Application Performance Monitoring

Throughout this lab, each section will be broken down into a series of steps. To navigate between sections, click each header to expand or collapse the sections.

Make sure you are logged into Datadog using the Datadog training account credentials provisioned for you. You can find that information by running `creds` in the lab terminal.

About Tracing

When an application is configured correctly to send traces to the Datadog Agent, Datadog can associate those traces with everything it knows about your infrastructure. You can fluidly navigate across traces, logs, processes, metrics, and events to get a complete picture of what was happening at any point in time.

Application tracing is enabled by using Datadog's APM libraries in application code. The libraries must be configured to know where and how to send the traces they collect. In a containerized environment, this can be done by setting environment variables that the libraries will look for to configure themselves.

Storedog's code has already been instrumented and configured using a `docker-compose.yml` file. Examine the file to see how that was done.

Examine the Configured Service

First, focus on the `discounts` service, which is a Python Flask application that connects to the PostgreSQL database. The `frontend` service makes HTTP requests to this service for discount code information.

1. In the IDE, open `docker-compose.yml`.
2. Look at the `discounts` service to see how the service is configured for APM:

```
33    discounts:
34      environment:
35        - FLASK_APP=discounts.py
36        - FLASK_DEBUG=1
37        - POSTGRES_PASSWORD
38        - POSTGRES_USER
39        - POSTGRES_HOST=db
40        - DD_SERVICE=discounts-service
41        - DD_ENV=dd101-sre
42        - DD_LOGS_INJECTION=true
43        - DD_TRACE_SAMPLE_RATE=1
44        - DD_SERVICE_MAPPING=postgres:database
45        - DD_PROFILING_ENABLED=true
46        - DD_AGENT_HOST=datadog
47      image: 'public.ecr.aws/x2b9z2t7/ddtraining/discounts-fixed:2.2.0'
48      command:
49        [
50          sh,
51          -c,
52          'ddtrace-run flask run --port=5001 --host=0.0.0.0',
53        ]
54      ports:
55        - '5001:5001'
```

Notice `DD_AGENT_HOST`, which tells the `ddtrace` library to send traces to the `datadog` service. You'll learn more about `ddtrace` later in the lab.

Also notice `DD_SERVICE_MAPPING` on line 44. This tells `ddtrace` to use the service name `database` instead of the default `postgres`, which it uses by default when it detects a PostgreSQL connection.

The `command` line overrides the container's default command, adding `ddtrace-run` to the command that starts the application.

3. Confirm that the discounts service application is sending traces to APM.

   In the lab terminal, run the Agent `status` command:

   ```
   docker-compose exec datadog agent status
   ```

4. Scroll to the **APM Agent** section. It is receiving Python traces from the discounts service:

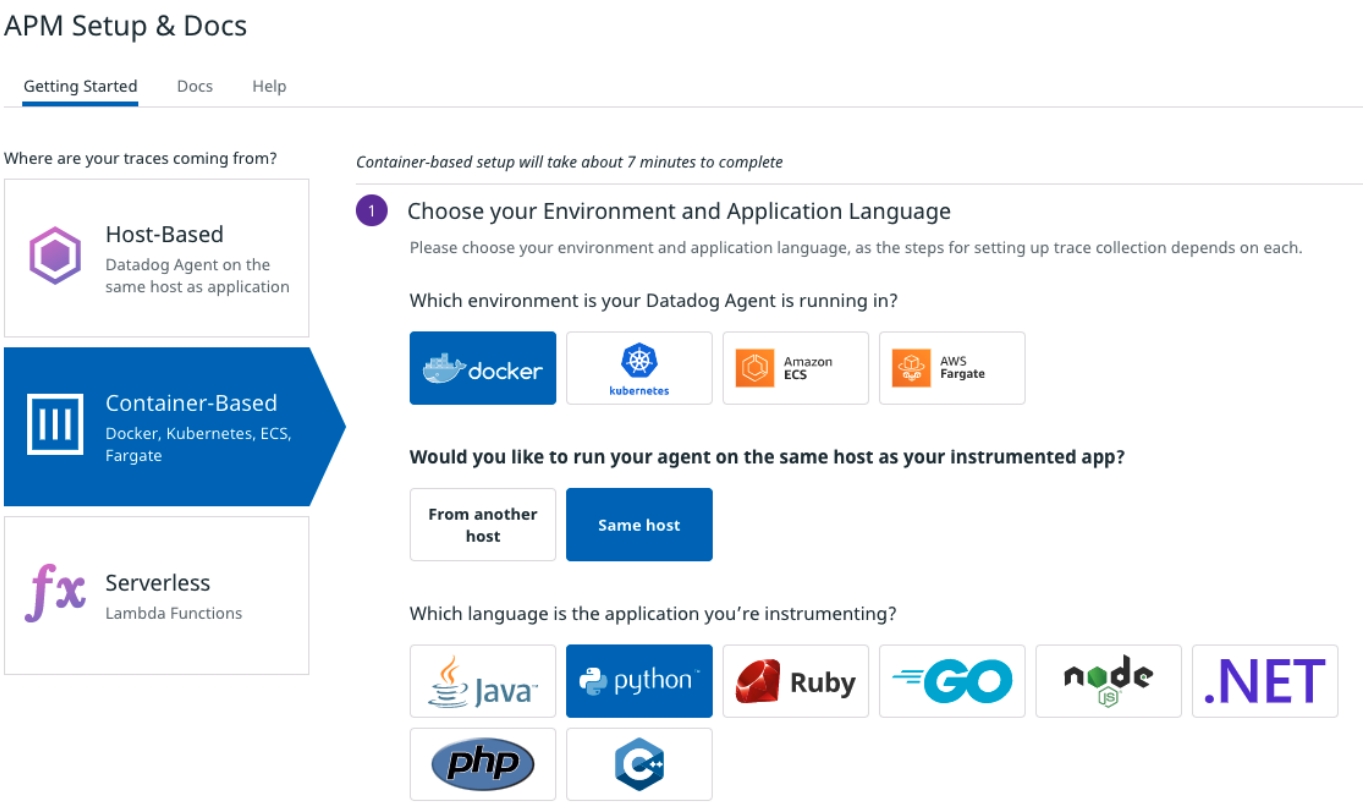
```
=========
APM Agent
=========
  Status: Running
  Pid: 376
  Uptime: 175 seconds
  Mem alloc: 13,535,224 bytes
  Hostname: dd101-sre-host
  Receiver: 0.0.0.0:8126
  Endpoints:
    https://trace.agent.datadoghq.com

  Receiver (previous minute)
  ==========================
    From python 3.9.6 (CPython), client 0.57.3
      Traces received: 9 (1,858,750 bytes)
      Spans received: 3789

    Default priority sampling rate: 100.0%
    Priority sampling rate for 'service:discounts-service,env:dd101-sre': 100.0%
```

Even though this service was already configured for APM for this lab, it's important to know where to find instructions on how to do it. Therefore, in the next section, you'll learn how Datadog can help you build this configuration if you were to do it yourself.

