Application Performance Monitoring Lab

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

Application traces are continuous streams of data about an application's execution. They can tell you the specific lines of code that ran, when they ran, and how long they took.

When configured correctly, Datadog associates 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.

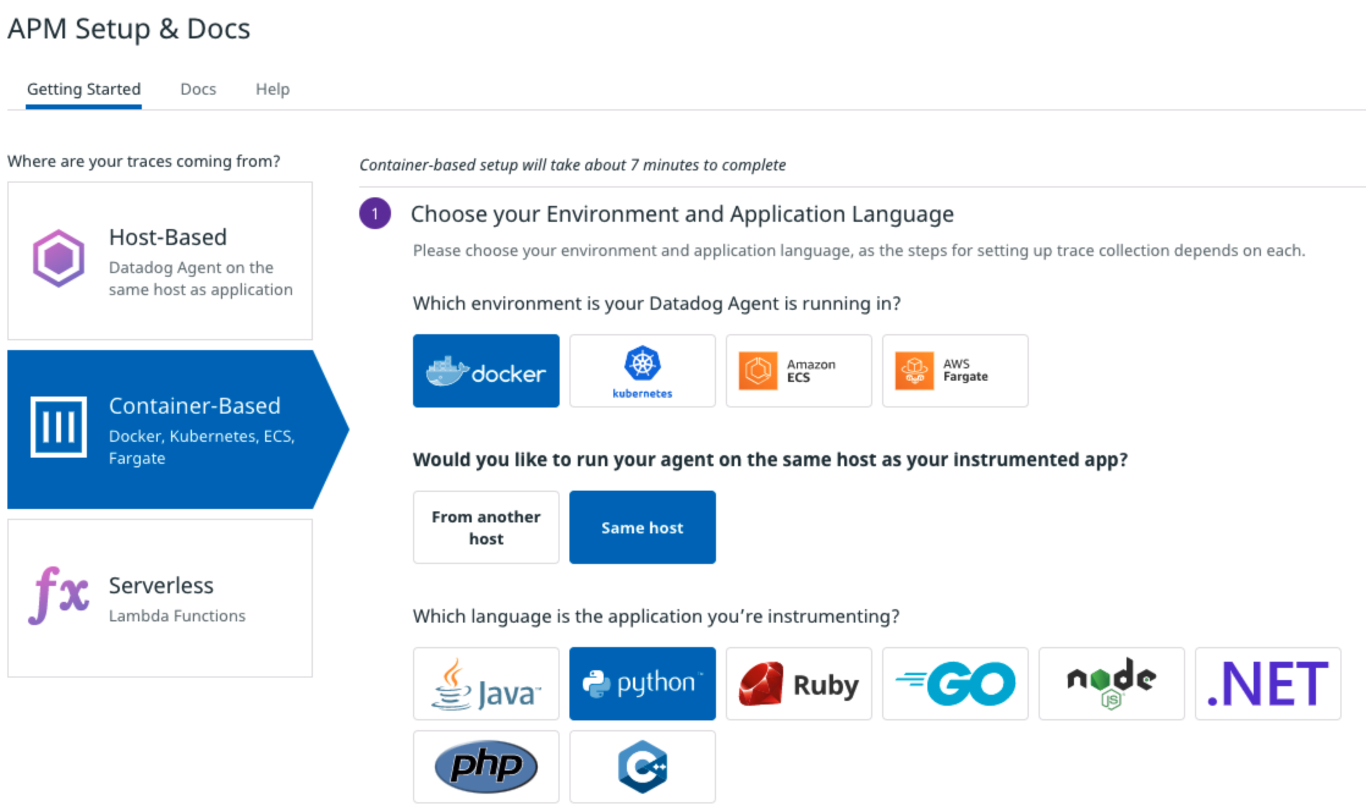
The Datadog Agent is automatically configured to accept traces. You need to instrument your applications to submit them to the Agent. This is done by adding a Datadog tracing library to your application.

