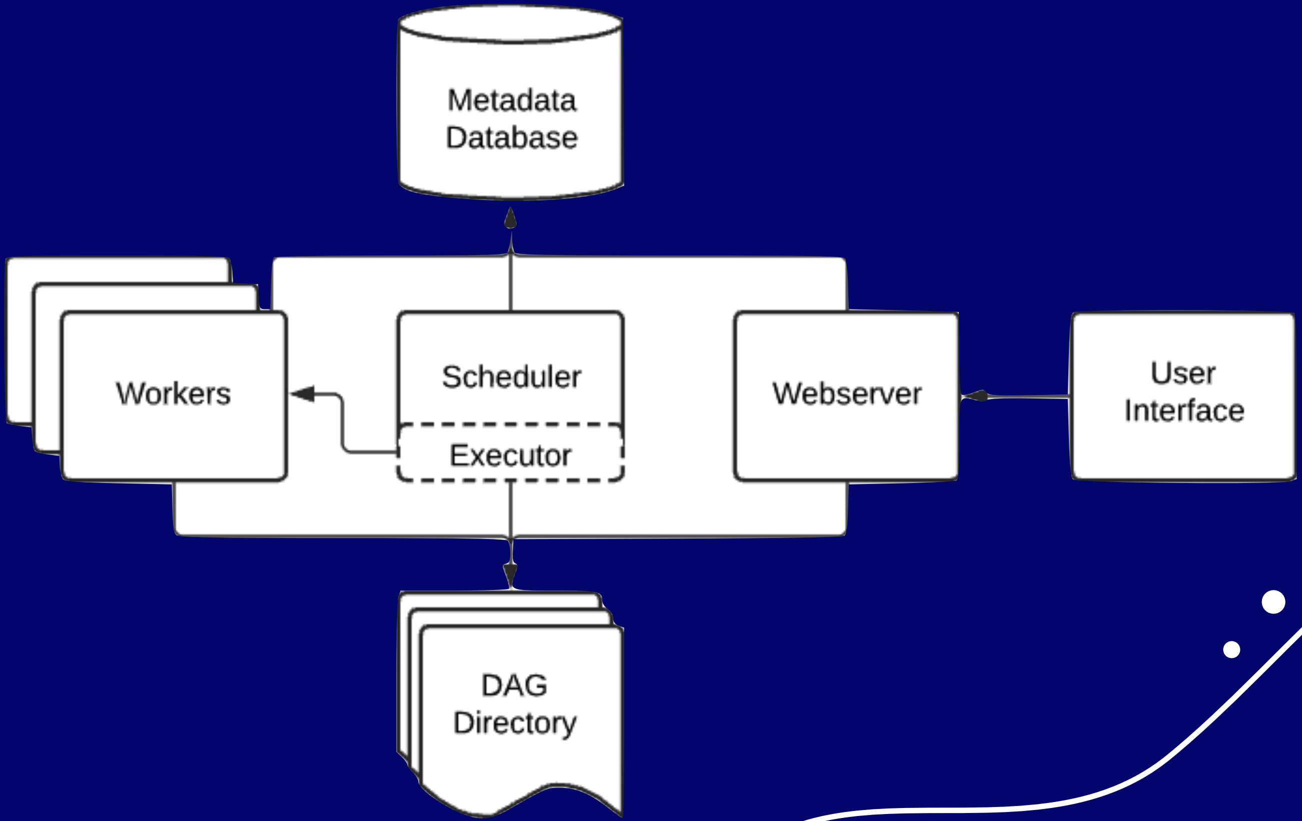
```
    print("Error importing Lightru
# Flask constructor takes the name of
# current module (__name__) as argument.
app = Flask(__name__)


# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# ‘/’ URL is bound with hello_world() function.
def hello_world(name):
    return 'Hello: ' + name
```

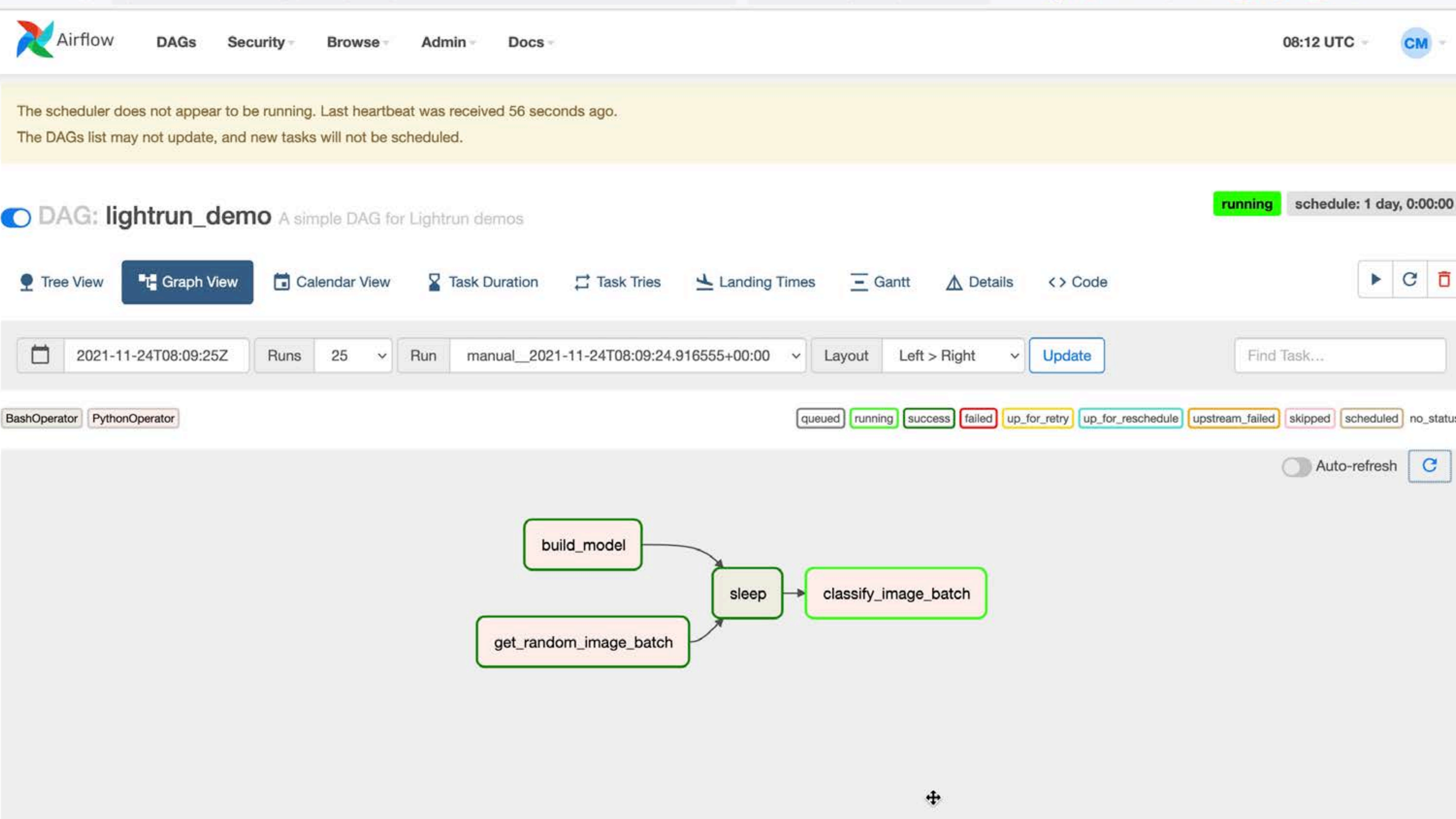


```
    print("Error importing Lightru
# Flask constructor takes the name of
# current module (__name__) as argument.
app = Flask(__name__)


# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def hello_world(name):
    return 'Hello: ' + name
```



```
    print("Error importing Lightru
# Flask constructor takes the name of
# current module (__name__) as argument.
app = Flask(__name__)


# The route() function of the Flask class is a decora
# which tells the application which URL should call
# the associated function.
@app.route('/<name>')
# '/' URL is bound with hello_world() function.
def hello_world(name):
    return 'Hello: ' + name
```


Metadata Database
Workers
Scheduler
Executor
Webserver
User Interface
DAG Directory



The scheduler does not appear to be running. Last heartbeat was received 56 seconds ago.
The DAGs list may not update, and new tasks will not be scheduled.