Explore the APM Wizard

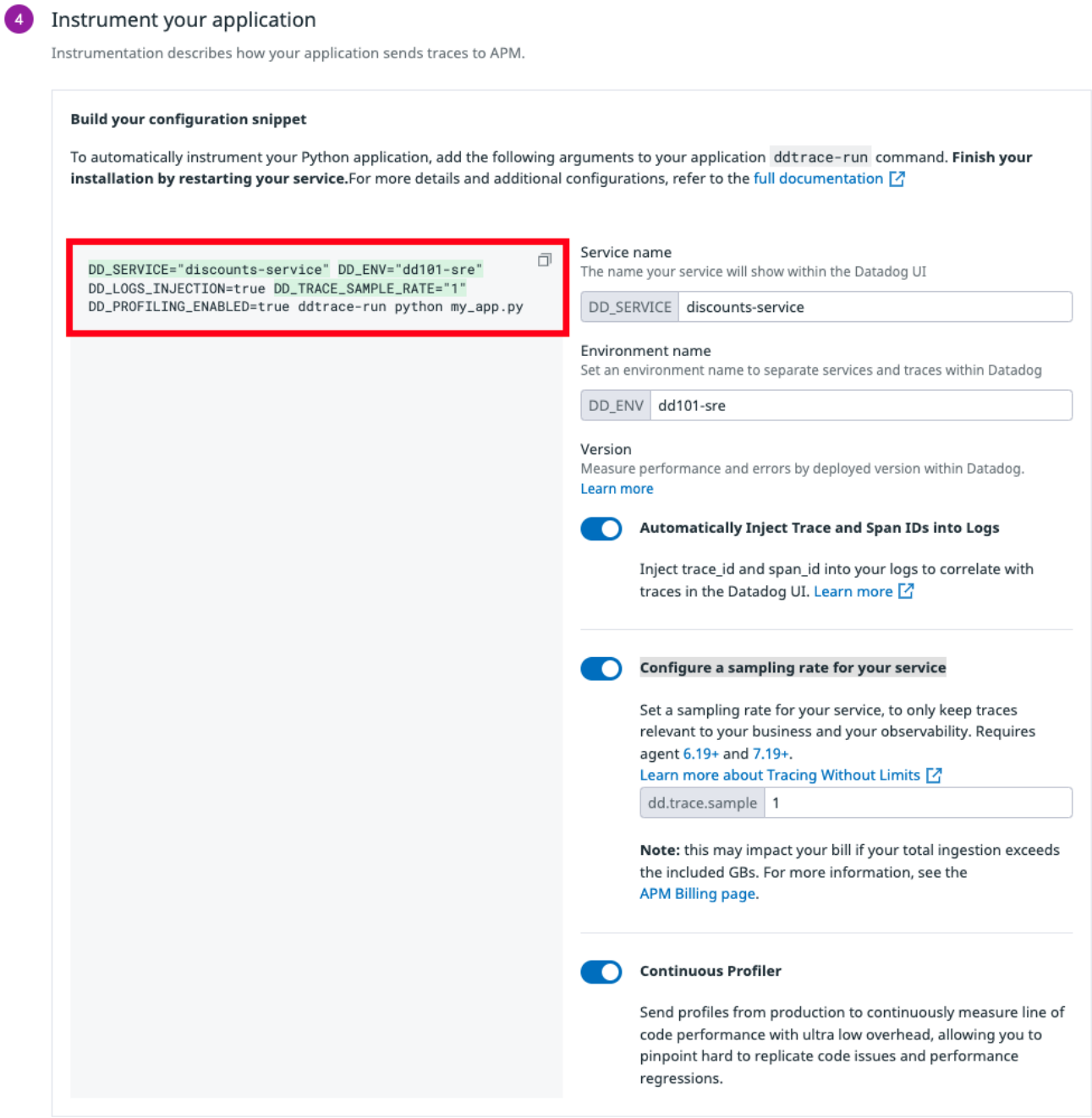
Datadog provides a handy wizard to walk you through configuring APM for a variety of languages and environment. You just saw how the Python-based discounts service is configured in the `docker-compose.yml` file. In this section, you'll see the APM wizard that is used to build that configuration.

1. Log in to Datadog using the trial credentials the lab created for you. You can run `creds` in the lab terminal whenever you need to retrieve your Datadog training account credentials.
2. Navigate to **APM > Setup & Configuration** and click on the **Service Setup** tab.
3. On the **APM Setup & Docs** page, in the left-hand column, click on **Container-Based**.
4. Under **Choose your Environment and Application Language**, click on **Docker**, then **Same host**, then **Python**.



The rest of the page is then updated to walk you through instrumenting your application based on the selections you made:

- Under **Run the Agent**, you are told how to configure the Agent container at runtime to accept traces from applications. In this lab, the Agent container is already configured similarly in the `docker-compose.yml` file.
- The **Install the Python client** step tells you the command to add the `ddtrace` library to a Python application. This is typically done by application developers, and has already been done for the discounts and advertisements services in this lab.
- The **Instrument your application** step helps you build the command for running a Python script with `ddtrace-run`. `ddtrace` relies on environment variables to know where and how to send traces to the Datadog Agent. To see how the APM configuration for this lab's discounts service was configured, do the following:
  1. In the form, set **DD_SERVICE** to `discounts-service`
  2. Set **DD_ENV** to `dd101-sre`
  3. Enable **Configure a sampling rate for your service** and leave the default value for **dd.trace.sample**
  4. Enable **Continuous Profiler**
  5. Confirm that your configuration snippet looks like this:



6. Back in the IDE, compare that configuration snippet with what's in `docker-compose.yml`:

```yaml
33    discounts:
34      environment:
35        - FLASK_APP=discounts.py
36        - FLASK_DEBUG=1
37        - POSTGRES_PASSWORD
38        - POSTGRES_USER
39        - POSTGRES_HOST=db
40        - DD_SERVICE=discounts-service
41        - DD_ENV=dd101-sre
42        - DD_LOGS_INJECTION=true
43        - DD_TRACE_SAMPLE_RATE=1
44        - DD_SERVICE_MAPPING=postgres:database
45        - DD_PROFILING_ENABLED=true
46        - DD_AGENT_HOST=datadog
47      image: 'public.ecr.aws/x2b9z2t7/ddtraining/discounts-fixed:2.2.0'
48      command:
49        [
50          sh,
51          -c,
52          'ddtrace-run flask run --port=5001 --host=0.0.0.0',
53        ]
54      ports:
55        - '5001:5001'
```

While the formatting is slightly different, these settings are the same ones that are set in the `docker-compose.yml` file.

Now that you've examined how the Storedog app has been configured for APM, in the next section, you'll see what these traces look like in Datadog.

Explore Traces in the Datadog App

Now that the Agent is collecting traces from the discounts service, take a look at those traces in Datadog.

It can take several minutes for the Datadog App to process traces when they first start coming in. You can preview the following steps until they do.

1. Navigate to **APM > Service Catalog**. If you have taken other courses within the past two weeks, you likely have a **Show data from** selector with multiple options. Make sure that `env:dd101-sre` is selected.
2. You'll see the `discounts-service` that you just enabled. You'll also see `database`, which doesn't send traces to the Datadog Agent directly.

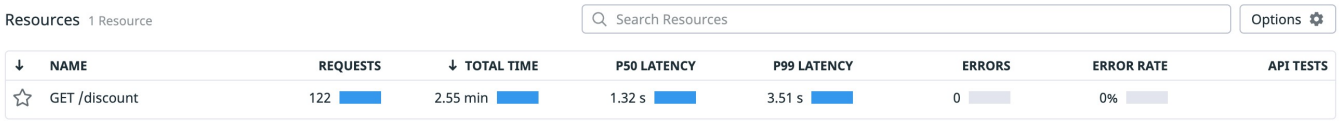
Show data from: env:dd101-sre

TYPE	SERVICE	LAST DEPLOY	REQUESTS	P95 LATENCY	ERROR RATE	INFRASTRU...	DASHBO...	MONITORS
	database		0.6 req/s	22.5 ms	0%	2		
	discounts-service		< 0.1 req/s	4.46 s	0%	2		

`database` shows up because `discounts-service` traces capture it. You configured the PostgreSQL integration in the previous lab, but it doesn't send *traces* to the Datadog Agent. Applications that *connect* to the database do.

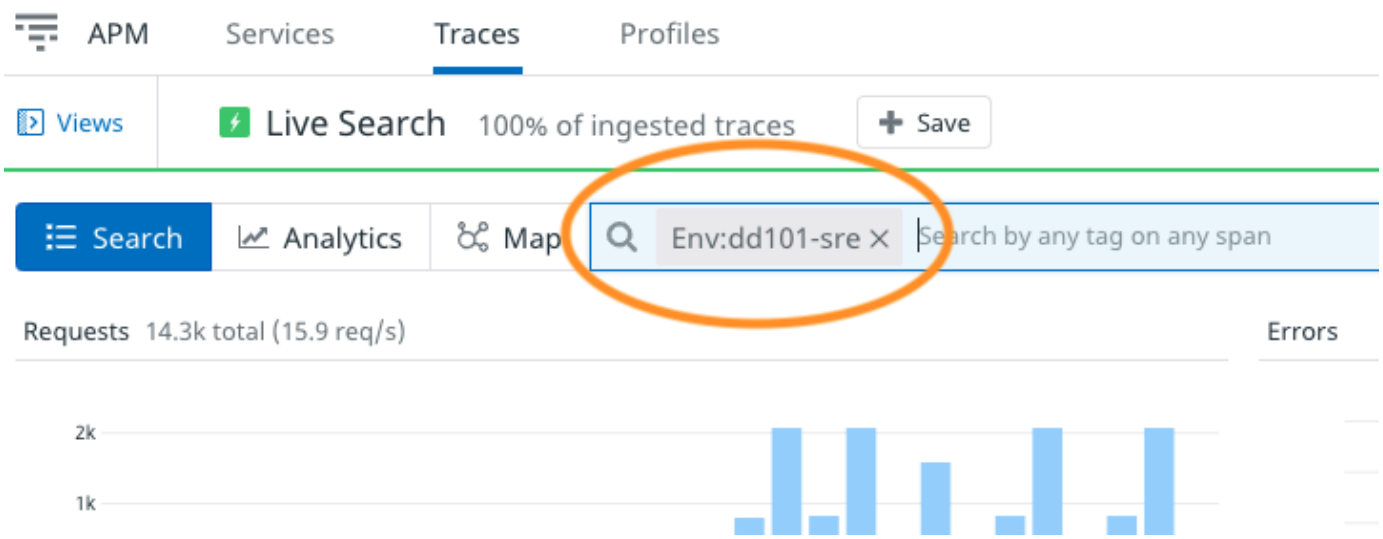
Note: You might see the old `postgres` service tags floating around from the previous labs, but after some time, they should be replaced with `database`.