Examine the Configured Service

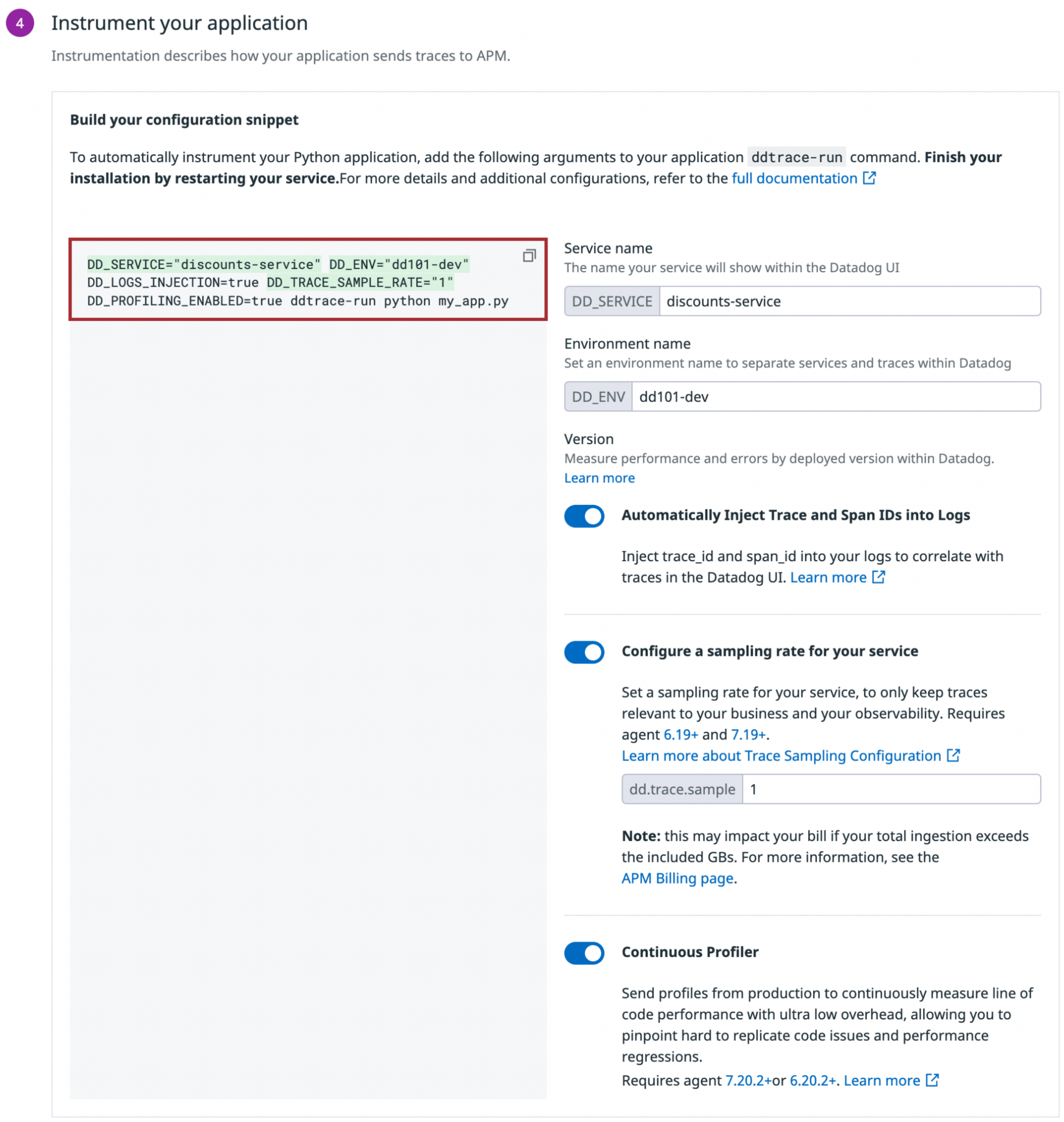
Everything for APM has been configured for you already in this lab, but it's important to know how it's done.

First, take a look at how Datadog guides you through enabling tracing for many languages in several environments:

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**.


APM Setup & Docs
Getting Started
Docs
Help
Where are your traces coming from?
Host-Based
Datadog Agent on the same host as application
Container-Based
Docker, Kubernetes, ECS, Fargate
Serverless
Lambda Functions
Container-based setup will take about 7 minutes to complete
1
Choose your Environment and Application Language
Please choose your environment and application language, as the steps for setting up trace collection depends on each.
Which environment is your Datadog Agent is running in?
docker
kubernetes
Amazon ECS
AWS Fargate
Would you like to run your agent on the same host as your instrumented app?
From another host
Same host
Which language is the application you're instrumenting?
Java
python
Ruby
GO
node
.NET
php

5. 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 for Storedog.
   - 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.
6. Under the **Instrument your application** step, do the following to see what the APM configuration for this lab's discounts service would look like:
   1. In the form, set **DD_SERVICE** to `discounts-service`.
   2. Set **DD_ENV** to `dd101-dev`.
   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:



4 Instrument your application

Instrumentation describes how your application sends traces to APM.

Build your configuration snippet

To automatically instrument your Python application, add the following arguments to your application `ddtrace-run` command. **Finish your installation by restarting your service.** For more details and additional configurations, refer to the full documentation

```
DD_SERVICE="discounts-service" DD_ENV="dd101-dev"
DD_LOGS_INJECTION=true DD_TRACE_SAMPLE_RATE="1"
DD_PROFILING_ENABLED=true ddtrace-run python my_app.py
```

Service name
The name your service will show within the Datadog UI

DD_SERVICE: discounts-service

Environment name
Set an environment name to separate services and traces within Datadog

DD_ENV: dd101-dev

Version
Measure performance and errors by deployed version within Datadog.
Learn more

Automatically Inject Trace and Span IDs into Logs

Inject trace_id and span_id into your logs to correlate with traces in the Datadog UI. Learn more

Configure a sampling rate for your service

Set a sampling rate for your service, to only keep traces relevant to your business and your observability. Requires agent 6.19+ and 7.19+.
Learn more about Trace Sampling Configuration