DAG: lightrun_demo A simple DAG for Lightrun demos

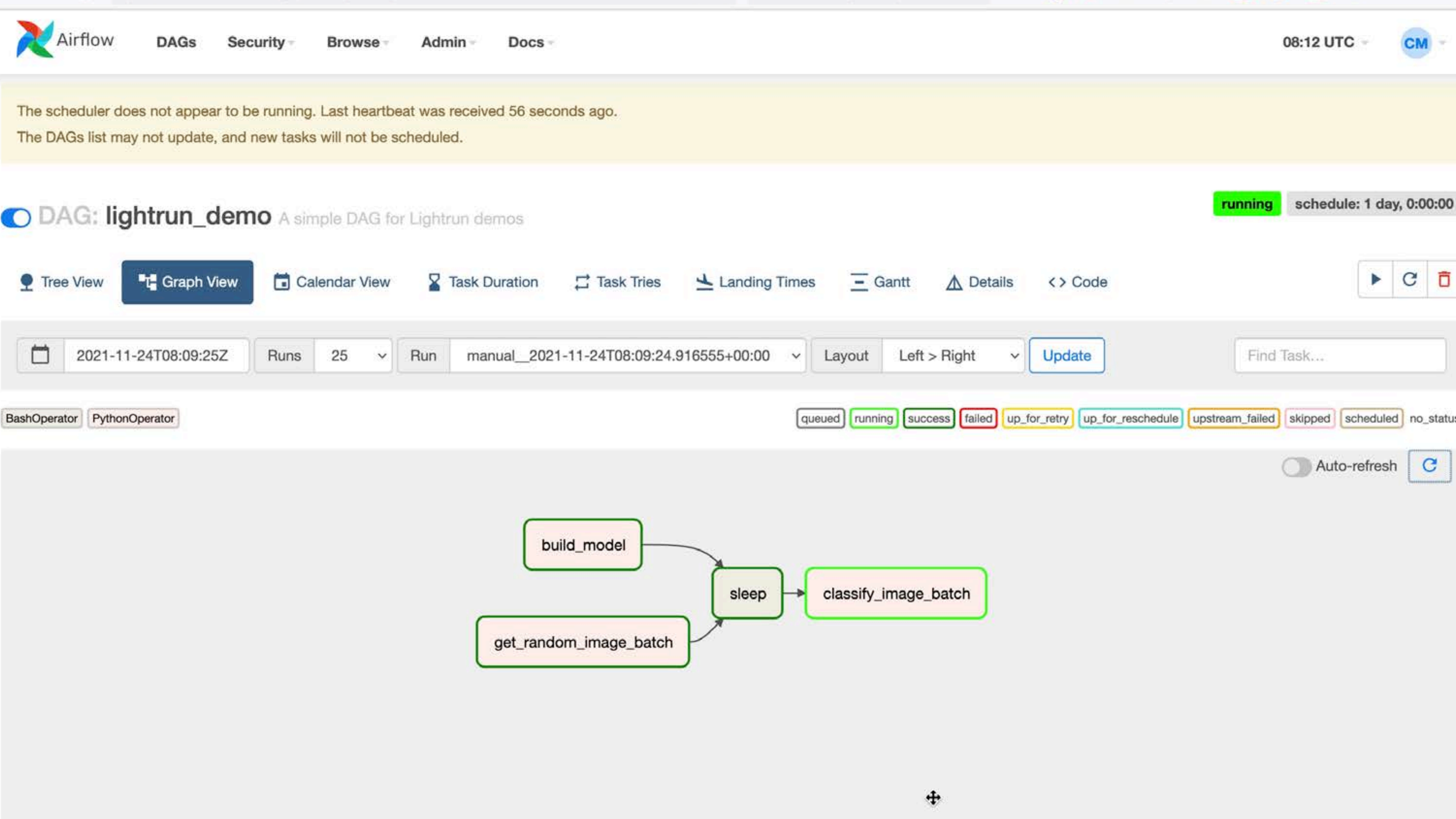
running schedule: 1 day, 0:00:00

2021-11-24T08:09:25Z

Runs 25

Run manual__2021-11-24T08:09:24.916555+00:00

Layout Left > Right

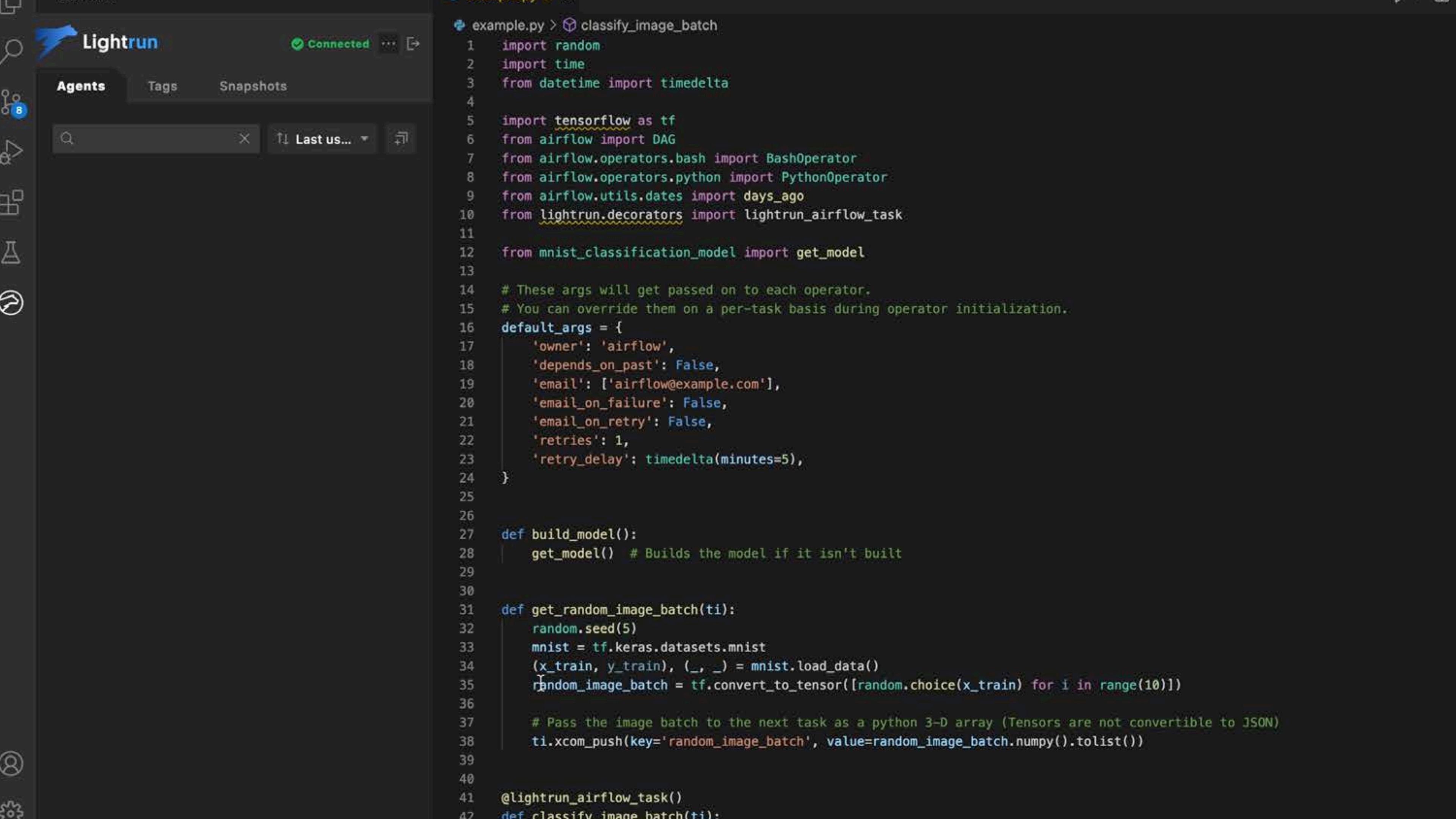


The scheduler does not appear to be running. Last heartbeat was received 56 seconds ago.
The DAGs list may not update, and new tasks will not be scheduled.

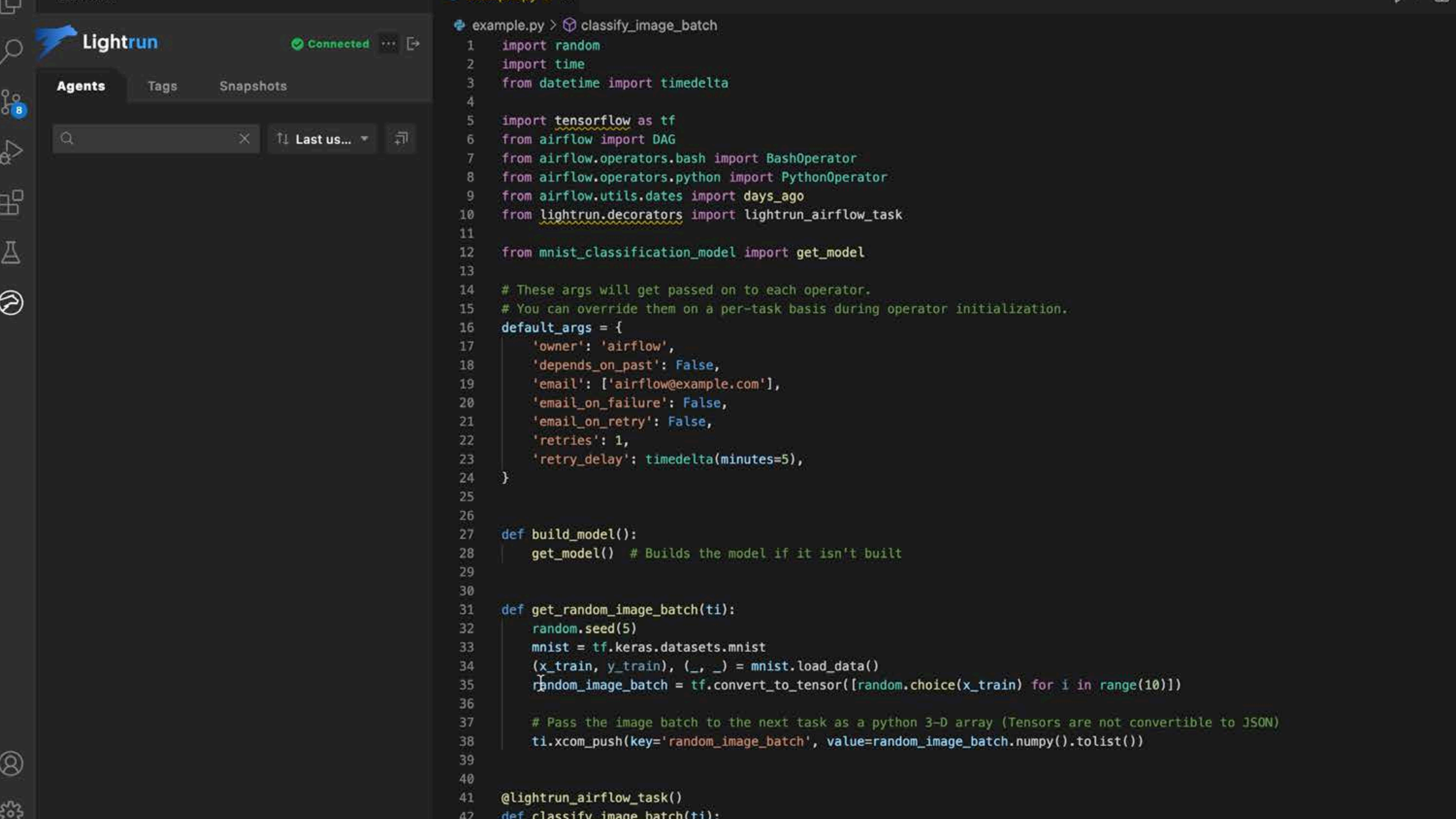
DAG: lightrun_demo A simple DAG for Lightrun demos

running schedule: 1 day, 0:00:00

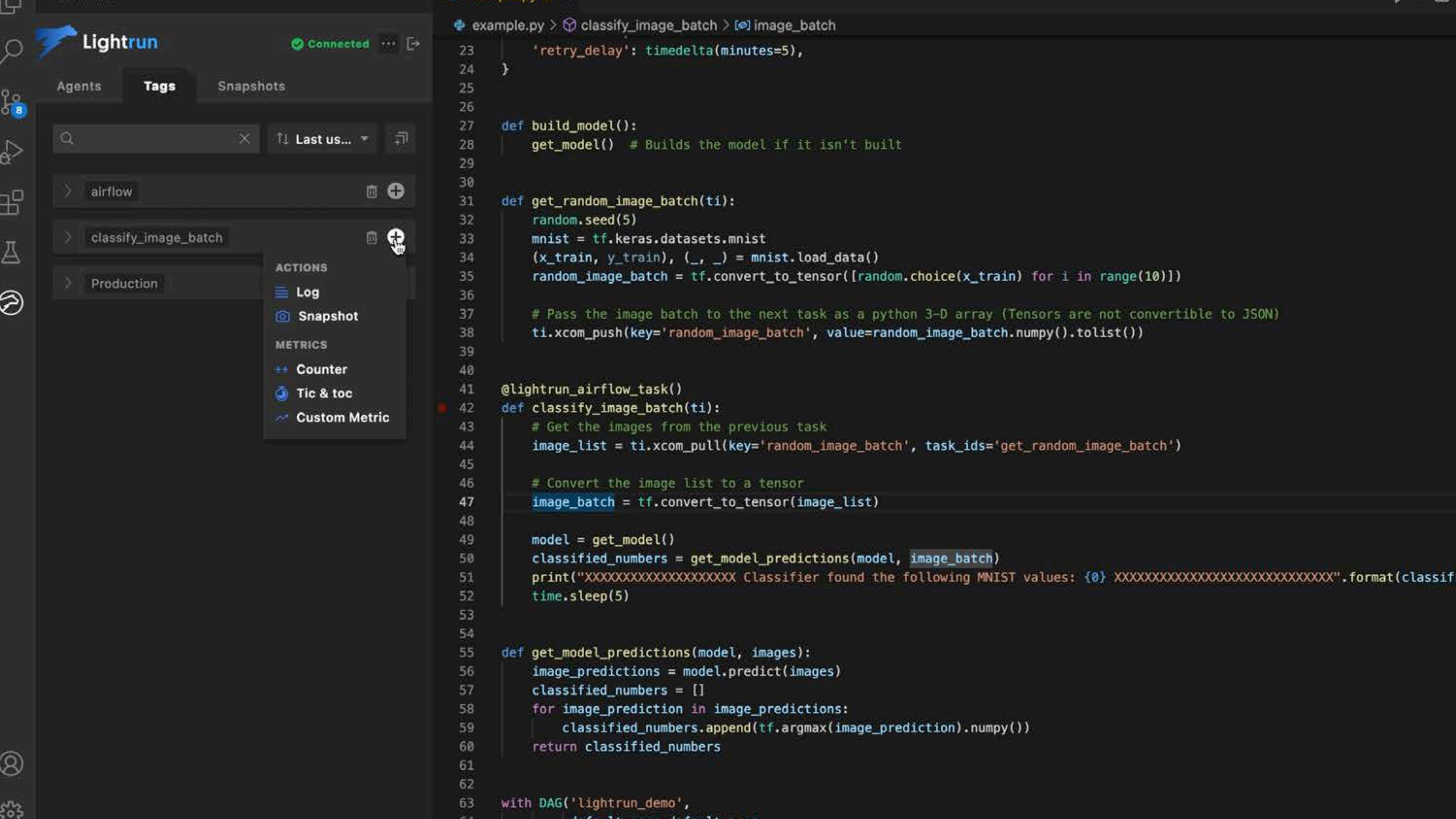
2021-11-24T08:09:25Z Runs 25 Run manual__2021-11-24T08:09:24.916555+00:00 Layout Left > Right