3. Click on **discounts-service** and scroll down to **Resources**. Here you will see all of the service's application endpoints that APM traced. This service has one endpoint, `GET /discount`

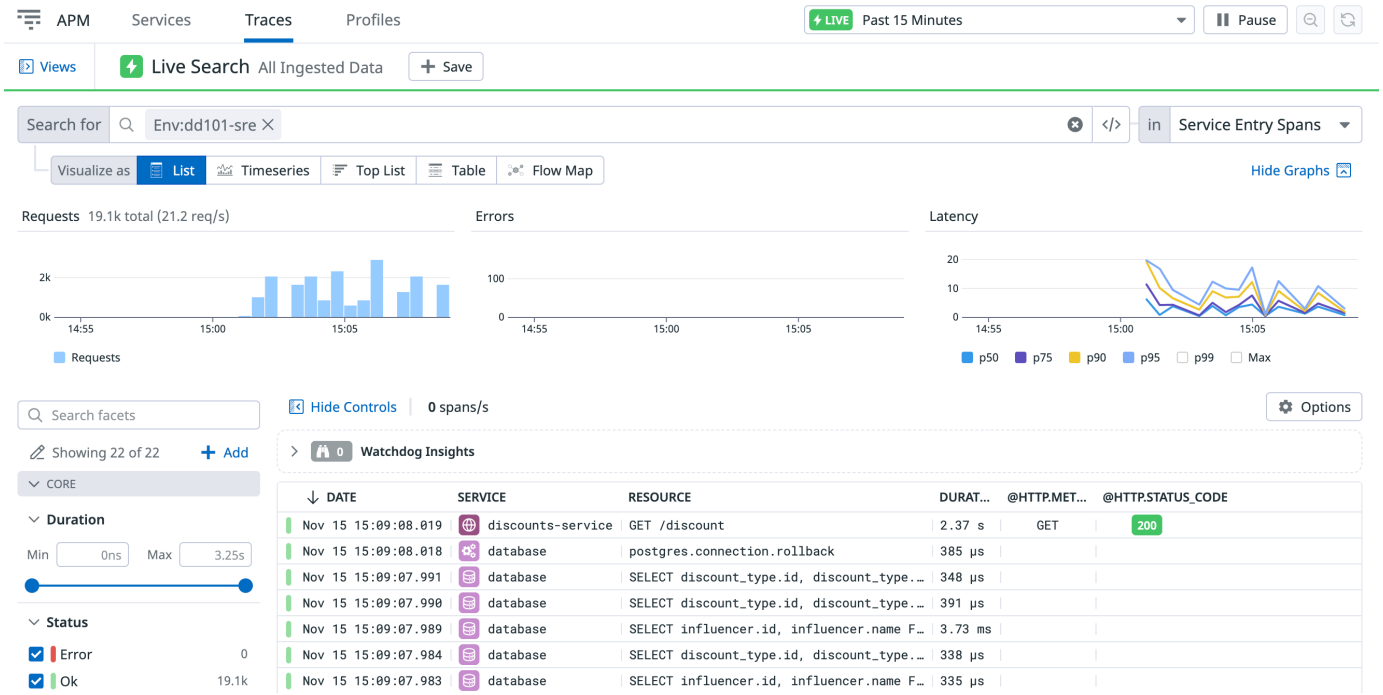


Note: If you don't see the `GET /discount` endpoint, scroll to the top of the page and check if the `operation` is set to `flask.request`.

4. Navigate to **APM** > **Traces**. If the search field contains something other than `Env:dd101-sre`, clear it and enter `Env:dd101-sre`:

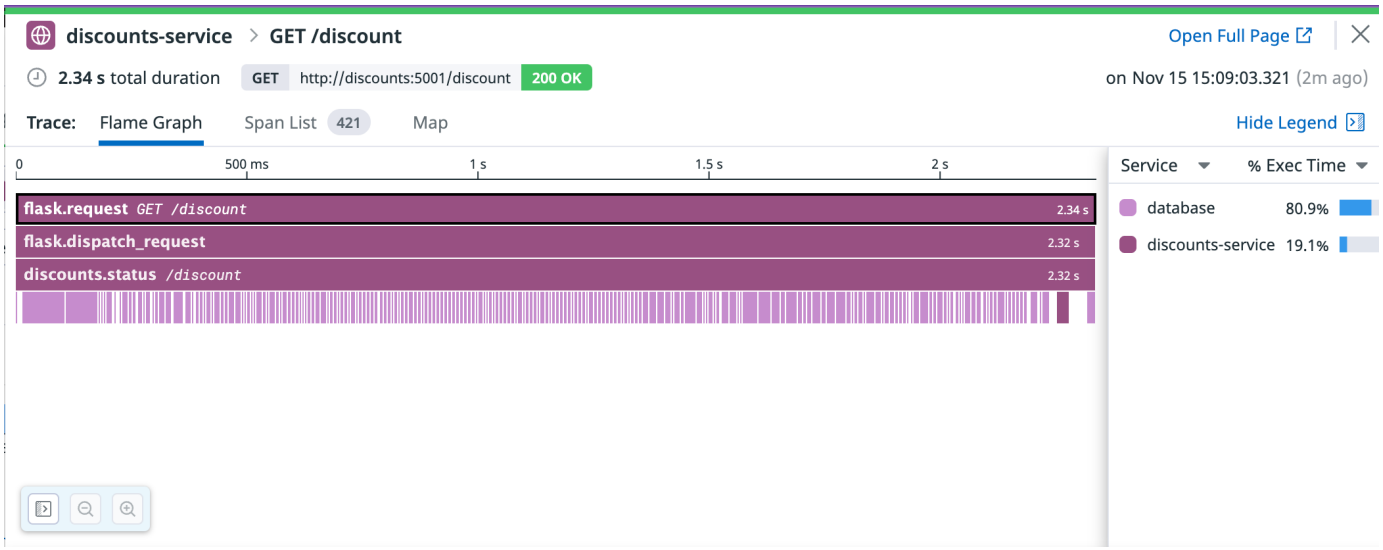


Here you see a live stream of the traces APM has captured over the past 15 minutes.



5. To see only the `discounts-service` traces, in the **Service** section of the left-hand pane, click on the `discounts-service` facet. This will filter all traces tagged with `service:discounts-service`.