dd.trace.sample: 1

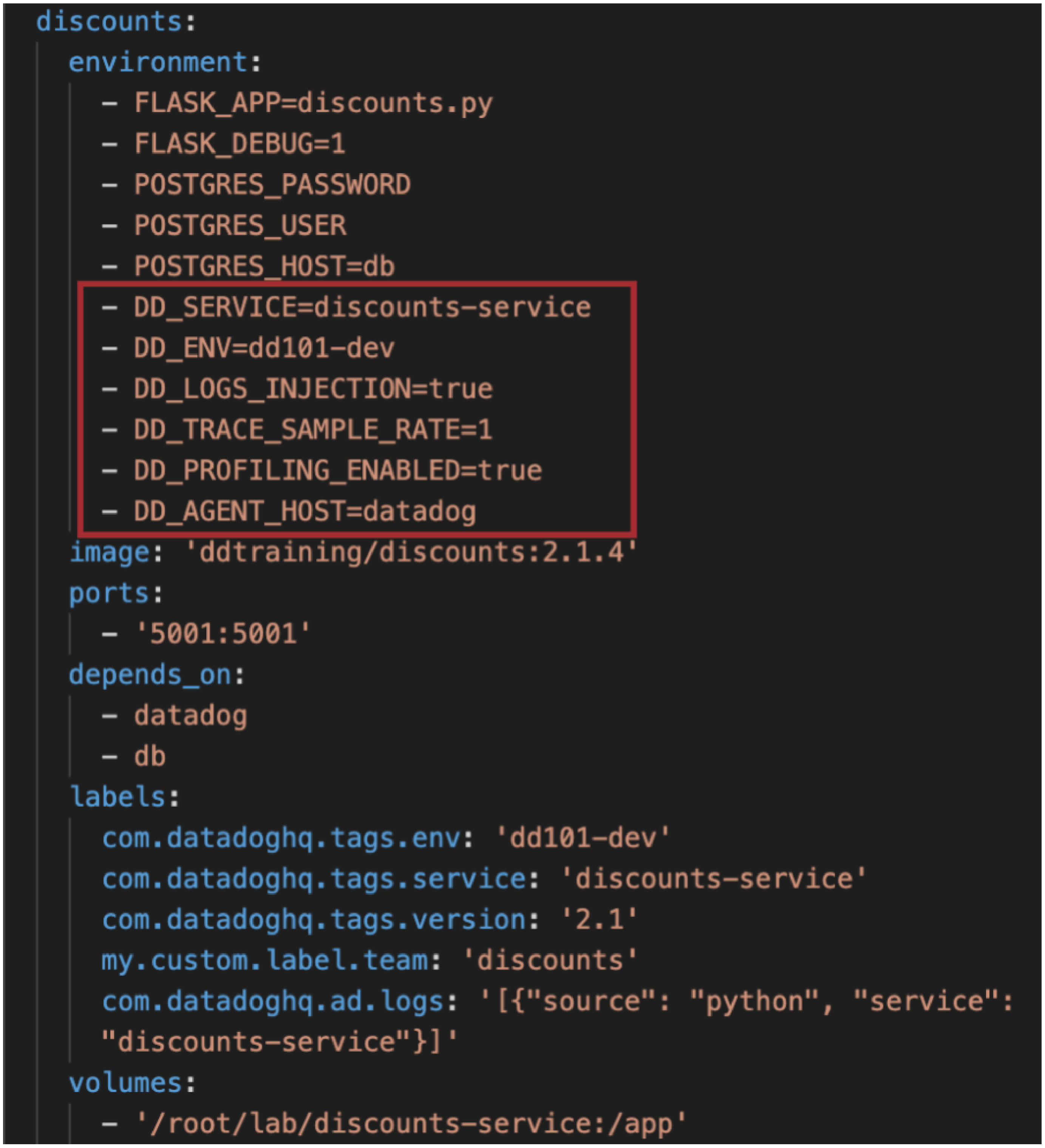
Note: this may impact your bill if your total ingestion exceeds the included GBs. For more information, see the APM Billing page.

Continuous Profiler

Send profiles from production to continuously measure line of code performance with ultra low overhead, allowing you to pinpoint hard to replicate code issues and performance regressions.
Requires agent 7.20.2+or 6.20.2+. Learn more

Now compare that with the configuration that is already in the `docker-compose.yml` file for Storedog.

1. In the lab IDE, open `docker-compose.yml`.
2. Compare that configuration snippet from Datadog with what's in `docker-compose.yml`:



```
discounts:
  environment:
    - FLASK_APP=discounts.py
    - FLASK_DEBUG=1
    - POSTGRES_PASSWORD
    - POSTGRES_USER
    - POSTGRES_HOST=db
    - DD_SERVICE=discounts-service
    - DD_ENV=dd101-dev
    - DD_LOGS_INJECTION=true
    - DD_TRACE_SAMPLE_RATE=1
    - DD_PROFILING_ENABLED=true
    - DD_AGENT_HOST=datadog
  image: 'ddtraining/discounts:2.1.4'
  ports:
    - '5001:5001'
  depends_on:
    - datadog
    - db
  labels:
    com.datadoghq.tags.env: 'dd101-dev'
    com.datadoghq.tags.service: 'discounts-service'
    com.datadoghq.tags.version: '2.1'
    my.custom.label.team: 'discounts'
    com.datadoghq.ad.logs: '[{"source": "python", "service":
    "discounts-service"}]'
  volumes:
    - '/root/lab/discounts-service:/app'
```

These values match the values you set under the **Instrument your application** step in Datadog. They provide context for the Datadog tracing library installed into each of these services at the application-level.

The `DD_AGENT_HOST` tells the Datadog tracing library to send traces to the `datadog` service.