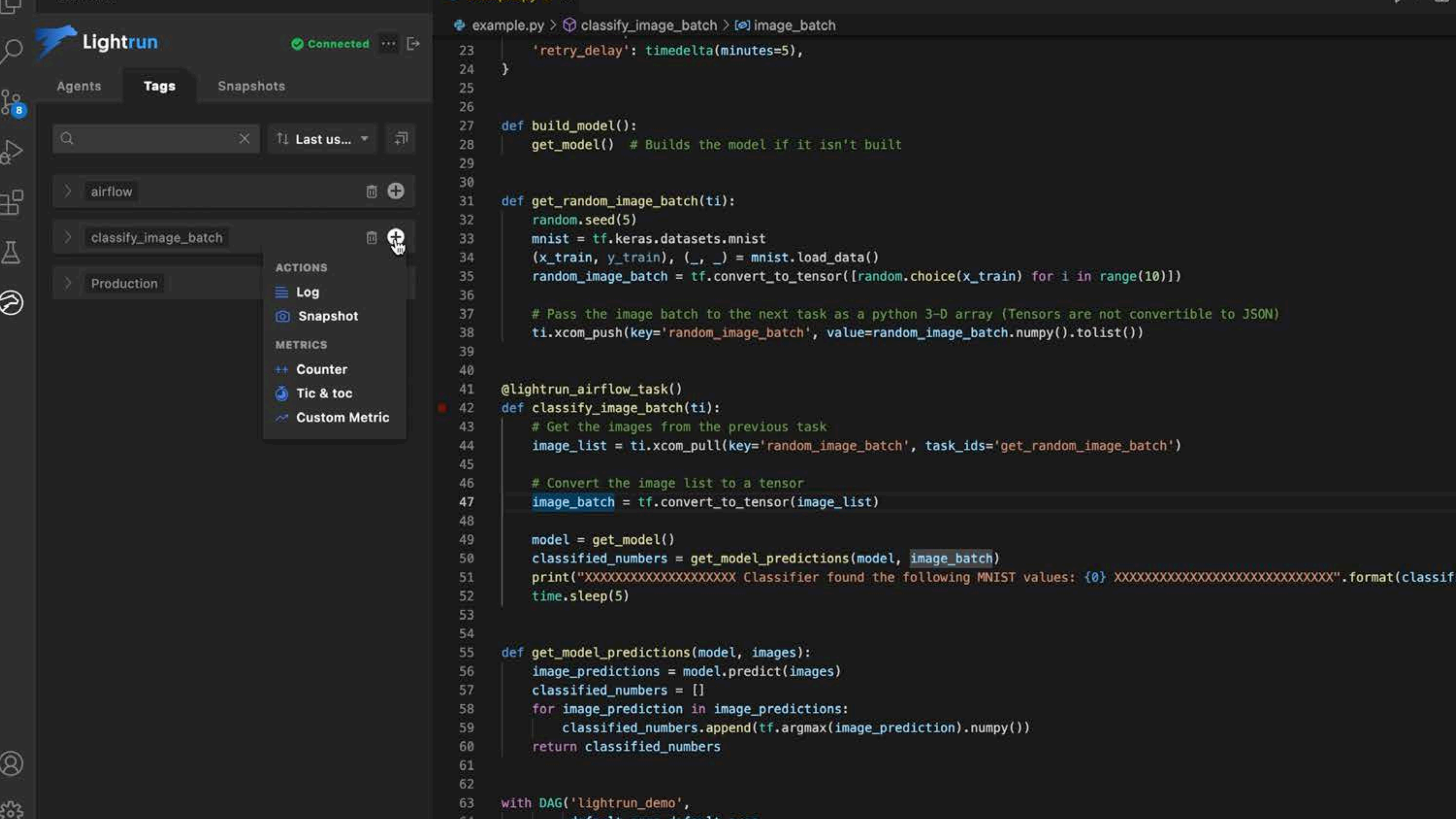
```python
import random
import time
from datetime import timedelta

import tensorflow as tf
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.operators.python import PythonOperator
from airflow.utils.dates import days_ago
from lightrun.decorators import lightrun_airflow_task

from mnist_classification_model import get_model

# These args will get passed on to each operator.
# You can override them on a per-task basis during operator initialization.
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}


def build_model():
    get_model()  # Builds the model if it isn't built


def get_random_image_batch(ti):
    random.seed(5)
    mnist = tf.keras.datasets.mnist
    (x_train, y_train), (_, _) = mnist.load_data()
    random_image_batch = tf.convert_to_tensor([random.choice(x_train) for i in range(10)])

    # Pass the image batch to the next task as a python 3-D array (Tensors are not convertible to JSON)
    ti.xcom_push(key='random_image_batch', value=random_image_batch.numpy().tolist())


@lightrun_airflow_task()
def classify_image_batch(ti):
```



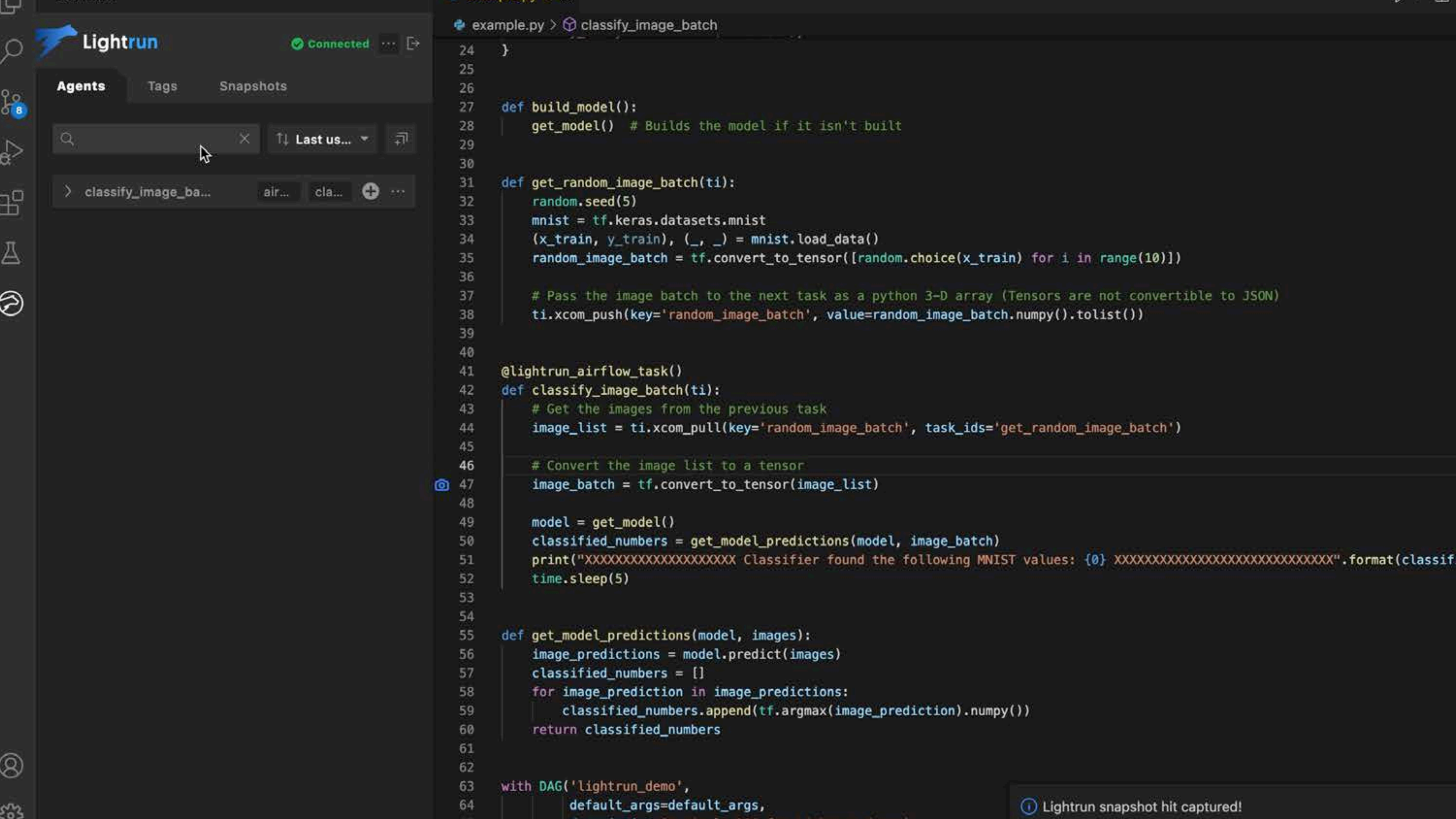
```python
import random
import time
from datetime import timedelta

import tensorflow as tf
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.operators.python import PythonOperator
from airflow.utils.dates import days_ago
from lightrun.decorators import lightrun_airflow_task

from mnist_classification_model import get_model

# These args will get passed on to each operator.
# You can override them on a per-task basis during operator initialization.
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}


def build_model():
    get_model()  # Builds the model if it isn't built


def get_random_image_batch(ti):
    random.seed(5)
    mnist = tf.keras.datasets.mnist
    (x_train, y_train), (_, _) = mnist.load_data()
    random_image_batch = tf.convert_to_tensor([random.choice(x_train) for i in range(10)])

    # Pass the image batch to the next task as a python 3-D array (Tensors are not convertible to JSON)
    ti.xcom_push(key='random_image_batch', value=random_image_batch.numpy().tolist())


@lightrun_airflow_task()
def classify_image_batch(ti):
```



```
    'retry_delay': timedelta(minutes=5),
}


def build_model():
    get_model()  # Builds the model if it isn't built


def get_random_image_batch(ti):
    random.seed(5)
    mnist = tf.keras.datasets.mnist
    (x_train, y_train), (_, _) = mnist.load_data()
    random_image_batch = tf.convert_to_tensor([random.choice(x_train) for i in range(10)])

    # Pass the image batch to the next task as a python 3-D array (Tensors are not convertible to JSON)
    ti.xcom_push(key='random_image_batch', value=random_image_batch.numpy().tolist())


@lightrun_airflow_task()
def classify_image_batch(ti):
    # Get the images from the previous task
    image_list = ti.xcom_pull(key='random_image_batch', task_ids='get_random_image_batch')

    # Convert the image list to a tensor
    image_batch = tf.convert_to_tensor(image_list)

    model = get_model()
    classified_numbers = get_model_predictions(model, image_batch)
    print("XXXXXXXXXXXXXXXXXXXX Classifier found the following MNIST values: {0} XXXXXXXXXXXXXXXXXXXXXXXXXXXXX".format(classif
    time.sleep(5)


def get_model_predictions(model, images):
    image_predictions = model.predict(images)
    classified_numbers = []
    for image_prediction in image_predictions:
        classified_numbers.append(tf.argmax(image_prediction).numpy())
    return classified_numbers


with DAG('lightrun_demo',
```

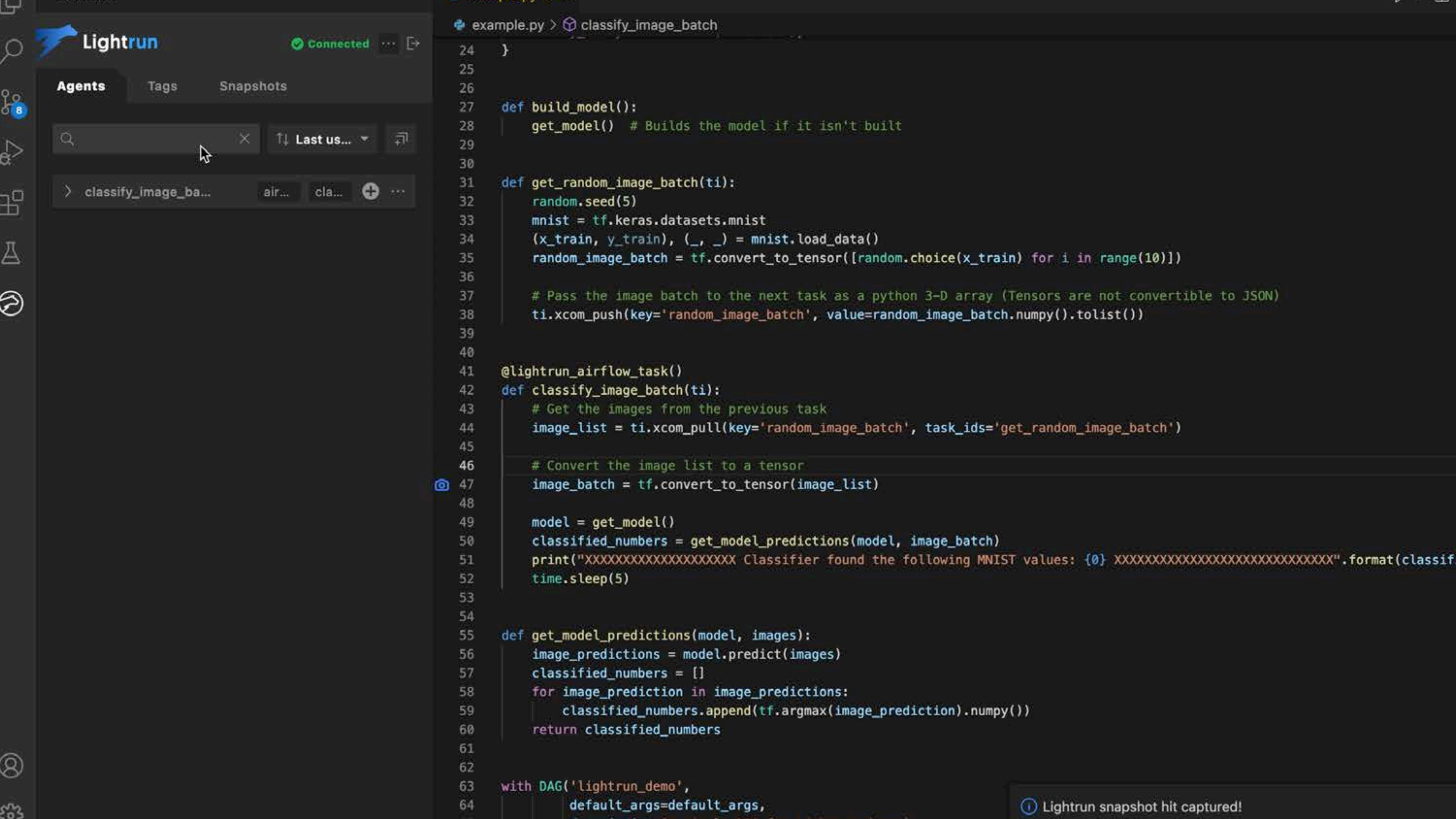


```python
    'retry_delay': timedelta(minutes=5),
}


def build_model():
    get_model()  # Builds the model if it isn't built


def get_random_image_batch(ti):
    random.seed(5)
    mnist = tf.keras.datasets.mnist
    (x_train, y_train), (_, _) = mnist.load_data()
    random_image_batch = tf.convert_to_tensor([random.choice(x_train) for i in range(10)])

    # Pass the image batch to the next task as a python 3-D array (Tensors are not convertible to JSON)
    ti.xcom_push(key='random_image_batch', value=random_image_batch.numpy().tolist())


@lightrun_airflow_task()
def classify_image_batch(ti):
    # Get the images from the previous task
    image_list = ti.xcom_pull(key='random_image_batch', task_ids='get_random_image_batch')

    # Convert the image list to a tensor
    image_batch = tf.convert_to_tensor(image_list)

    model = get_model()
    classified_numbers = get_model_predictions(model, image_batch)
    print("XXXXXXXXXXXXXXXXXXXX Classifier found the following MNIST values: {0} XXXXXXXXXXXXXXXXXXXXXXXXXXXXX".format(classif
    time.sleep(5)


def get_model_predictions(model, images):
    image_predictions = model.predict(images)
    classified_numbers = []
    for image_prediction in image_predictions:
        classified_numbers.append(tf.argmax(image_prediction).numpy())
    return classified_numbers


with DAG('lightrun_demo',
```