6. Click on a `discounts-service` trace to open the trace details side panel. The flame graph displays the time spent in each service for this trace:



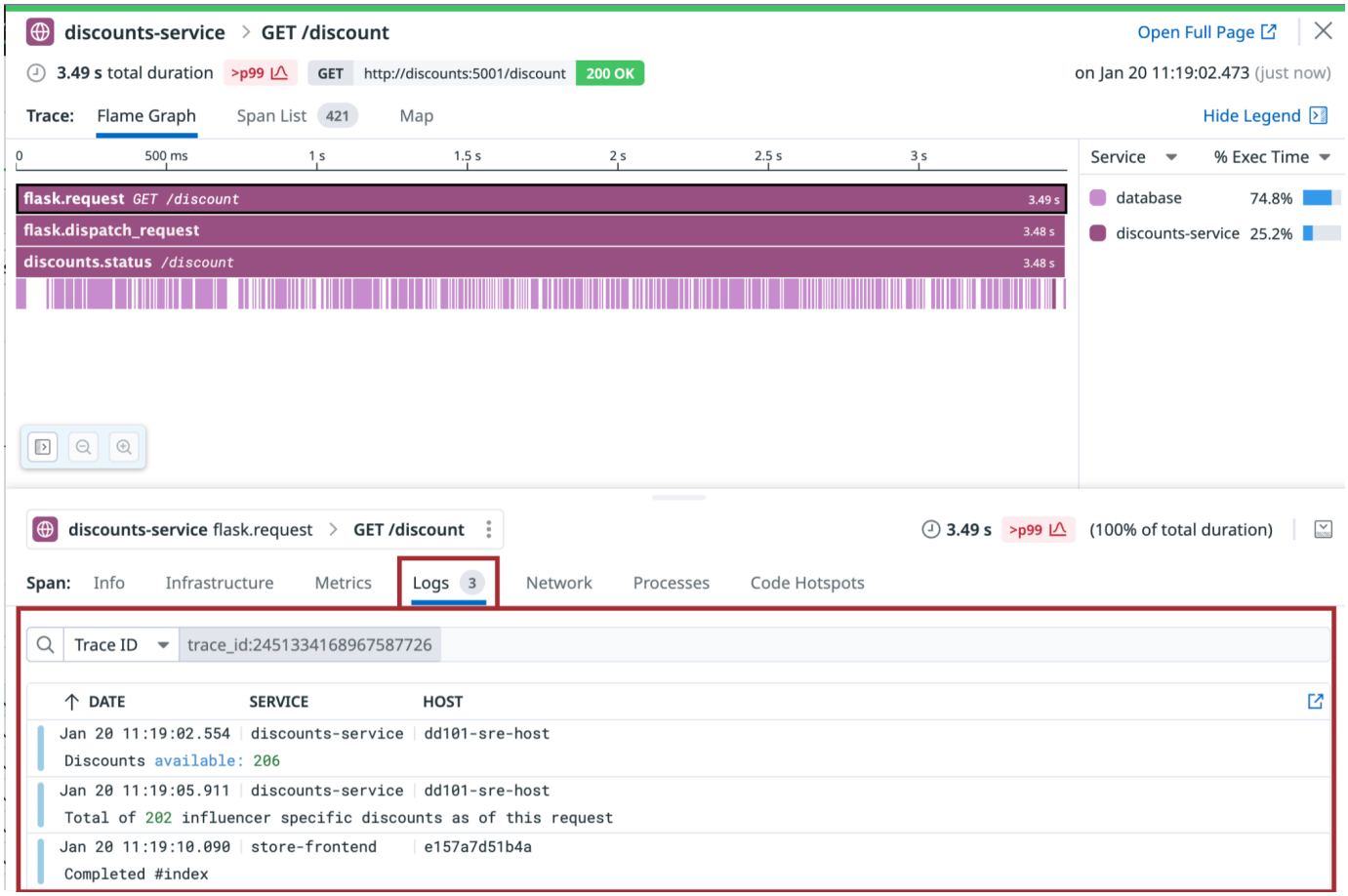
Traverse APM Traces to Log Entries

Take a moment to examine the logs for the traces.

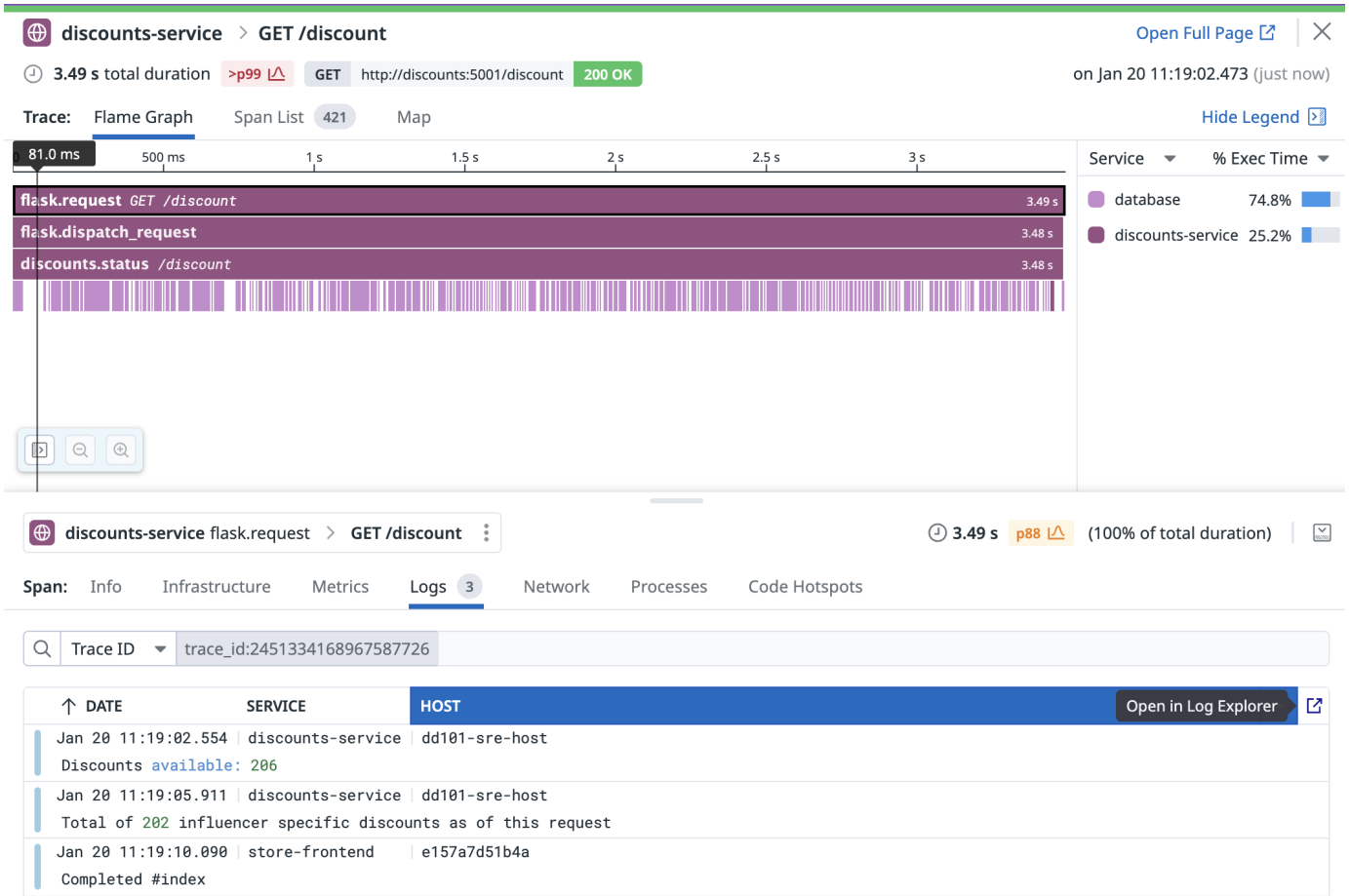
1. Click the **Logs** tab at the bottom of the trace details panel. You can resize the logs display area by dragging the horizontal divider at the top.

   > **Note**: If you see a message like, "Get started with Log Management...", click the link to enable logging. Then return to **APM > Traces** and open a `discounts-service` trace again. Click on the **Logs** tab.