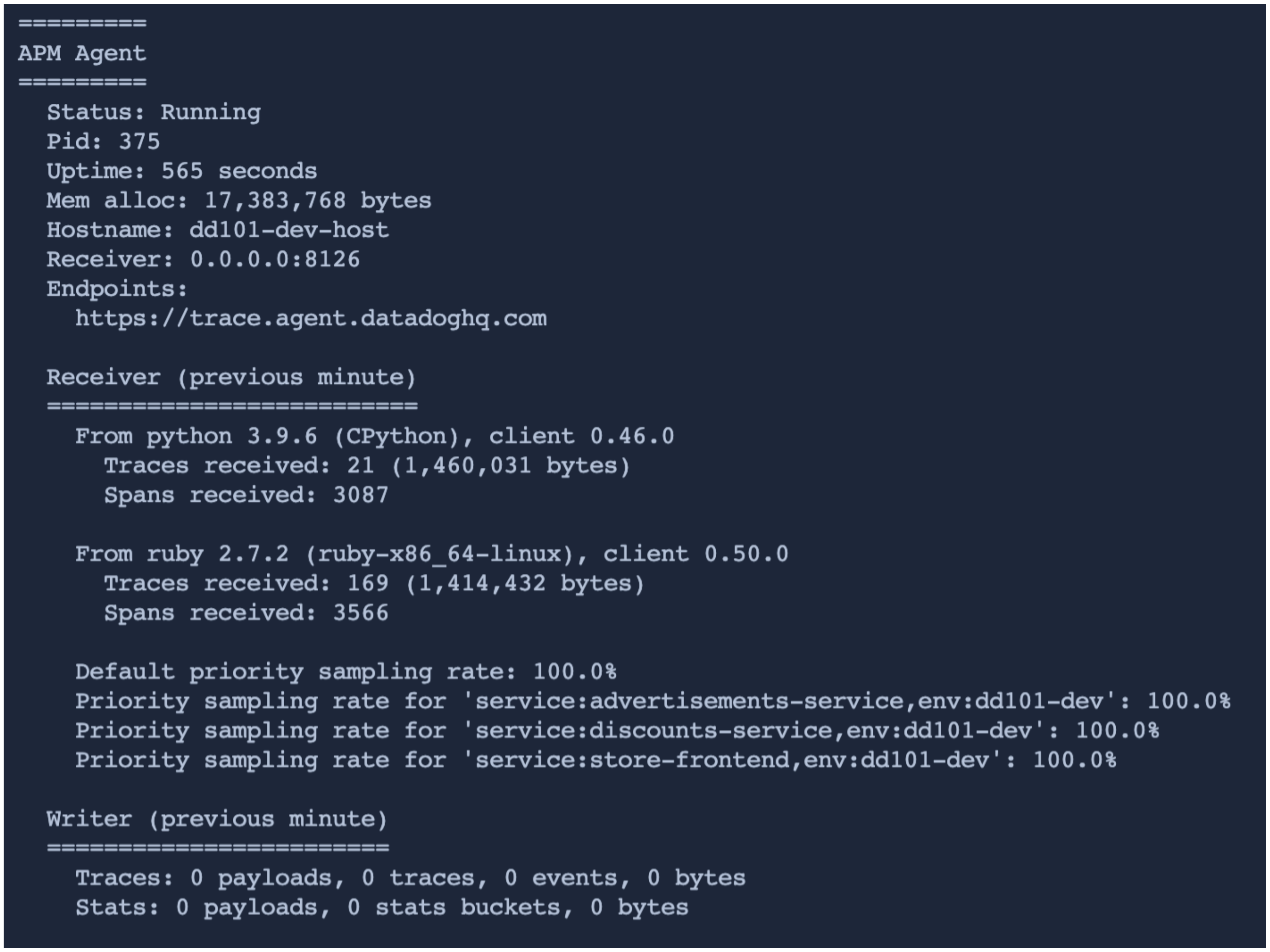
3. In the IDE, open `discounts-service/requirements.txt`. These are the libraries that will be installed when the discounts service starts up.

   On line 4, you can see that `ddtrace` is already listed. Thanks to Storedog's forward-thinking developers, you don't need to do anything further with the application code.

4. In the lab terminal, confirm that the `discounts-service`, `advertisements-service`, and `store-frontend` service are all sending traces to APM by running the following command:

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

5. In the **APM Agent** section, notice it is receiving Python traces from all three services:



```
=========
APM Agent
=========
  Status: Running
  Pid: 375
  Uptime: 565 seconds
  Mem alloc: 17,383,768 bytes
  Hostname: dd101-dev-host
  Receiver: 0.0.0.0:8126
  Endpoints:
    https://trace.agent.datadoghq.com

  Receiver (previous minute)
  ==========================
    From python 3.9.6 (CPython), client 0.46.0
      Traces received: 21 (1,460,031 bytes)
      Spans received: 3087

    From ruby 2.7.2 (ruby-x86_64-linux), client 0.50.0
      Traces received: 169 (1,414,432 bytes)
      Spans received: 3566

    Default priority sampling rate: 100.0%
    Priority sampling rate for 'service:advertisements-service,env:dd101-dev': 100.0%
    Priority sampling rate for 'service:discounts-service,env:dd101-dev': 100.0%
    Priority sampling rate for 'service:store-frontend,env:dd101-dev': 100.0%

  Writer (previous minute)
  ========================
    Traces: 0 payloads, 0 traces, 0 events, 0 bytes
    Stats: 0 payloads, 0 stats buckets, 0 bytes
```

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 Datadog to process traces when they first start coming in, so wait a few minutes if you don't see anything at first.