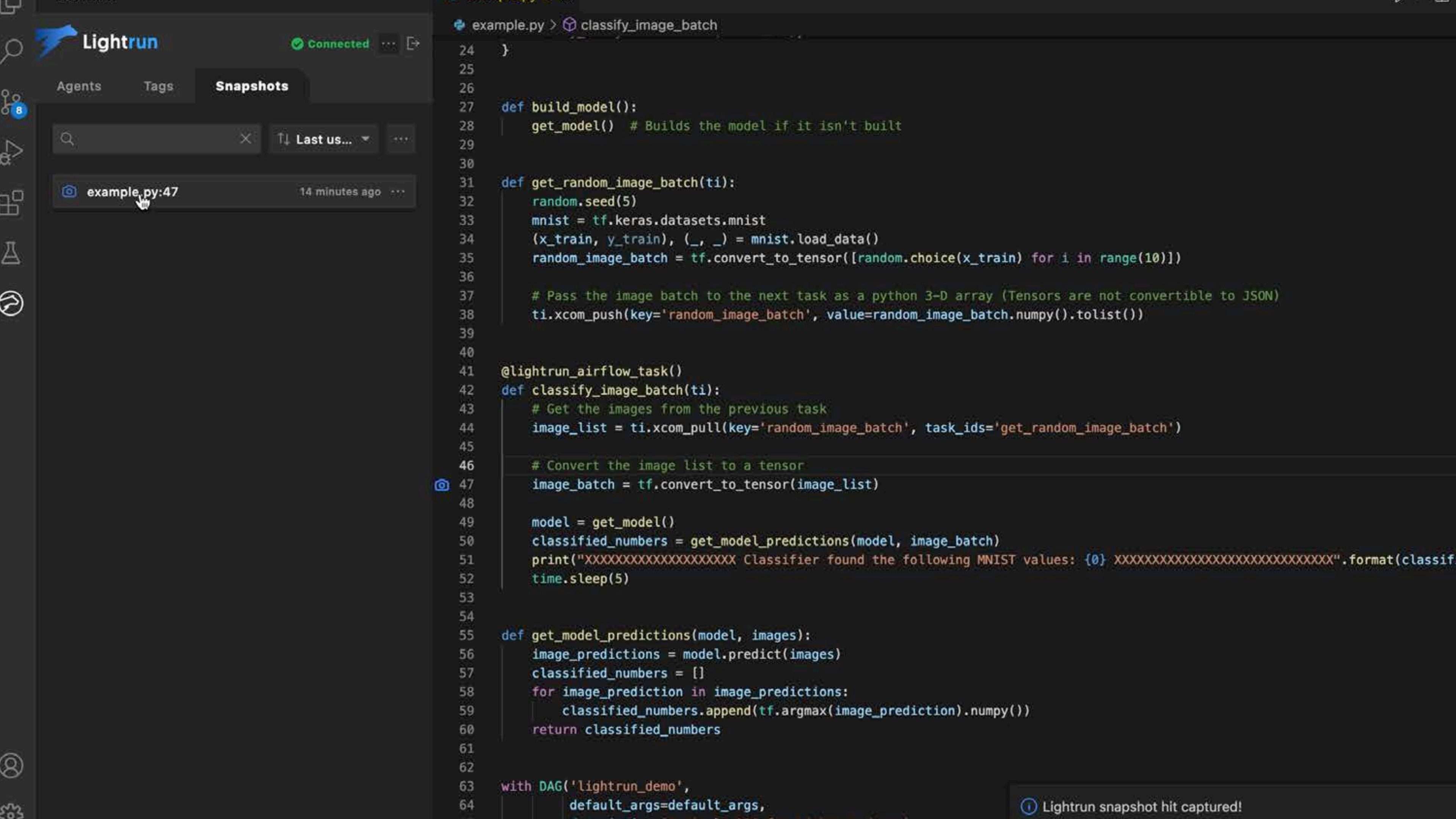
```python
}


def build_model():
    get_model()  # Builds the model if it isn't built


def get_random_image_batch(ti):
    random.seed(5)
    mnist = tf.keras.datasets.mnist
    (x_train, y_train), (_, _) = mnist.load_data()
    random_image_batch = tf.convert_to_tensor([random.choice(x_train) for i in range(10)])

    # Pass the image batch to the next task as a python 3-D array (Tensors are not convertible to JSON)
    ti.xcom_push(key='random_image_batch', value=random_image_batch.numpy().tolist())


@lightrun_airflow_task()
def classify_image_batch(ti):
    # Get the images from the previous task
    image_list = ti.xcom_pull(key='random_image_batch', task_ids='get_random_image_batch')

    # Convert the image list to a tensor
    image_batch = tf.convert_to_tensor(image_list)

    model = get_model()
    classified_numbers = get_model_predictions(model, image_batch)
    print("XXXXXXXXXXXXXXXXXXXX Classifier found the following MNIST values: {0} XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX".format(classif
    time.sleep(5)


def get_model_predictions(model, images):
    image_predictions = model.predict(images)
    classified_numbers = []
    for image_prediction in image_predictions:
        classified_numbers.append(tf.argmax(image_prediction).numpy())
    return classified_numbers


with DAG('lightrun_demo',
        default_args=default_args,
```

Lightrun snapshot hit captured!



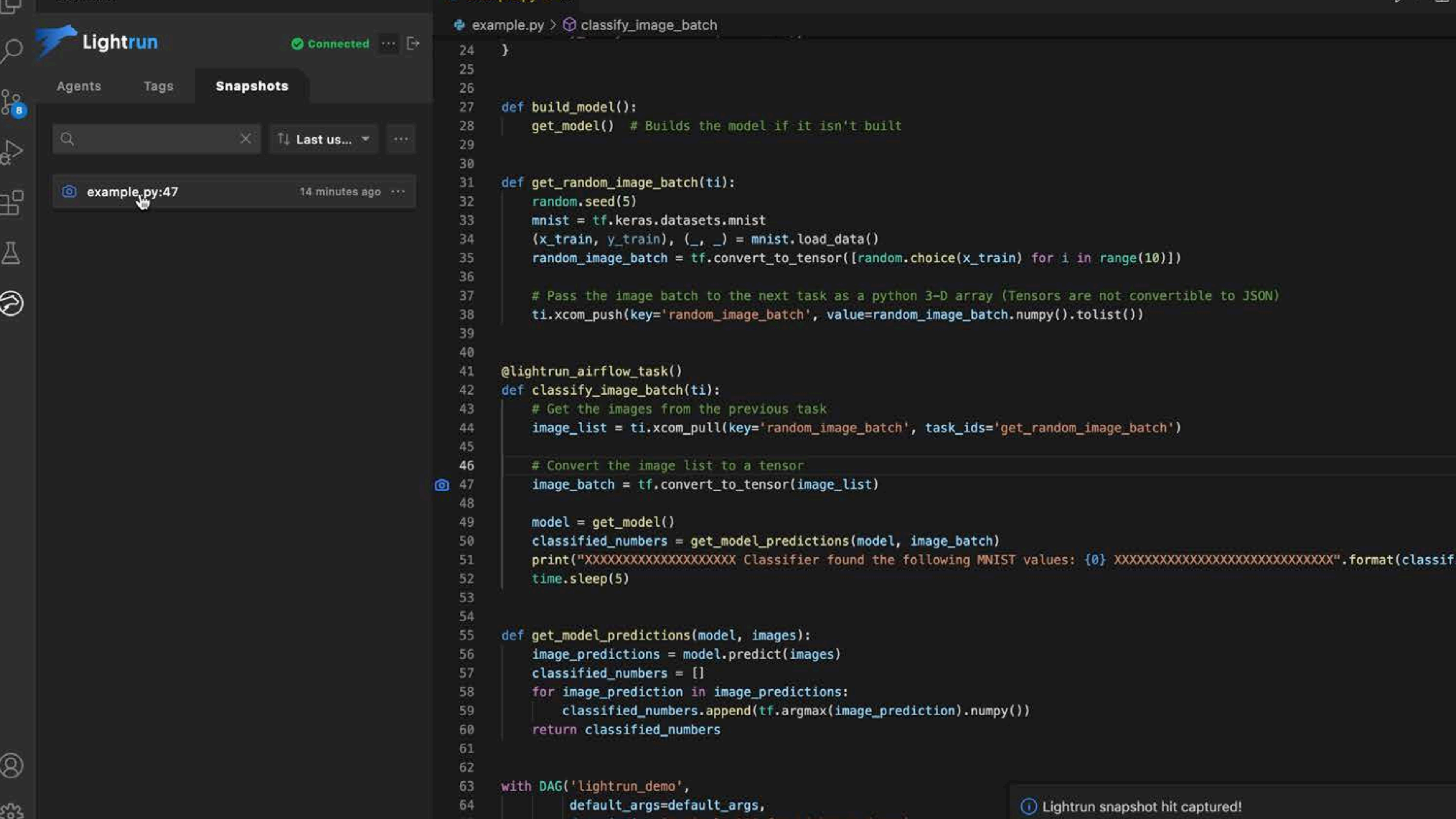
```python
}


def build_model():
    get_model()  # Builds the model if it isn't built


def get_random_image_batch(ti):
    random.seed(5)
    mnist = tf.keras.datasets.mnist
    (x_train, y_train), (_, _) = mnist.load_data()
    random_image_batch = tf.convert_to_tensor([random.choice(x_train) for i in range(10)])

    # Pass the image batch to the next task as a python 3-D array (Tensors are not convertible to JSON)
    ti.xcom_push(key='random_image_batch', value=random_image_batch.numpy().tolist())


@lightrun_airflow_task()
def classify_image_batch(ti):
    # Get the images from the previous task
    image_list = ti.xcom_pull(key='random_image_batch', task_ids='get_random_image_batch')

    # Convert the image list to a tensor
    image_batch = tf.convert_to_tensor(image_list)

    model = get_model()
    classified_numbers = get_model_predictions(model, image_batch)
    print("XXXXXXXXXXXXXXXXXXXX Classifier found the following MNIST values: {0} XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX".format(classif
    time.sleep(5)


def get_model_predictions(model, images):
    image_predictions = model.predict(images)
    classified_numbers = []
    for image_prediction in image_predictions:
        classified_numbers.append(tf.argmax(image_prediction).numpy())
    return classified_numbers


with DAG('lightrun_demo',
         default_args=default_args,
```