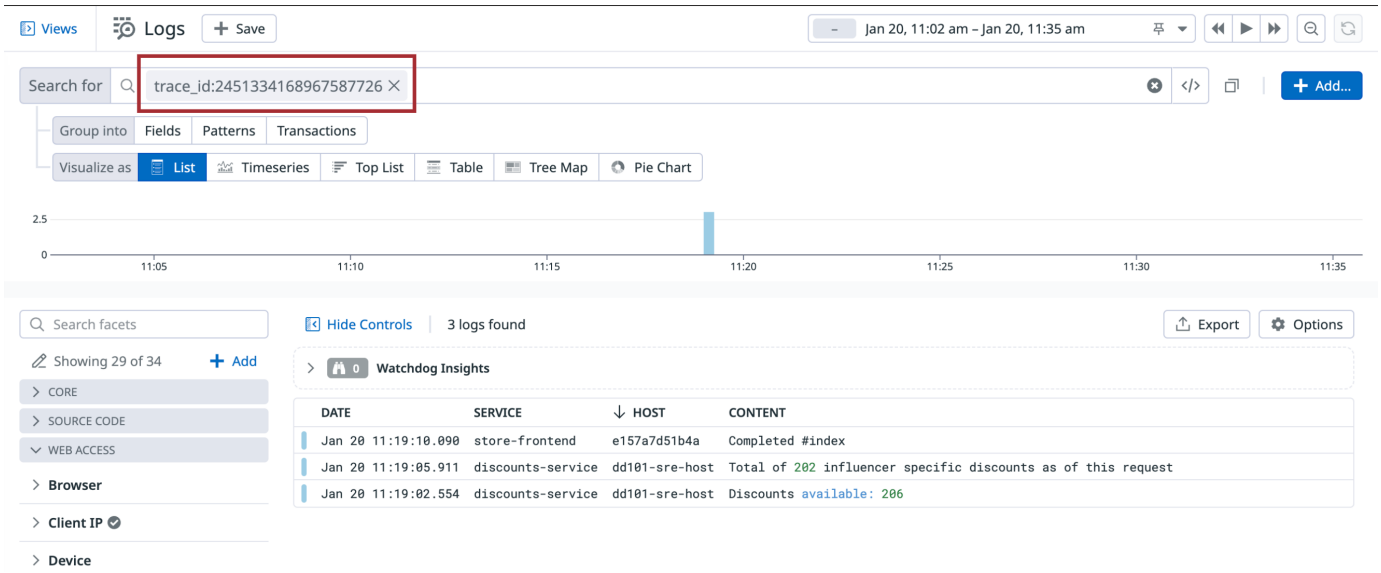
   These are the related log lines captured during the trace's timeframe:



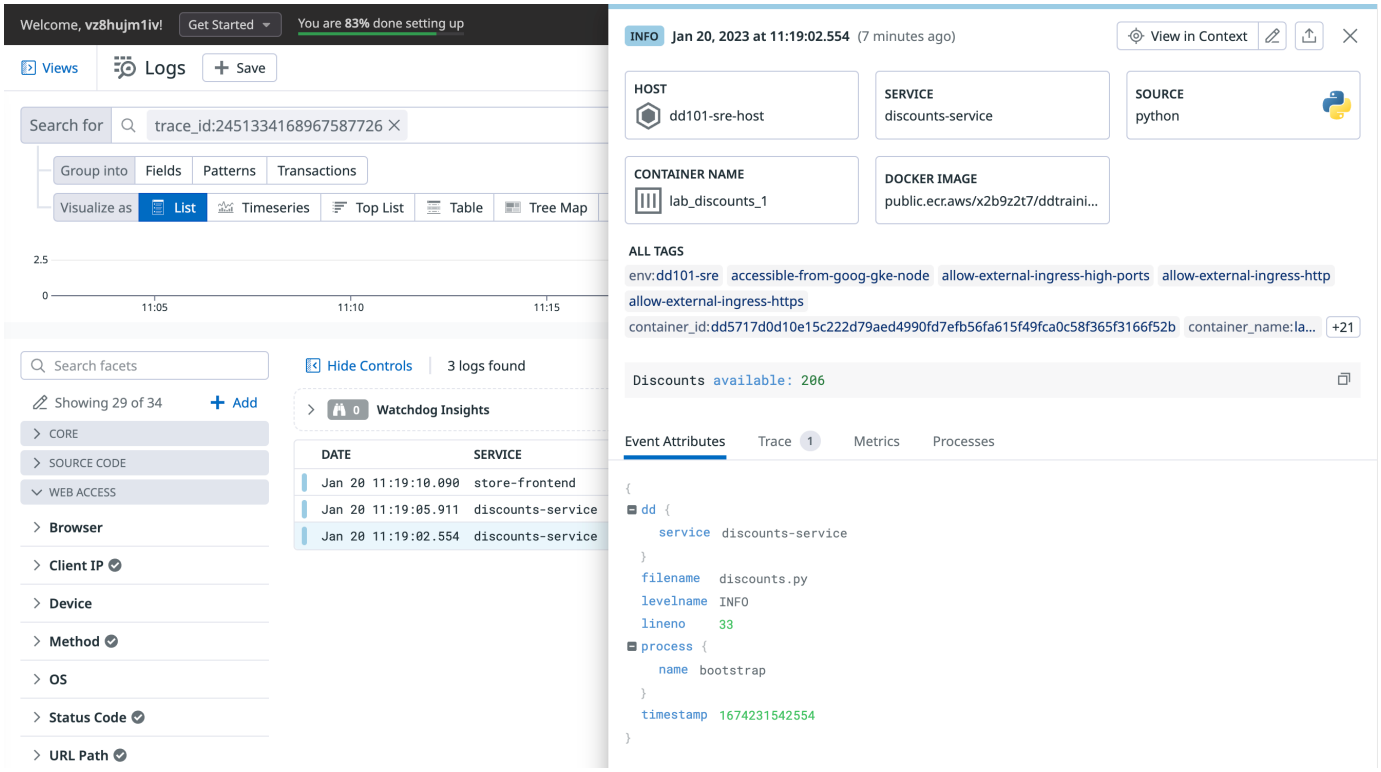
2. Mouse over each entry and look at the flame graph. You'll see a vertical line marking the exact point in the trace that the log line was emitted. This is enabled by the `DD_LOGS_INJECTION` configuration option.
3. In the Logs table, click the icon for **Open in Log Explorer** to the far end of the **Hosts** column, as shown in the following image:



This will open a new tab to the Logs Explorer page with the associated logs for the trace you just viewed based on the `trace_id`.



4. Click on one of the log entries for `discounts-service` to open the log entry details panel.



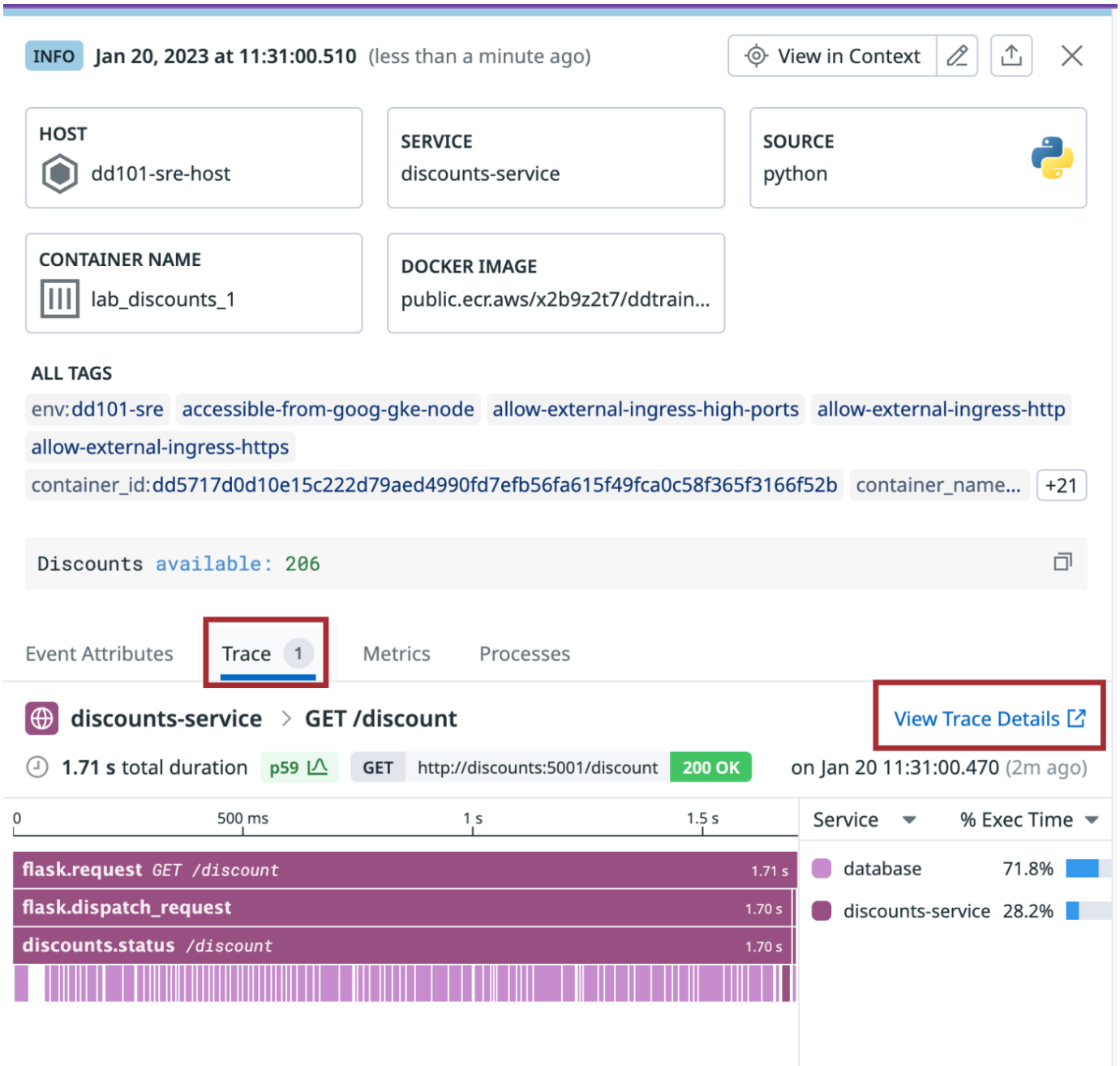
When the `ddtrace` library finds the environment variable `DD_LOGS_INJECTION=true`, it automatically injects tracing data into the log lines and formats the output as JSON. You can learn more about `ddtrace` for Python in the Datadog Python APM Client documentation.