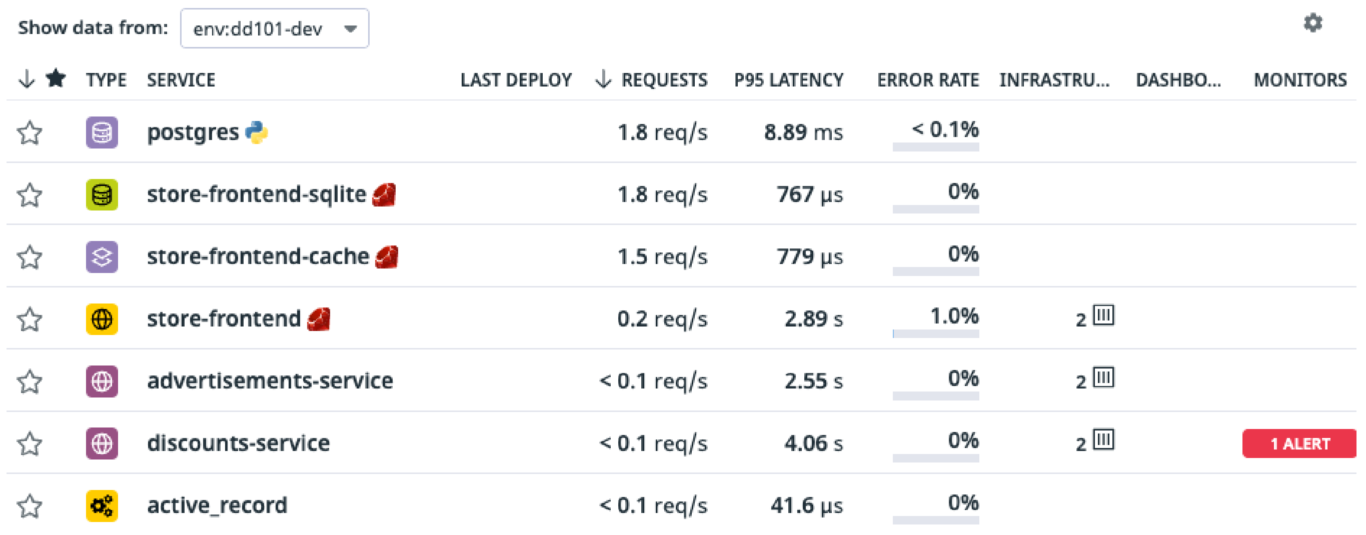
1. In Datadog, navigate to **APM > Service Catalog**. If you have taken other courses within the past two weeks, you likely have a **Show data from** selector with multiple options. Make sure that `env:dd101-dev` is selected.

2. You'll see a list of all the services that are enabled. You'll also see `postgres`, which doesn't communicate with the Datadog Agent directly.

   `postgres` shows up because `discounts-service` and `advertisements-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.

   Similarly, you'll see some new services that don't correspond to Docker containers: `active_record`, `store-frontend-cache`, and `store-frontend-sqlite`. The `store-frontend` service is instrumented to tag these subsidiary areas of the application as services. You can see how this is done in the `store-frontend/config/initializers/datadog.rb` file.