Lightrun snapshot hit captured!



```
}


def build_model():
    get_model()  # Builds the model if it isn't built


def get_random_image_batch(ti):
    random.seed(5)
    mnist = tf.keras.datasets.mnist
    (x_train, y_train), (_, _) = mnist.load_data()
    random_image_batch = tf.convert_to_tensor([random.choice(x_train) for i in range(10)])

    # Pass the image batch to the next task as a python 3-D array (Tensors are not convertible to JSON)
    ti.xcom_push(key='random_image_batch', value=random_image_batch.numpy().tolist())


@lightrun_airflow_task()
def classify_image_batch(ti):
    # Get the images from the previous task
    image_list = ti.xcom_pull(key='random_image_batch', task_ids='get_random_image_batch')

    # Convert the image list to a tensor
    image_batch = tf.convert_to_tensor(image_list)

    model = get_model()
    classified_numbers = get_model_predictions(model, image_batch)
    print("XXXXXXXXXXXXXXXXXXXX Classifier found the following MNIST values: {0} XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX".format(classif
    time.sleep(5)


def get_model_predictions(model, images):
    image_predictions = model.predict(images)
    classified_numbers = []
    for image_prediction in image_predictions:
        classified_numbers.append(tf.argmax(image_prediction).numpy())
    return classified_numbers


with DAG('lightrun_demo',
         default_args=default_args,
```

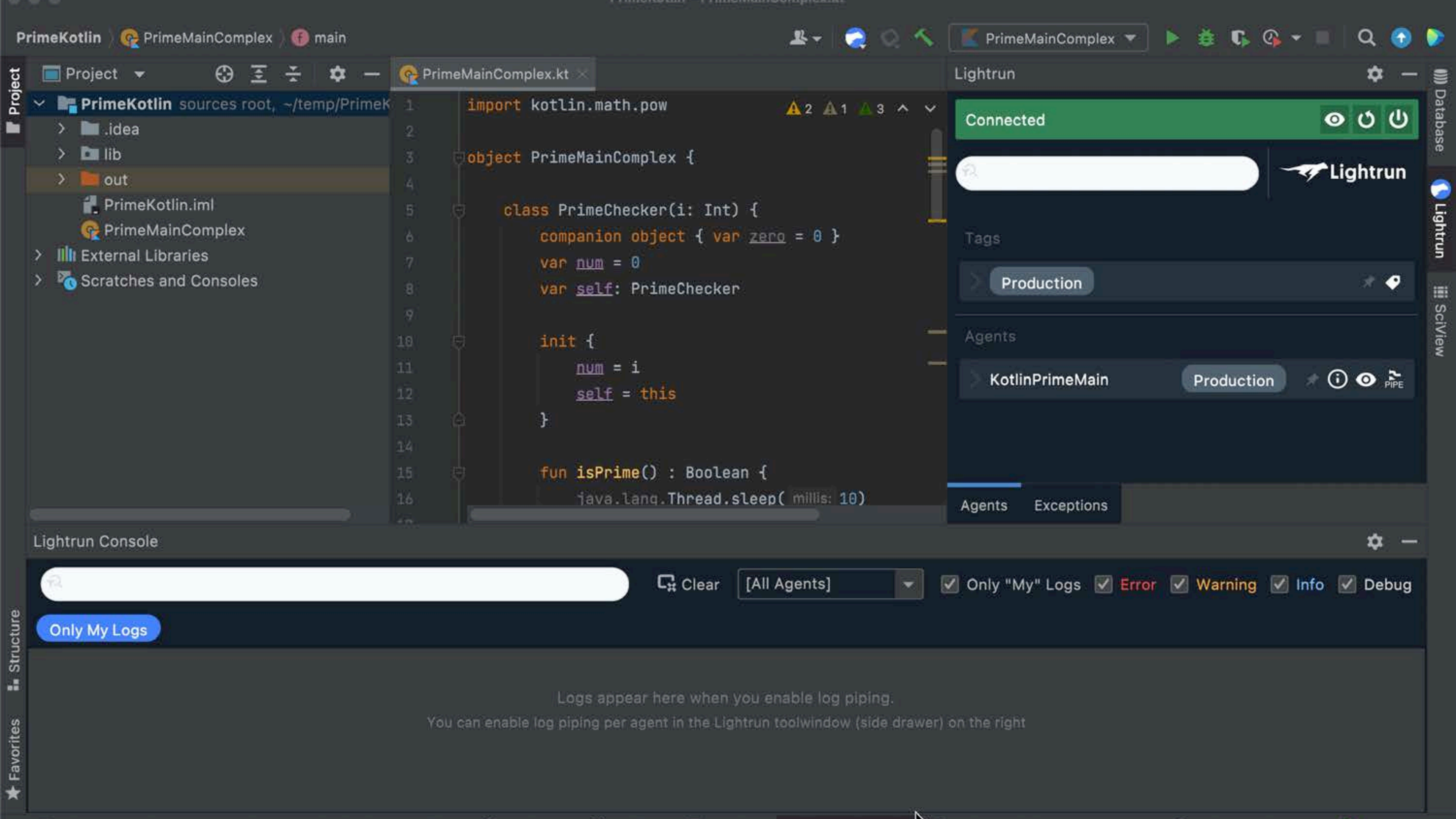
Lightrun snapshot hit captured!



```
}


def build_model():
    get_model()  # Builds the model if it isn't built


def get_random_image_batch(ti):
    random.seed(5)
    mnist = tf.keras.datasets.mnist
    (x_train, y_train), (_, _) = mnist.load_data()
    random_image_batch = tf.convert_to_tensor([random.choice(x_train) for i in range(10)])

    # Pass the image batch to the next task as a python 3-D array (Tensors are not convertible to JSON)
    ti.xcom_push(key='random_image_batch', value=random_image_batch.numpy().tolist())


@lightrun_airflow_task()
def classify_image_batch(ti):
    # Get the images from the previous task
    image_list = ti.xcom_pull(key='random_image_batch', task_ids='get_random_image_batch')

    # Convert the image list to a tensor
    image_batch = tf.convert_to_tensor(image_list)

    model = get_model()
    classified_numbers = get_model_predictions(model, image_batch)
    print("XXXXXXXXXXXXXXXXXXXX Classifier found the following MNIST values: {0} XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX".format(classif
    time.sleep(5)


def get_model_predictions(model, images):
    image_predictions = model.predict(images)
    classified_numbers = []
    for image_prediction in image_predictions:
        classified_numbers.append(tf.argmax(image_prediction).numpy())
    return classified_numbers


with DAG('lightrun_demo',
         default_args=default_args,
```

Lightrun snapshot hit captured!

Demo: PrimeMain, Kotlin



```
import kotlin.math.pow

object PrimeMainComplex {

    class PrimeChecker(i: Int) {
        companion object { var zero = 0 }
        var num = 0
        var self: PrimeChecker

        init {
            num = i
            self = this
        }

        fun isPrime() : Boolean {
            java.lang.Thread.sleep( millis: 10)
```

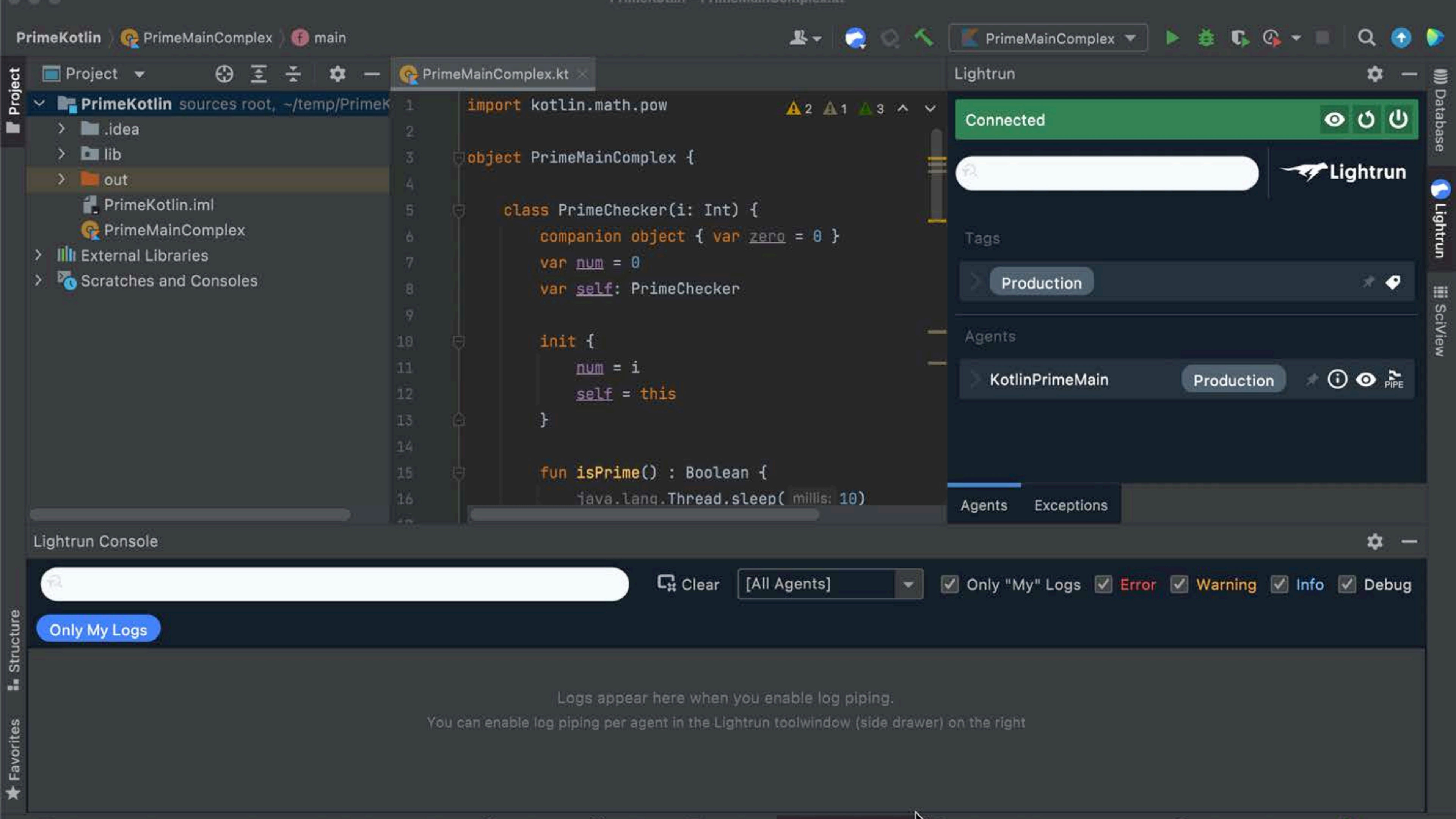
Connected

Tags: Production

Agents: KotlinPrimeMain Production

[All Agents] ☑ Only "My" Logs ☑ Error ☑ Warning ☑ Info ☑ Debug

Logs appear here when you enable log piping.
You can enable log piping per agent in the Lightrun toolwindow (side drawer) on the right



```
import kotlin.math.pow

object PrimeMainComplex {

    class PrimeChecker(i: Int) {
        companion object { var zero = 0 }
        var num = 0
        var self: PrimeChecker

        init {
            num = i
            self = this
        }

        fun isPrime() : Boolean {
            java.lang.Thread.sleep( millis: 10)
```