Traverse Log Entries to APM Traces

Now that you know how to navigate from an APM trace to its associated logs, try to navigate from a log line to its associated APM trace:

1. Back on the Log Explorer page, clear the search field and change the timeframe dropdown in the upper-right corner to `Past 15 Minutes` to view all of the most recent logs.
2. In the **Service** section of the left-hand pane, click on the `discounts-service` facet to see only the logs from the `discounts-service`.
3. Click on a `discounts-service` log line that states `Discounts available...`
4. In the log entry details panel, click on the **Trace** tab.
5. Here is the trace for this log line, right in the log details view!

   Click on **View Trace Details** to view the trace in the APM trace details page.



You can now travel back and forth between APM traces and logs for the discounts service. In the next section, you'll gather traces from the remaining Storedog services.

Trace All the Services

In the previous step, you learned how to trace Storedog's discounts service. In this step, you're going to trace Storedog's other services. These include the advertisements service, which is also a Python Flask application, and the store frontend, which is a Ruby Spree application.

You can enable the advertisements service exactly as you did the discounts service.

Datadog maintains a `ddtrace` client for Ruby, which the store frontend application already uses. There is no equivalent to the Python client's `ddtrace-run` in the Ruby client, so you don't need to change the `command` in `docker-compose.yml`. You only need to add the environment variables.

For your convenience, there is a `docker-compose.yml` file that is already updated.

1. In the lab terminal, copy the updated version over the current `docker-compose.yml` file by running the following command:

```
cp /root/docker-compose-complete.yml /root/lab/docker-compose.yml
```

2. In the IDE, check the `docker-compose.yml` file to look at the newly added environment variables for the `advertisements` and `frontend` services. You may need to reopen the file to see the changes.

3. Return to the terminal and restart the application by running the following command:

```
docker-compose down && docker-compose up -d
```

4. To confirm that the Agent is picking up new traces, run the Agent `status` command:

```
docker-compose exec datadog agent status
```

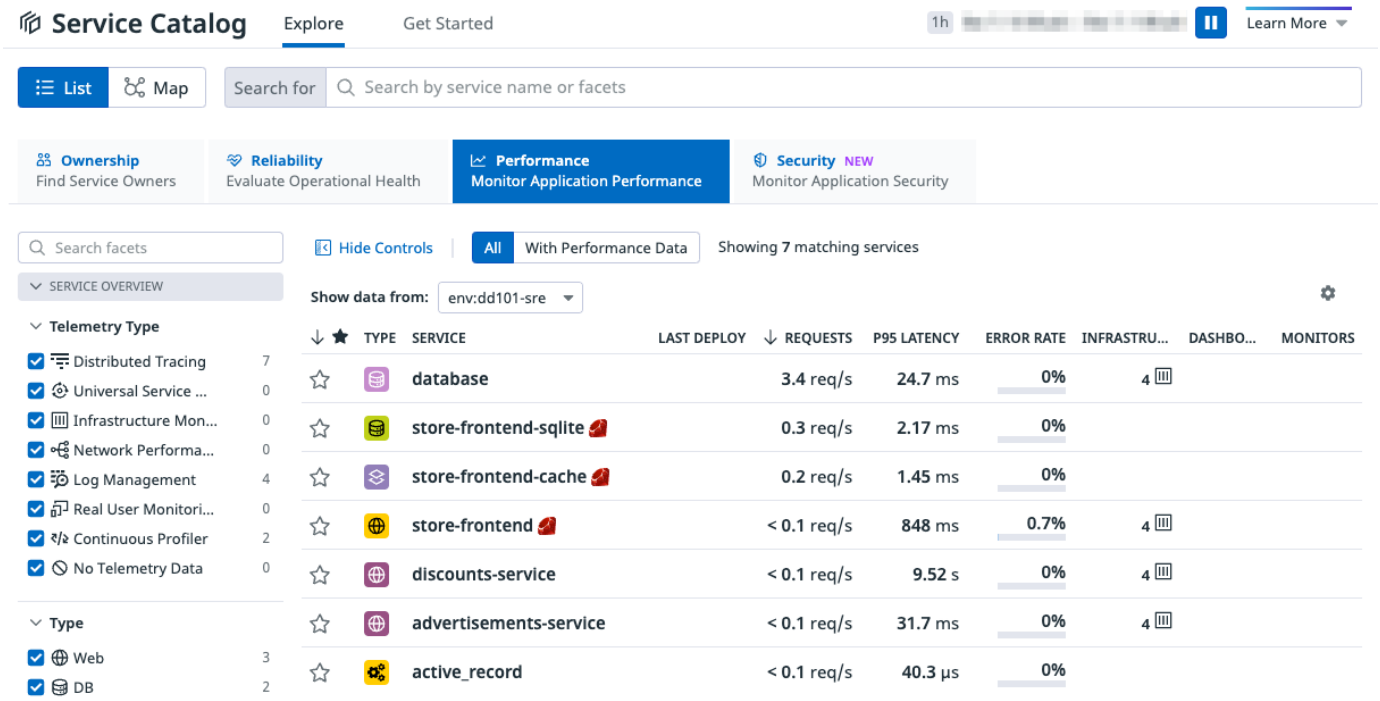
Find the **APM Agent** section of the output:

```
Receiver (previous minute)
==========================
  From python 3.9.6 (CPython), client 0.57.3
    Traces received: 31 (1,921,989 bytes)
    Spans received: 4042

  From ruby 2.7.2 (ruby-x86_64-linux), client 0.50.0
    Traces received: 224 (1,988,062 bytes)
    Spans received: 5019

  Default priority sampling rate: 100.0%
  Priority sampling rate for 'service:active_record,env:dd101-sre': 100.0%
  Priority sampling rate for 'service:advertisements-service,env:dd101-sre': 100.0%
  Priority sampling rate for 'service:discounts-service,env:dd101-sre': 100.0%
  Priority sampling rate for 'service:store-frontend,env:dd101-sre': 100.0%
  Priority sampling rate for 'service:store-frontend-cache,env:dd101-sre': 100.0%
  Priority sampling rate for 'service:store-frontend-sqlite,env:dd101-sre': 100.0%
```