3. You'll notice the `discounts-service` has an alert in the **Monitors** column on the page:



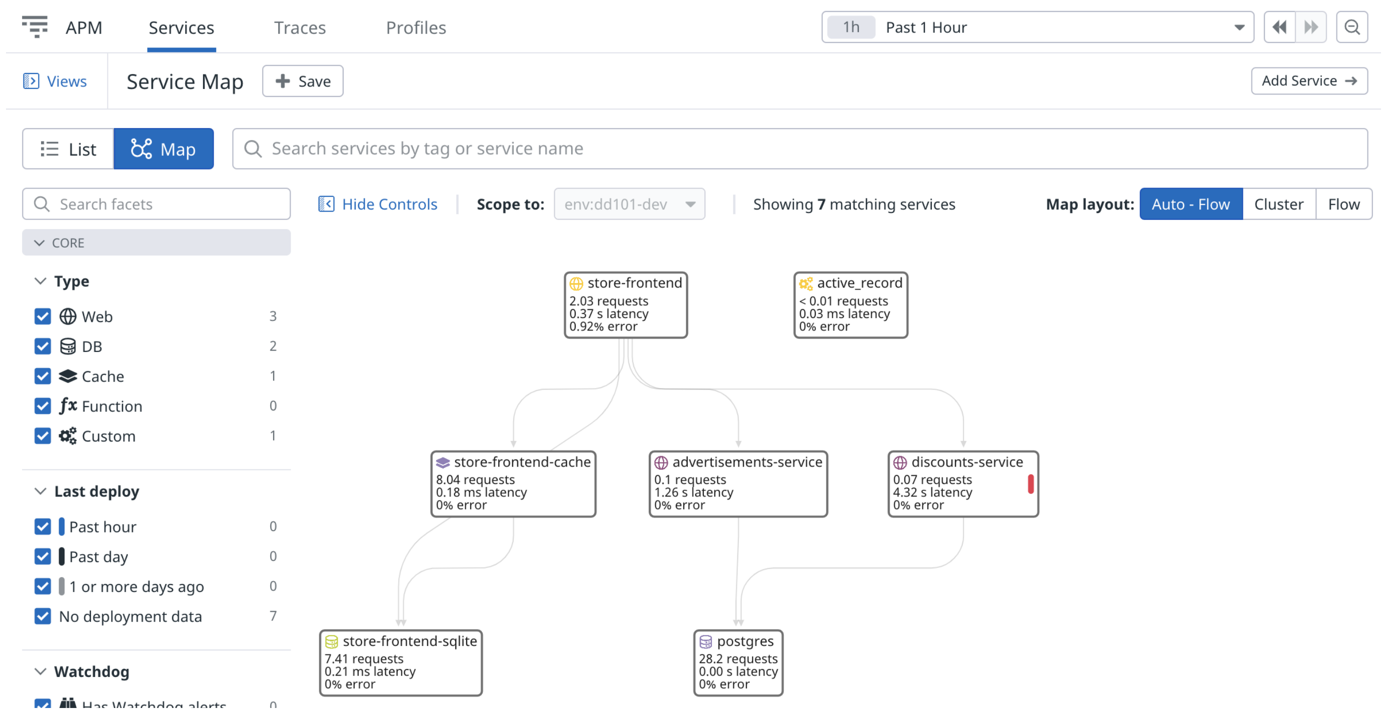
Show data from: env:dd101-dev

TYPE	SERVICE	LAST DEPLOY	REQUESTS	P95 LATENCY	ERROR RATE	INFRASTRU...	DASHBO...	MONITORS
	postgres		1.8 req/s	8.89 ms	< 0.1%			
	store-frontend-sqlite		1.8 req/s	767 µs	0%			
	store-frontend-cache		1.5 req/s	779 µs	0%			
	store-frontend		0.2 req/s	2.89 s	1.0%	2		
	advertisements-service		< 0.1 req/s	2.55 s	0%	2		
	discounts-service		< 0.1 req/s	4.06 s	0%	2		1 ALERT
	active_record		< 0.1 req/s	41.6 µs	0%			

A site reliability engineer (SRE) at Storedog created a monitor to check the latency of this service, and it seems to be a bit higher than expected.

You'll come back to this in a few minutes, but for now, continue exploring APM.

4. Select the **Map** tab in the top left corner, and you'll see a map of how each service communicates with one another:



store-frontend: 2.03 requests, 0.37 s latency, 0.92% error
active_record: < 0.01 requests, 0.03 ms latency, 0% error
store-frontend-cache: 8.04 requests, 0.18 ms latency, 0% error
advertisements-service: 0.1 requests, 1.26 s latency, 0% error
discounts-service: 0.07 requests, 4.32 s latency, 0% error
store-frontend-sqlite: 7.41 requests, 0.21 ms latency, 0% error
postgres: 28.2 requests, 0.00 s latency, 0% error

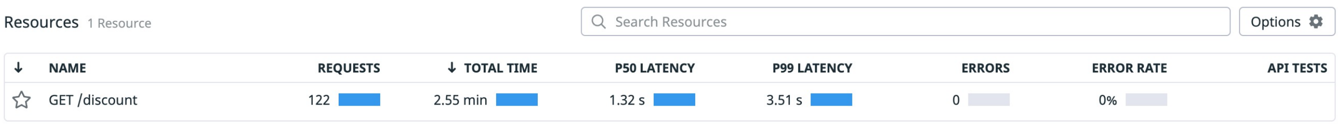
Note: This map can take a long time to appear after Datadog receives new traces. Come back later if you don't see it right now.

5. Notice the **Map layout** buttons in the upper-right corner. Switch between **Cluster** and **Flow** to see the differences. The **Flow** layout is more suited for larger service maps.

   Also notice that the service monitor status is indicated in red for the `discount-service` in these maps.

6. Click on **discounts-service** and then **View service overview**. (If you don't have a service map yet, you can go to the Service Catalog and hover over **discounts-service** in the list then click **Full Page**.)