Connected

Tags

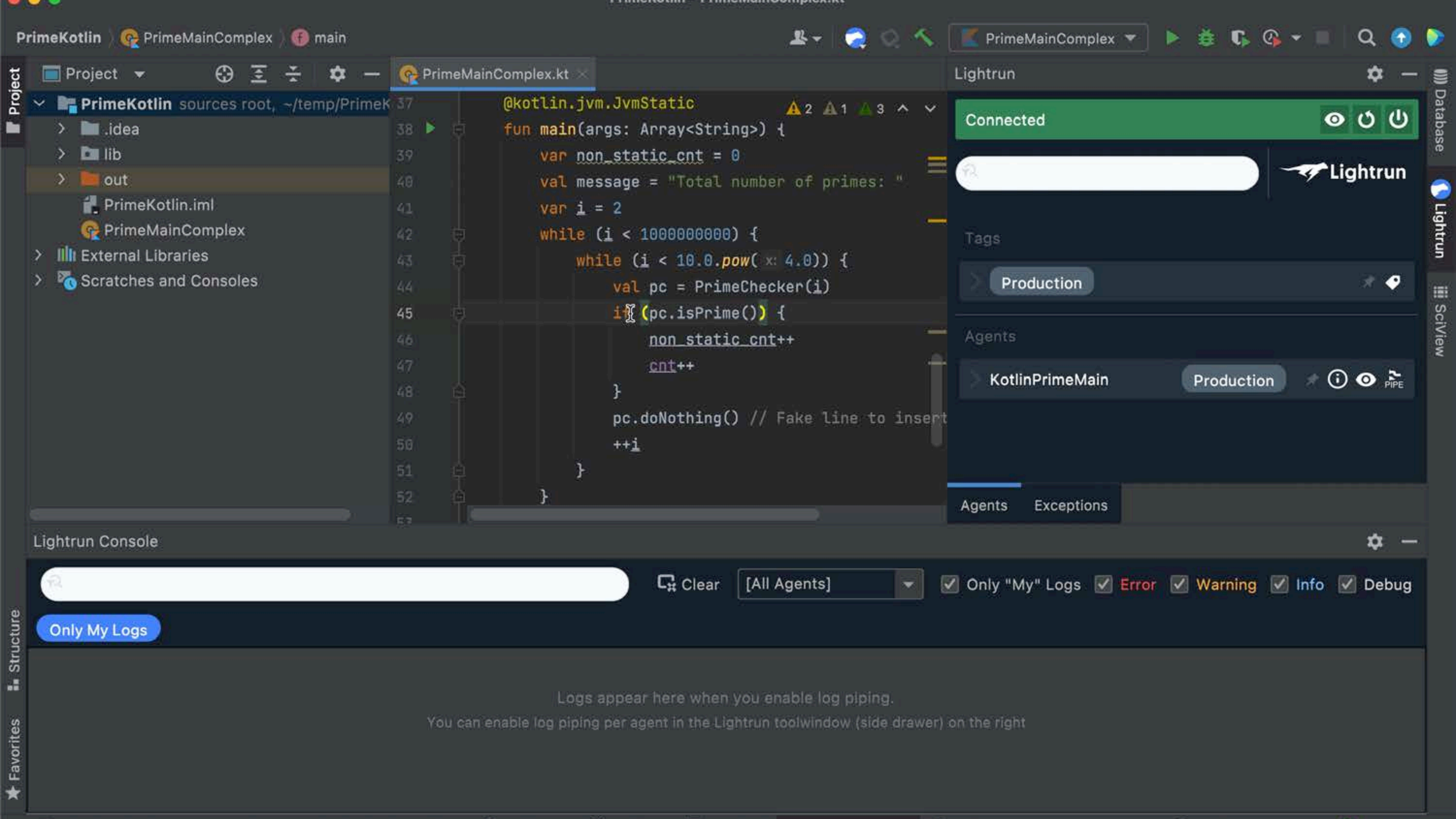
Production

Agents

KotlinPrimeMain Production

[All Agents] ☑ Only "My" Logs ☑ Error ☑ Warning ☑ Info ☑ Debug

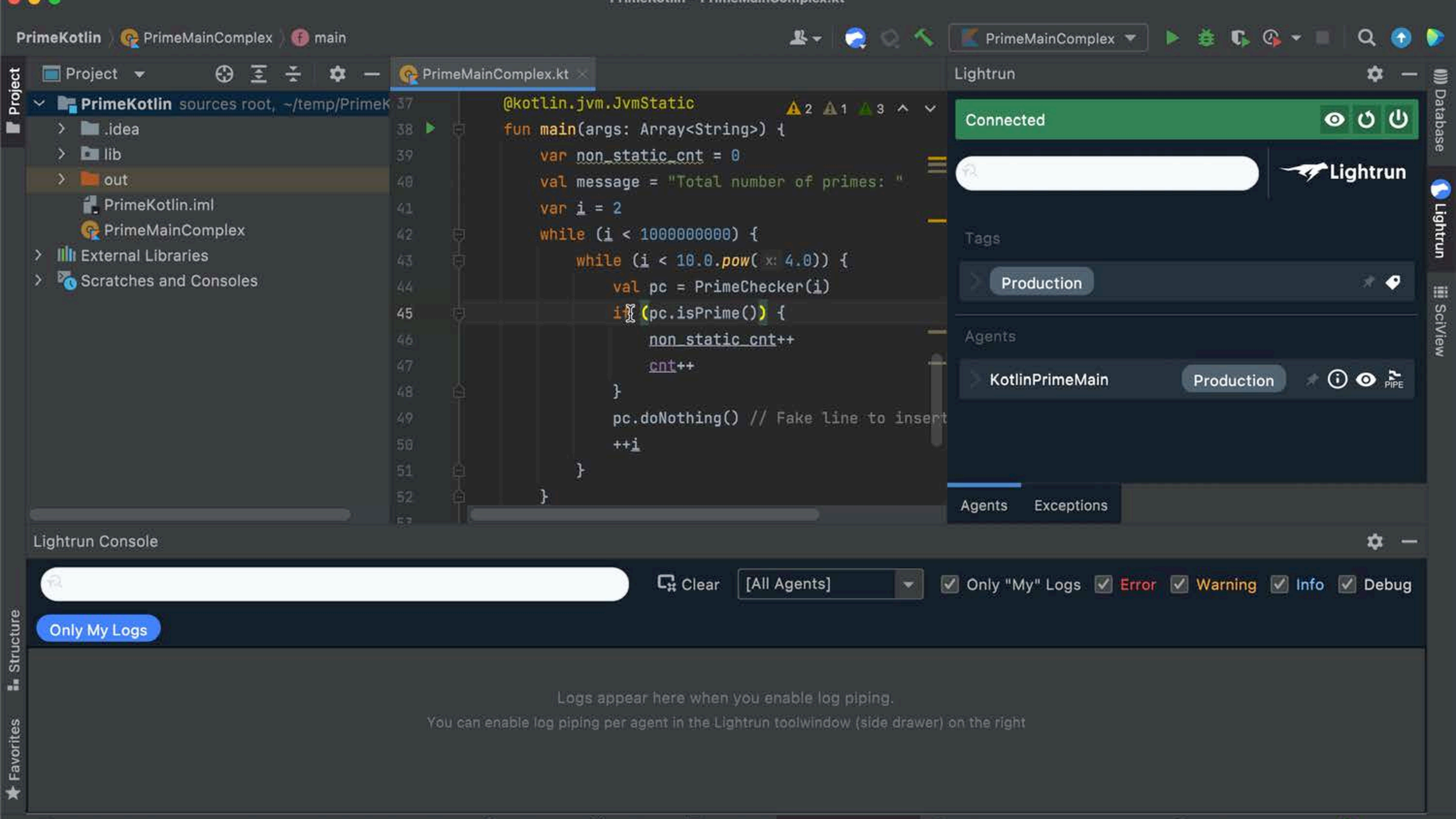
Logs appear here when you enable log piping.
You can enable log piping per agent in the Lightrun toolwindow (side drawer) on the right



```
@kotlin.jvm.JvmStatic
fun main(args: Array<String>) {
    var non_static_cnt = 0
    val message = "Total number of primes: "
    var i = 2
    while (i < 1000000000) {
        while (i < 10.0.pow( x: 4.0)) {
            val pc = PrimeChecker(i)
            if (pc.isPrime()) {
                non_static_cnt++
                cnt++
            }
            pc.doNothing() // Fake line to insert
            ++i
        }
    }
```

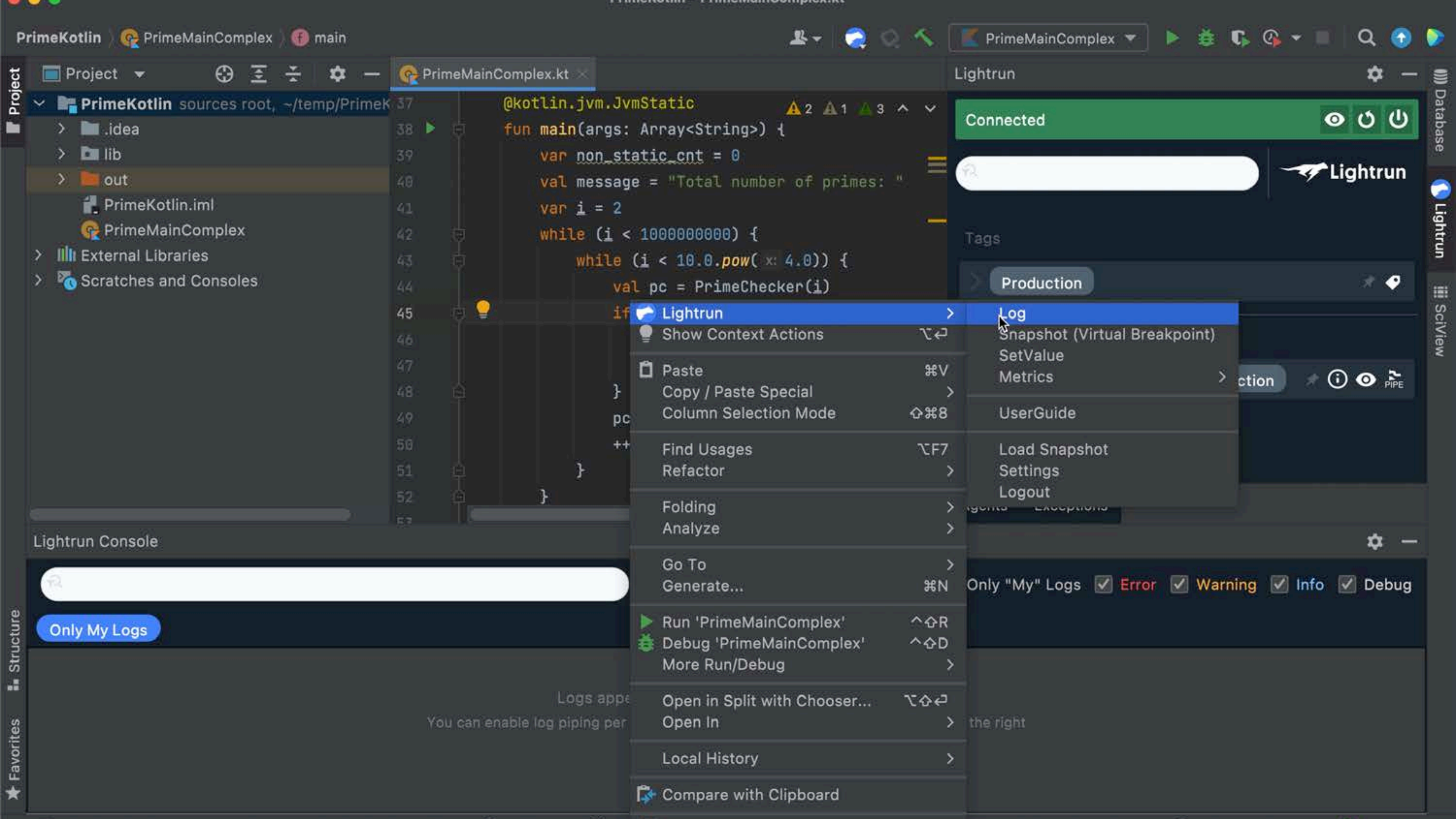
Logs appear here when you enable log piping.