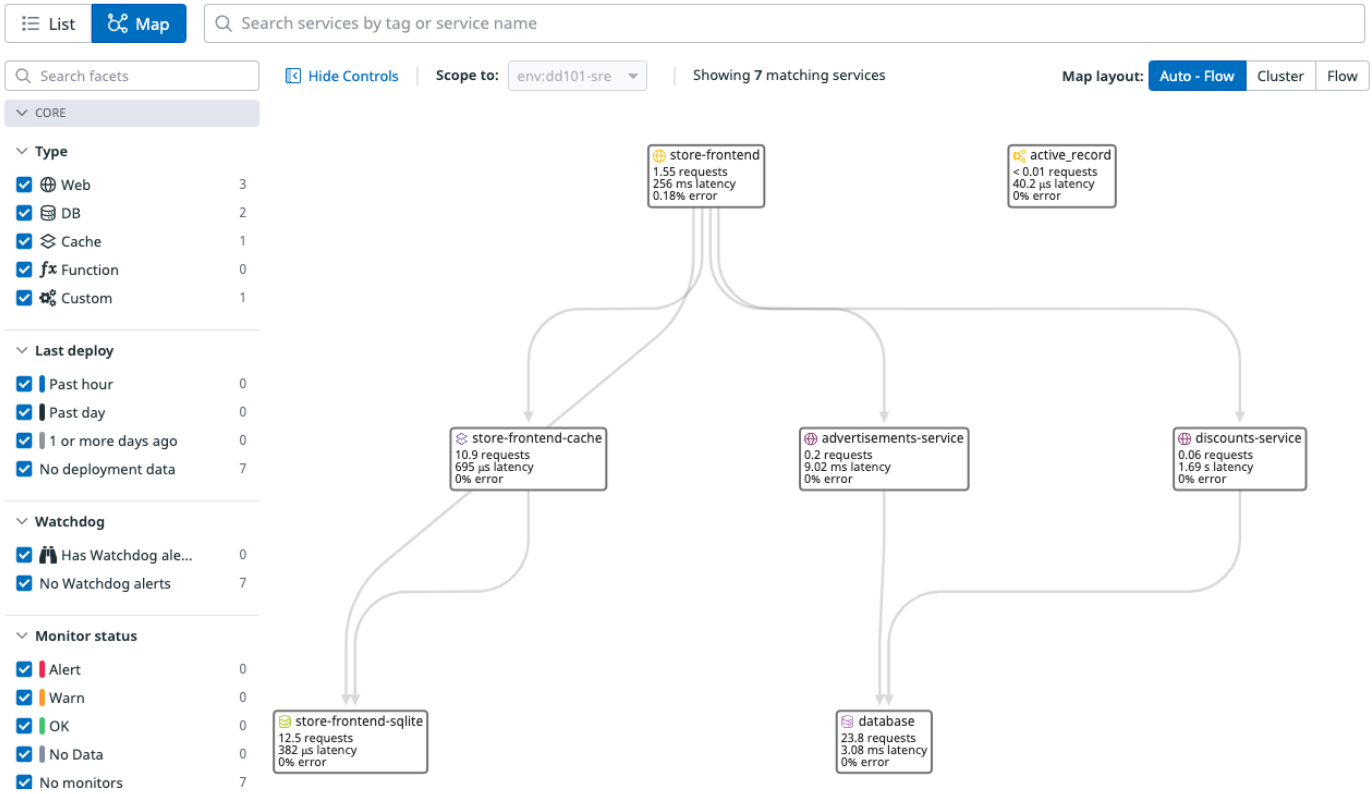
Note: It may take a few minutes for the Agent to pick up the new traces.

Notice that in addition to the services configured in `docker-compose.yml`, two new services are emitting traces: `store-frontend-cache` and `store-frontend-sqlite`. These are running in the `frontend` container, and APM can collect their traces as well.

5. Navigate to **APM > Service Catalog** in the Datadog app, and explore the traces for the entire application. It may take a few minutes for them all to roll in. You might have to refresh the page to see them.

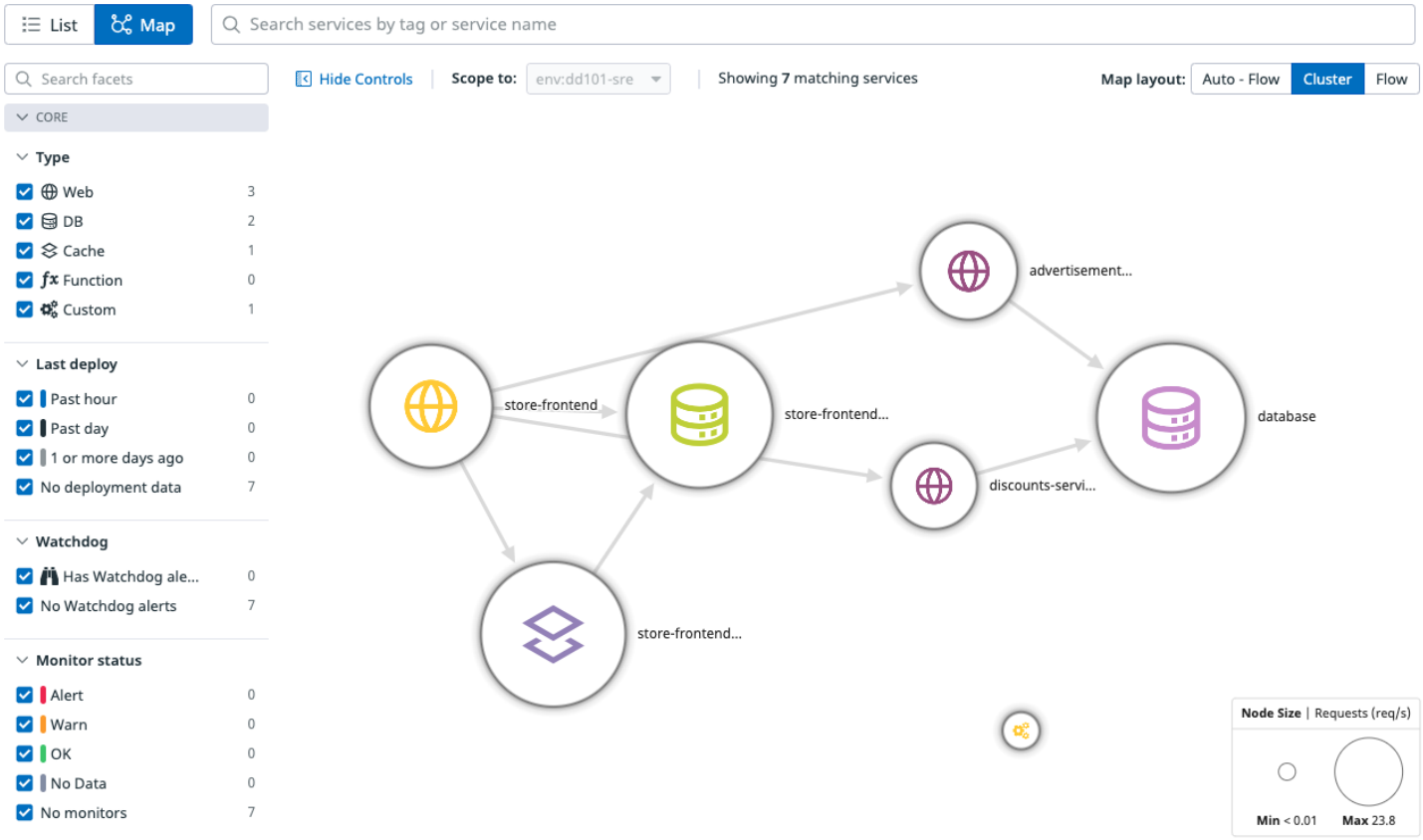


6. Finally, navigate to **APM > Service Map** to visualize the services communicating with each other.



> **Note**: It can take a some time for newly instrumented services to appear in the Service Map. Feel free to come back to this page later.

There are two Service Map layouts: **Flow**, pictured above and optimized for larger maps; and **Cluster**, optimized for smaller maps. Click the **Cluster** control in the upper-right corner to see that layout:



In the next section, you'll take a look at the metrics that the Continuous Profiler displays.

SREs and Continuous Profiling

Datadog's Continuous Profiler is a powerful APM feature. It gives you insight into the system resource consumption of your applications beyond traces. You can see CPU time, memory allocation, file IO, garbage collection, network throughput, and more.

These metrics are useful for application developers to assess and improve their code performance. They're also useful for SREs to dig deeper into issues that affect their infrastructure. More information leads to better decision making, and more efficient delegation to other teams.

The continuous profiler can also alert you to inefficient code that might have billing consequences, or inform your decisions about resource allocation.