7. On the resulting page, 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`

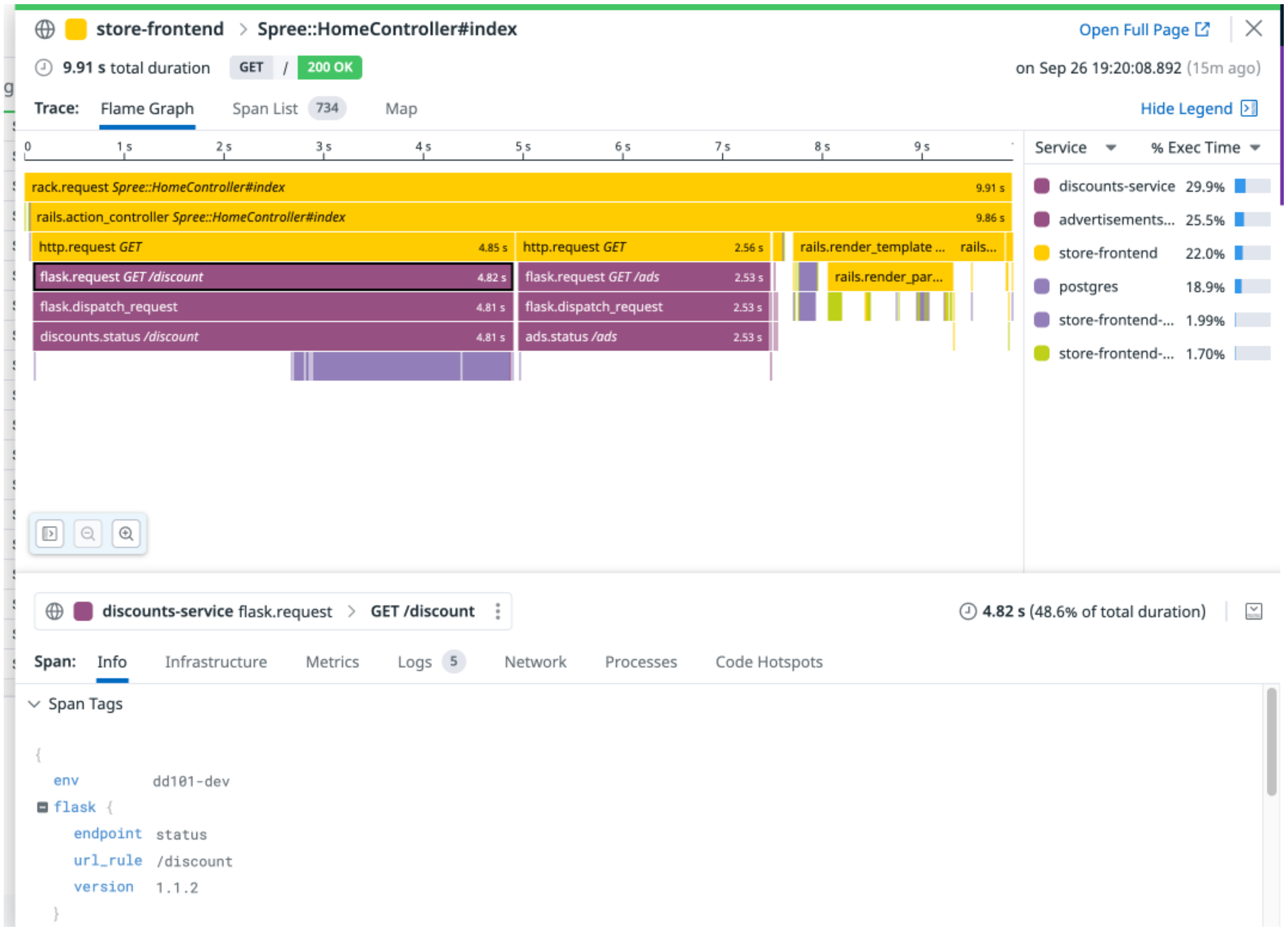

Resources 1 Resource

NAME	REQUESTS	TOTAL TIME	P50 LATENCY	P99 LATENCY	ERRORS	ERROR RATE	API TESTS
GET /discount	122	2.55 min	1.32 s	3.51 s	0	0%	

8. Navigate to **APM > Traces**. Here you see a live stream of the traces APM has captured over the past 15 minutes. Try and locate one for the `discounts-service` service.

   Note: You may have to filter the traces by service name. Use the facets menu on the left to filter by the `discounts-service` service.

9. 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:


store-frontend > Spree::HomeController#index
Open Full Page
9.91 s total duration
GET / 200 OK
on Sep 26 19:20:08.892 (15m ago)
Trace: Flame Graph
Span List 734
Map
Hide Legend
rack.request Spree::HomeController#index 9.91 s
rails.action_controller Spree::HomeController#index 9.86 s
http.request GET 4.85 s
http.request GET 2.56 s
rails.render_template ...
flask.request GET /discount 4.82 s
flask.request GET /ads 2.53 s
rails.render_par...
flask.dispatch_request 4.81 s
flask.dispatch_request 2.53 s
discounts.status /discount 4.81 s
ads.status /ads 2.53 s
Service
% Exec Time
discounts-service 29.9%
advertisements... 25.5%
store-frontend 22.0%
postgres 18.9%
store-frontend-... 1.99%
store-frontend-... 1.70%
discounts-service flask.request > GET /discount
4.82 s (48.6% of total duration)
Span: Info
Infrastructure
Metrics
Logs 5
Network
Processes
Code Hotspots
Span Tags

```
{
  env        dd101-dev
  flask {
    endpoint  status
    url_rule  /discount
    version   1.1.2
  }
```