You can enable log piping per agent in the Lightrun toolwindow (side drawer) on the right



```
    @kotlin.jvm.JvmStatic
    fun main(args: Array<String>) {
        var non_static_cnt = 0
        val message = "Total number of primes: "
        var i = 2
        while (i < 1000000000) {
            while (i < 10.0.pow( x: 4.0)) {
                val pc = PrimeChecker(i)
                if (pc.isPrime()) {
                    non_static_cnt++
                    cnt++
                }
                pc.doNothing() // Fake line to insert
                ++i
            }
        }
```

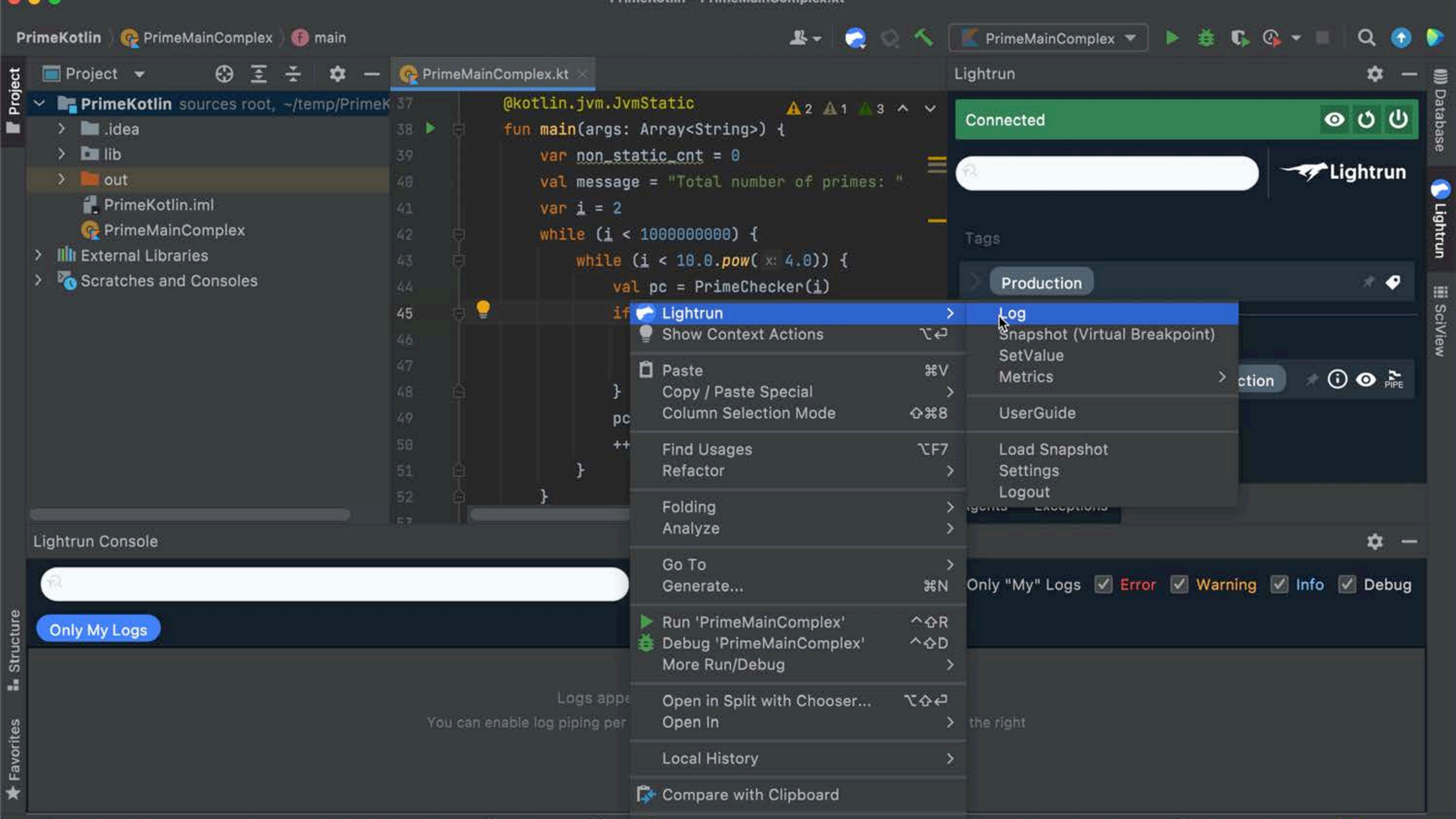
Logs appear here when you enable log piping.
You can enable log piping per agent in the Lightrun toolwindow (side drawer) on the right



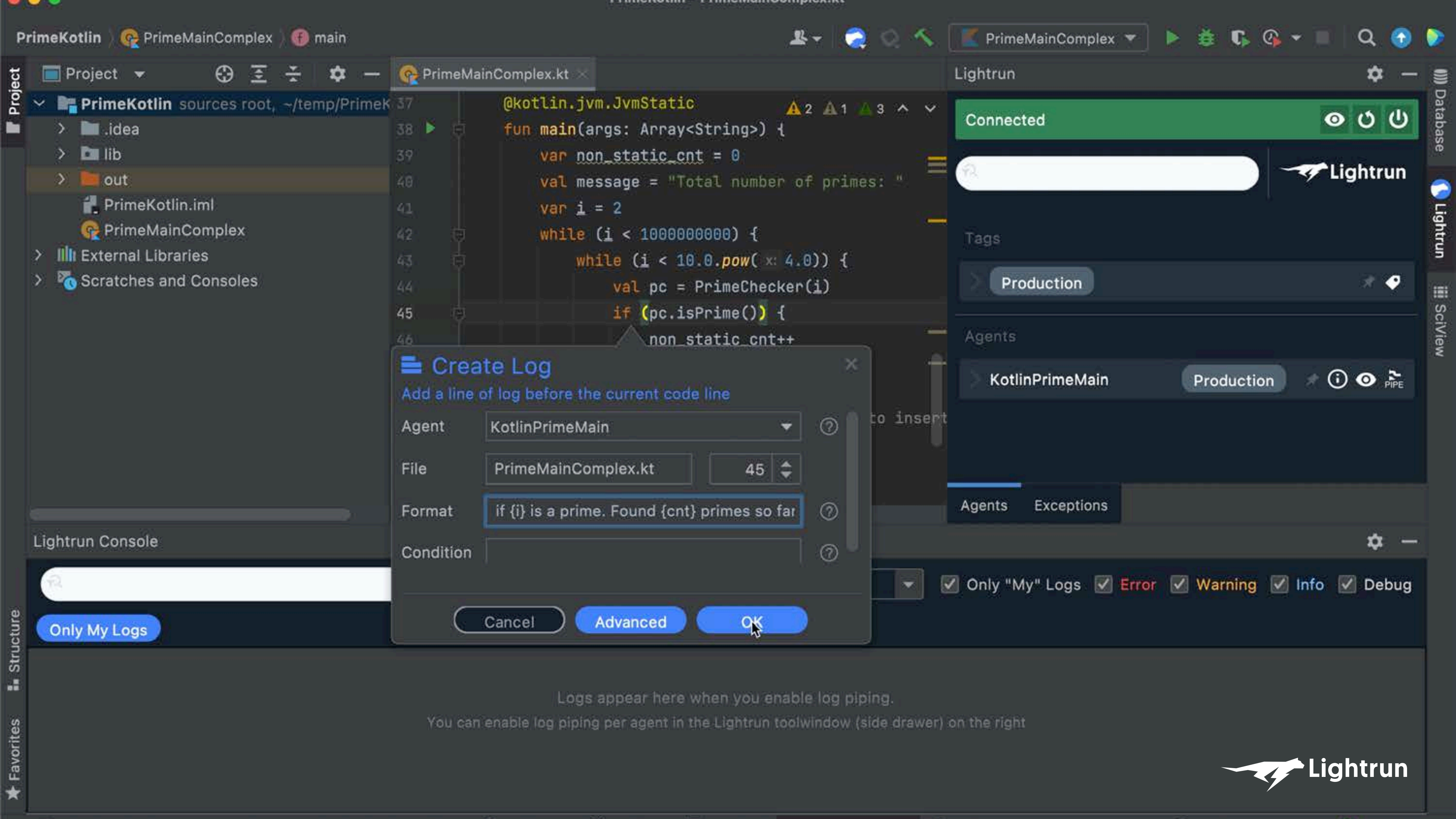
```
@kotlin.jvm.JvmStatic
fun main(args: Array<String>) {
    var non_static_cnt = 0
    val message = "Total number of primes: "
    var i = 2
    while (i < 1000000000) {
        while (i < 10.0.pow( x: 4.0)) {
            val pc = PrimeChecker(i)
            if


        }
        pc
        ++
    }
}
```

Logs appe

You can enable log piping per   the right



```
    @kotlin.jvm.JvmStatic
    fun main(args: Array<String>) {
        var non_static_cnt = 0
        val message = "Total number of primes: "
        var i = 2
        while (i < 1000000000) {
            while (i < 10.0.pow( x: 4.0)) {
                val pc = PrimeChecker(i)
                if

                }
                pc
                ++
            }
        }
```



```
@kotlin.jvm.JvmStatic
fun main(args: Array<String>) {
    var non_static_cnt = 0
    val message = "Total number of primes: "
    var i = 2
    while (i < 1000000000) {
        while (i < 10.0.pow( x: 4.0)) {
            val pc = PrimeChecker(i)
            if (pc.isPrime()) {
                non_static_cnt++
```

Create Log

Add a line of log before the current code line

Agent: KotlinPrimeMain

File: PrimeMainComplex.kt 45

Format: if {i} is a prime. Found {cnt} primes so far

Condition:

Cancel Advanced OK

Connected

Tags

Production

Agents

KotlinPrimeMain Production

Agents Exceptions

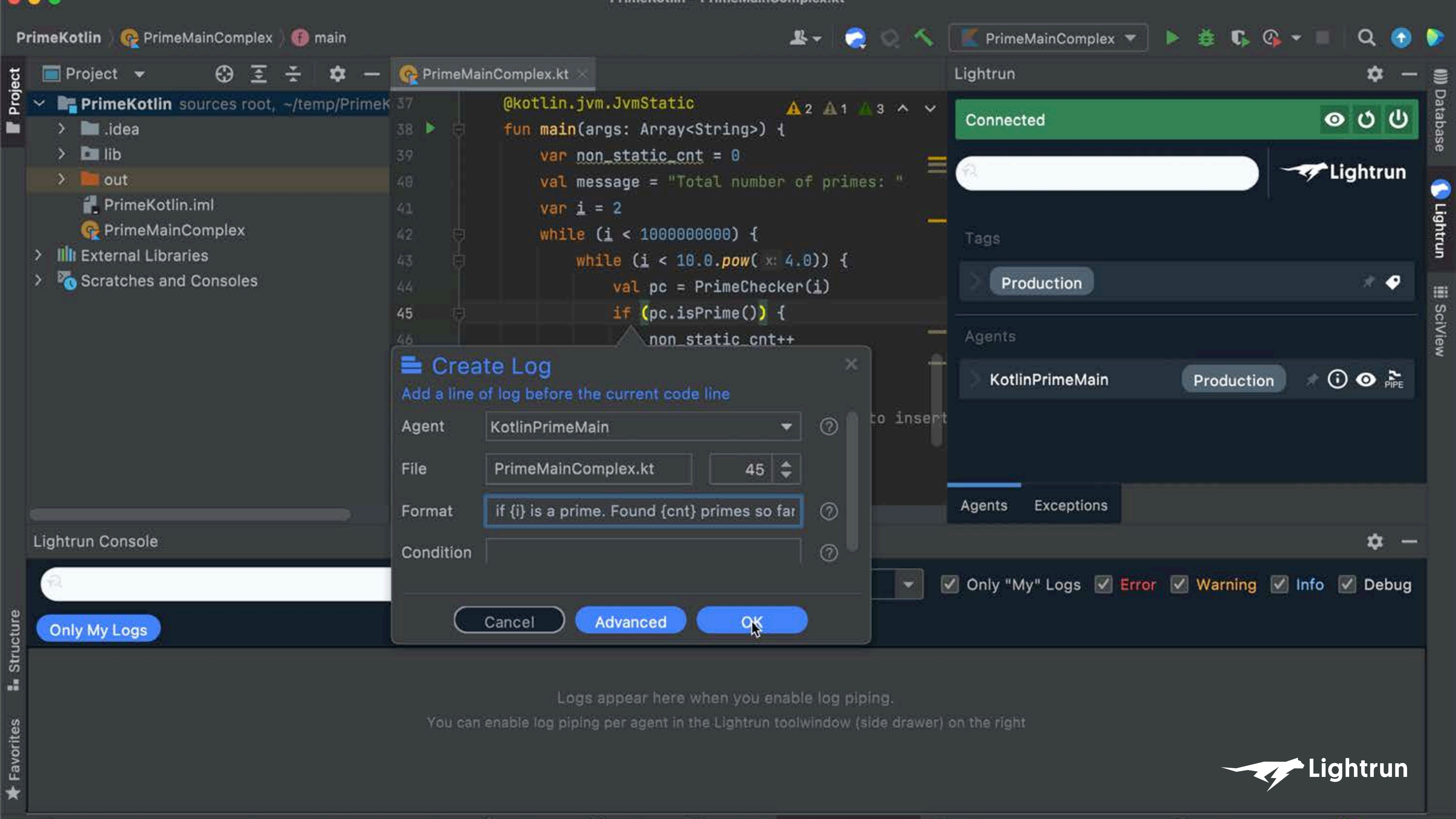
Lightrun Console

☑ Only "My" Logs ☑ Error ☑ Warning ☑ Info ☑ Debug

Only My Logs

Logs appear here when you enable log piping.
You can enable log piping per agent in the Lightrun toolwindow (side drawer) on the right

Lightrun



```
@kotlin.jvm.JvmStatic
fun main(args: Array<String>) {
    var non_static_cnt = 0
    val message = "Total number of primes: "
    var i = 2
    while (i < 1000000000) {
        while (i < 10.0.pow( x: 4.0)) {
            val pc = PrimeChecker(i)
            if (pc.isPrime()) {
                non_static_cnt++
```

to insert

Create Log

Add a line of log before the current code line

Agent: KotlinPrimeMain

File: PrimeMainComplex.kt 45

Format: if {i} is a prime. Found {cnt} primes so far

Condition:

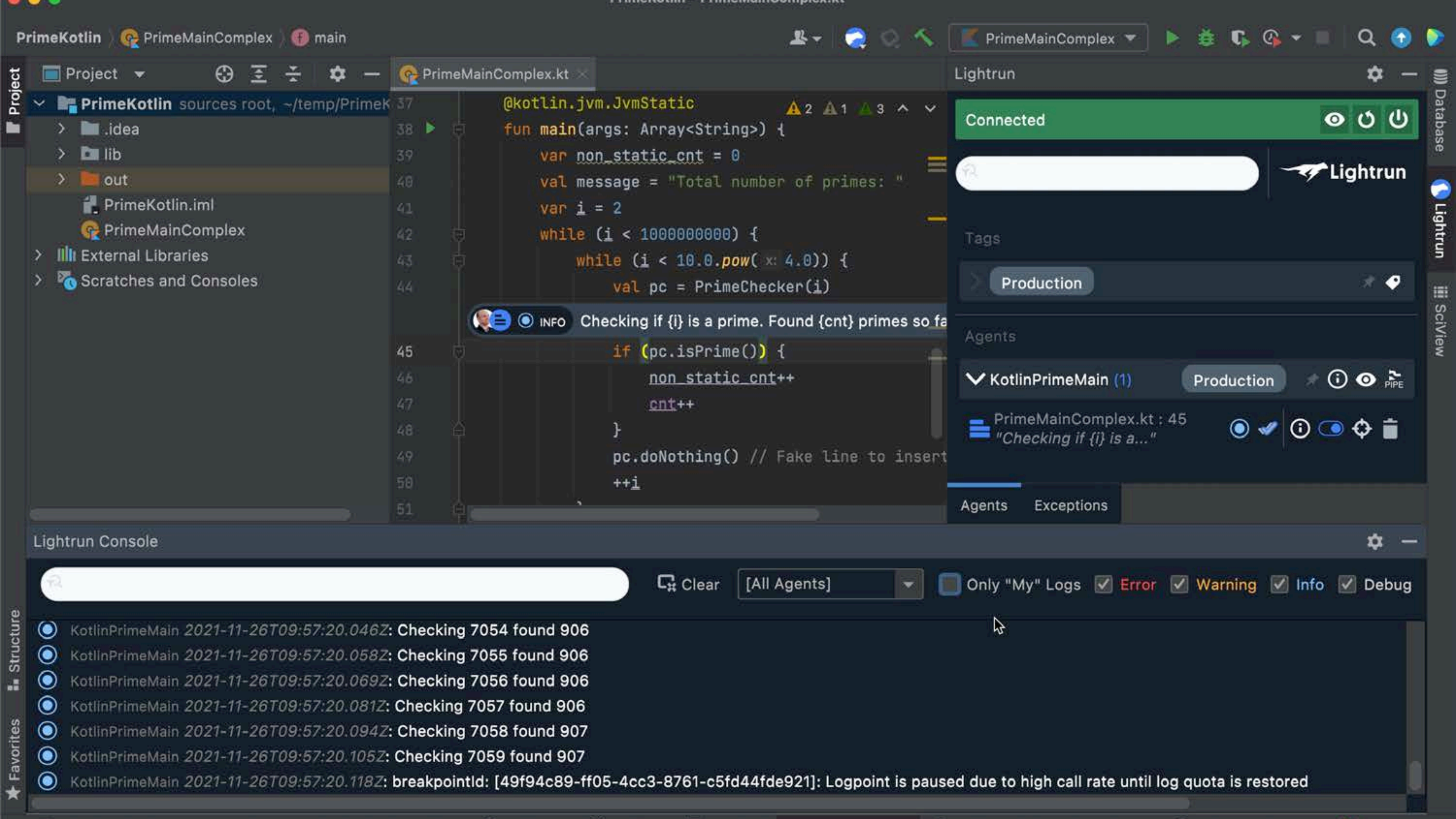
Connected

Tags: Production

Agents: KotlinPrimeMain Production

☑ Only "My" Logs ☑ Error ☑ Warning ☑ Info ☑ Debug

Logs appear here when you enable log piping.
You can enable log piping per agent in the Lightrun toolwindow (side drawer) on the right



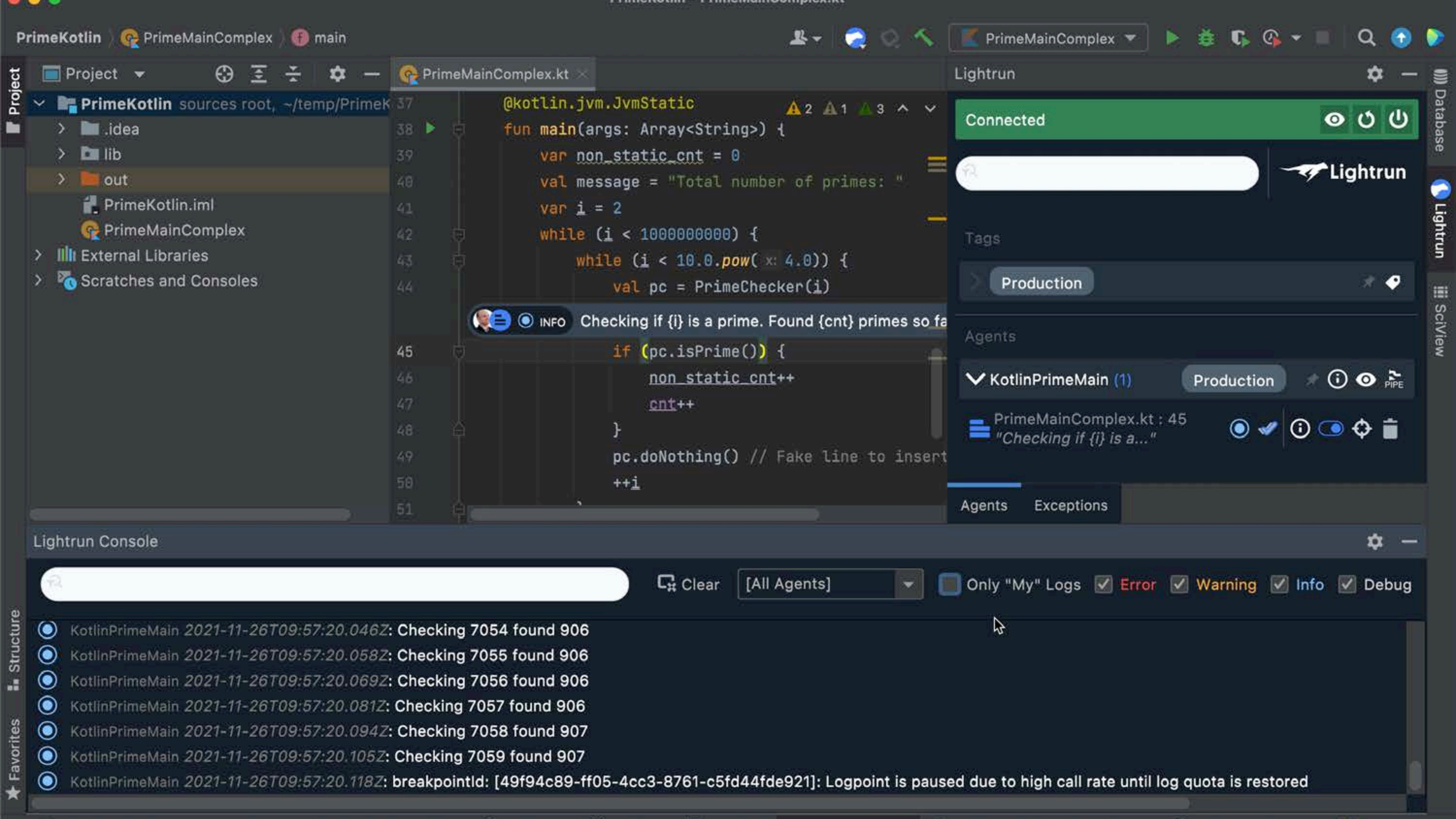
```
@kotlin.jvm.JvmStatic
fun main(args: Array<String>) {
    var non_static_cnt = 0
    val message = "Total number of primes: "
    var i = 2
    while (i < 1000000000) {
        while (i < 10.0.pow( x: 4.0)) {
            val pc = PrimeChecker(i)
            INFO  Checking if {i} is a prime. Found {cnt} primes so fa
            if (pc.isPrime()) {
                non_static_cnt++
                cnt++
            }
            pc.doNothing() // Fake line to insert
            ++i
```

PrimeMainComplex.kt : 45
"Checking if {i} is a..."

Lightrun Console

Clear [All Agents] □ Only "My" Logs ☑ Error ☑ Warning ☑ Info ☑ Debug

```
KotlinPrimeMain 2021-11-26T09:57:20.046Z: Checking 7054 found 906
KotlinPrimeMain 2021-11-26T09:57:20.058Z: Checking 7055 found 906
KotlinPrimeMain 2021-11-26T09:57:20.069Z: Checking 7056 found 906
KotlinPrimeMain 2021-11-26T09:57:20.081Z: Checking 7057 found 906
KotlinPrimeMain 2021-11-26T09:57:20.094Z: Checking 7058 found 907
KotlinPrimeMain 2021-11-26T09:57:20.105Z: Checking 7059 found 907
KotlinPrimeMain 2021-11-26T09:57:20.118Z: breakpointId: [49f94c89-ff05-4cc3-8761-c5fd44fde921]: Logpoint is paused due to high call rate until log quota is restored
```



```
@kotlin.jvm.JvmStatic
fun main(args: Array<String>) {
    var non_static_cnt = 0
    val message = "Total number of primes: "
    var i = 2
    while (i < 1000000000) {
        while (i < 10.0.pow( x: 4.0)) {
            val pc = PrimeChecker(i)
    INFO Checking if {i} is a prime. Found {cnt} primes so fa
            if (pc.isPrime()) {
                non_static_cnt++
                cnt++
            }
            pc.doNothing() // Fake line to insert
            ++i
```

KotlinPrimeMain (1) Production

PrimeMainComplex.kt : 45
"Checking if {i} is a..."

```
KotlinPrimeMain 2021-11-26T09:57:20.046Z: Checking 7054 found 906
KotlinPrimeMain 2021-11-26T09:57:20.058Z: Checking 7055 found 906
KotlinPrimeMain 2021-11-26T09:57:20.069Z: Checking 7056 found 906
KotlinPrimeMain 2021-11-26T09:57:20.081Z: Checking 7057 found 906
KotlinPrimeMain 2021-11-26T09:57:20.094Z: Checking 7058 found 907
KotlinPrimeMain 2021-11-26T09:57:20.105Z: Checking 7059 found 907
KotlinPrimeMain 2021-11-26T09:57:20.118Z: breakpointId: [49f94c89-ff05-4cc3-8761-c5fd44fde921]: Logpoint is paused due to high call rate until log quota is restored
```

Summary

Summary

JVM

Summary

JVM

node JS

Summary

JVM node js python™

Summary

JVM node python™

Stability
Inserted Actions are emulated in a dedicated Sandbox to validate there are no side effects of the original flow and state of the process

Summary

JVM node js python™

Stability
Inserted Actions are emulated in a dedicated Sandbox to validate there are no side effects of the original flow and state of the process

Security
Authorization and authentication, integration with common IDPs

ISO 27001

Summary

JVM node JS python™

Stability
Inserted Actions are emulated in a dedicated Sandbox to validate there are no side effects of the original flow and state of the process

Security
Authorization and authentication, integration with common IDPs

ISO 27001

Privacy
PII redaction and blacklisting of files / methods / members

Summary

JVM node JS python™

Stability
Inserted Actions are emulated in a dedicated Sandbox to validate there are no side effects of the original flow and state of the process

Security
Authorization and authentication, integration with common IDPs

ISO 27001

Privacy
PII redaction and blacklisting of files / methods / members

Footprint
CPU footprint is negligible.
Memory and network footprints are capped and configurable

Summary

JVM node JS python™

Stability
Inserted Actions are emulated in a dedicated Sandbox to validate there are no side effects of the original flow and state of the process

Security
Authorization and authentication, integration with common IDPs

ISO 27001

Privacy
PII redaction and blacklisting of files / methods / members

Footprint
CPU footprint is negligible.
Memory and network footprints are capped and configurable

Environment agnostic
Operates on-prem / cloud, microservices, serverless

aws

Lightrun

Lightrun

Questions?

Shai Almog

Developer Advocate @ Lightrun

twitter.com/debugagent

talktotheduck.dev

github.com/shai-almog

shaia@lightrun.com