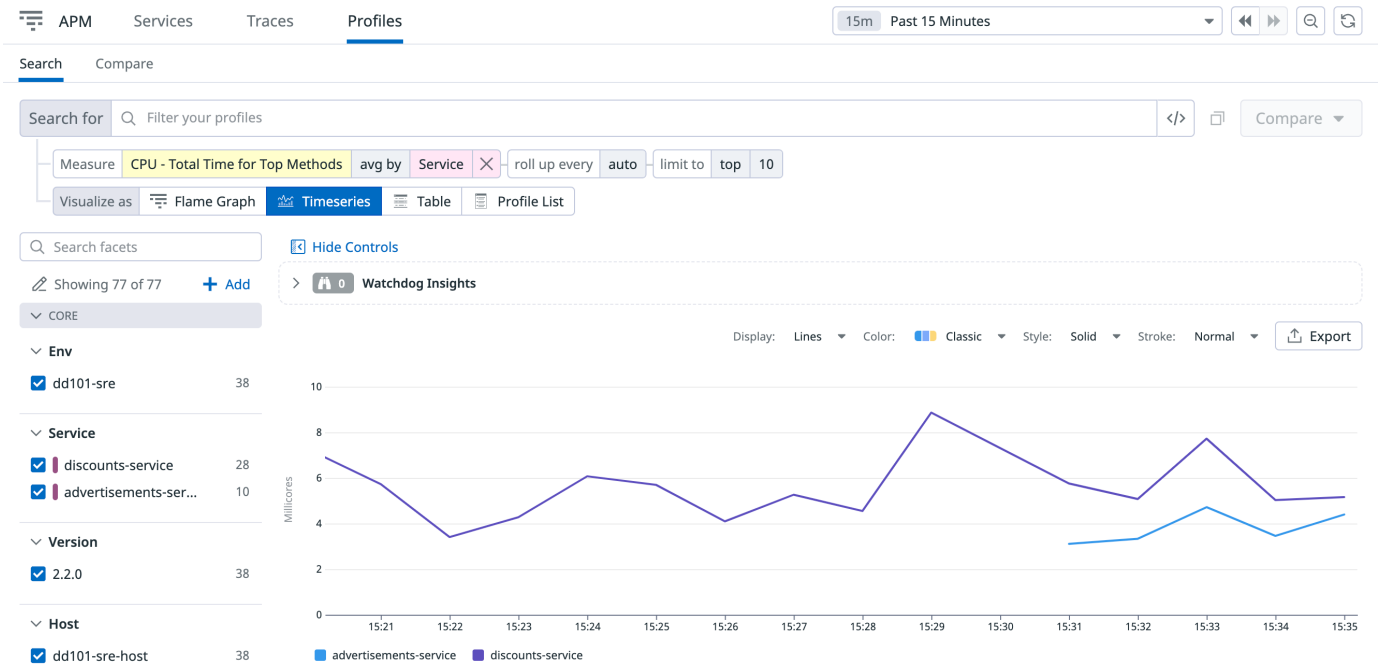
Datadog libraries support profiling for Go, Java, node, Python, and Ruby. You can learn more about configuring profiling for these languages in the Profiling Documentation.

> **Note**: Profiling is enabled automatically for Storedog's Python services that send APM traces. Profiling for Ruby just came out of beta, and the `store-frontend` service has not yet enabled it.

1. Navigate to **APM > Profiles**.

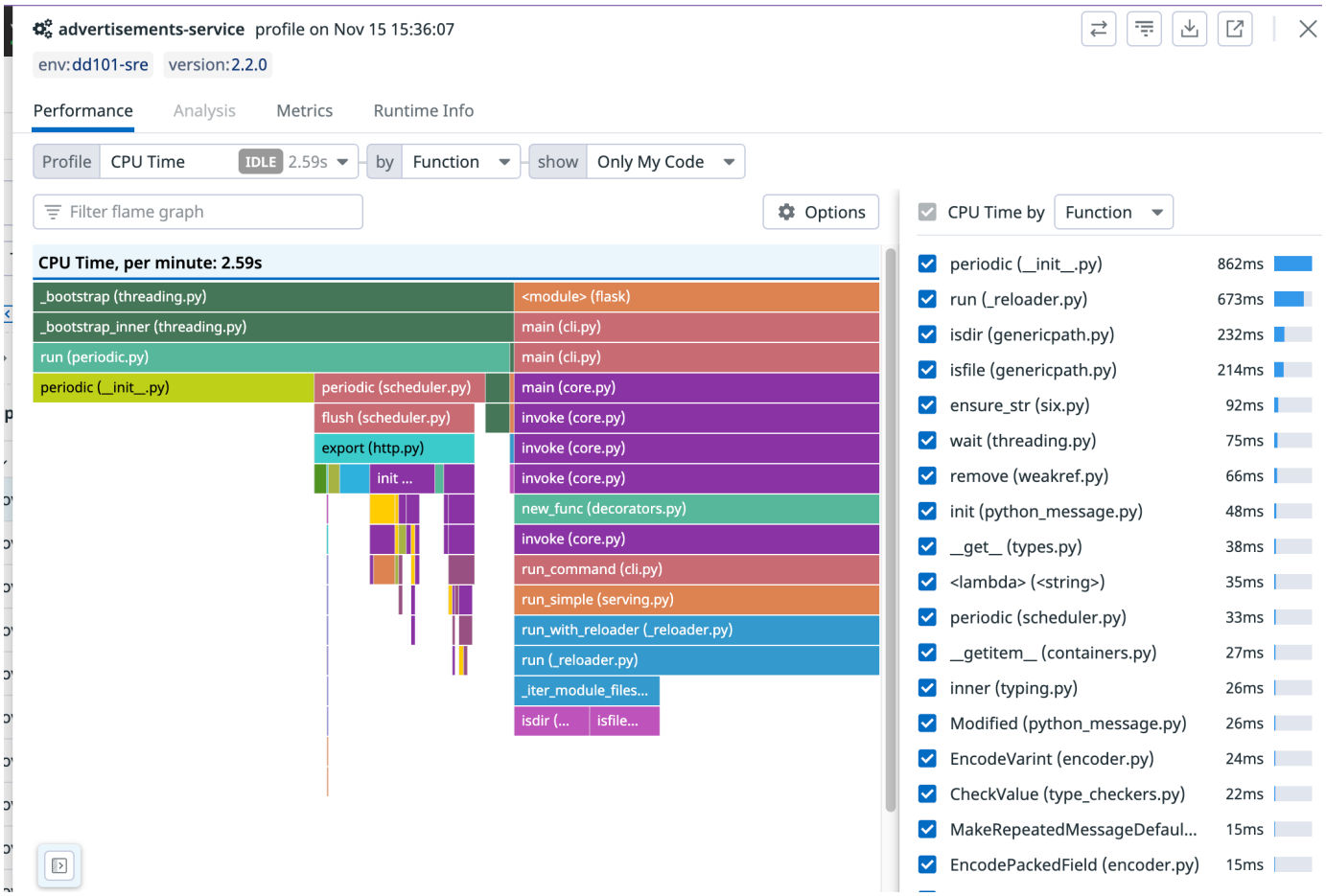
   > **Note**: If you see a "Discover Datadog Continuous Profiler" page, refresh the page.

2. In the timeframe dropdown in the upper-right corner, select `Past 15 Minutes`.
3. In the **Service** facet section of the left-hand panel, select `discounts-service` and `advertisements-service`.
4. For **Visualize as**, select **Timeseries**.
5. For **Measure**, keep `CPU - Total Time for Top Methods` but change **avg by** to `Service` to compare the two services.



   Note that you can also change the **Measure** to a variety of other metrics, such as `CPU Cores` and `Wall Time`.

6. Switch **Visualize as** to **Profile List** and click on a profile for `advertisements-service`. It will open up in a side panel:



7. Mouse over the spans in the flame graph to see more information.
8. In the **CPU Time by** dropdown to the right, change **Function** to **Library**. This gives you insight into the proportion of resources consumed by frameworks, as compared to the application your organization builds upon them.

The important parts to remember about the continuous profiler are how to enable it, and what it provides. It's another powerful tool in your Datadog toolbox, and you should let your developers know about it if they don't already!

Lab Conclusion

Great job! You have examined how APM was enabled on all of Storedog's services, and you can fluidly navigate between their traces and log entries in the Datadog app.

You also know that APM provides profiling, and where to look for documentation for configuring it for your applications. This is a great resource for developers to optimize their code and reduce strain on resources.

When you're done, enter the following command in the terminal:

```
finish
```

Click the **Check** button in the lower right corner of the lab and wait for the lab to close down before moving on to the next lesson.