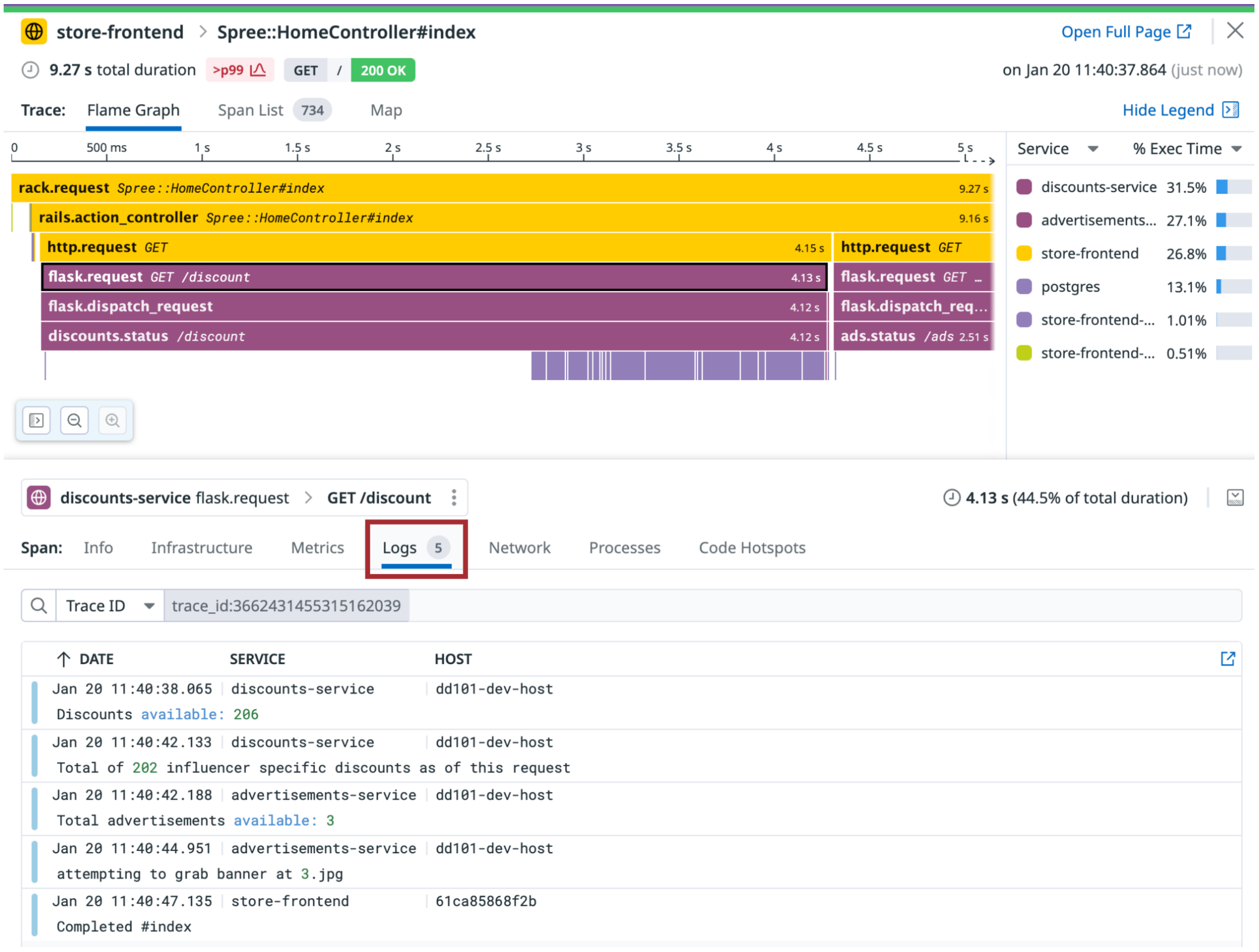
Traverse APM Traces to Log Entries

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.

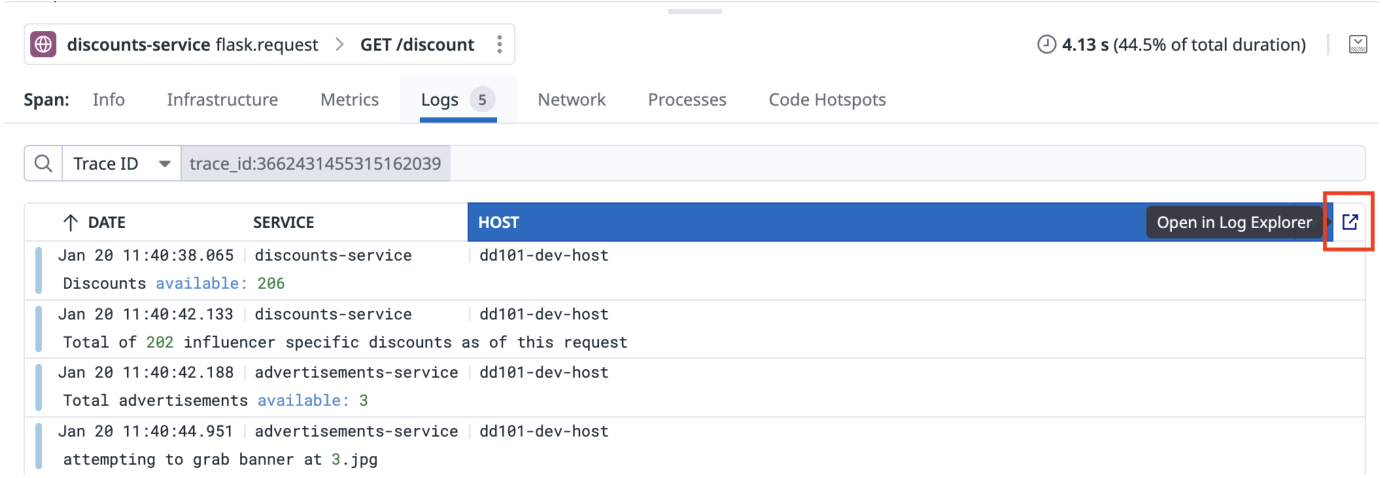
   These are the related log lines captured during the trace's timeframe. You may notice that it's taking a long time to complete this request.


store-frontend > Spree::HomeController#index
Open Full Page
9.27 s total duration >p99 GET / 200 OK
on Jan 20 11:40:37.864 (just now)
Trace: Flame Graph Span List 734 Map
Hide Legend
rack.request Spree::HomeController#index 9.27 s
rails.action_controller Spree::HomeController#index 9.16 s
http.request GET 4.15 s
http.request GET
flask.request GET /discount 4.13 s
flask.dispatch_request 4.12 s
discounts.status /discount 4.12 s
ads.status /ads 2.51 s
Service % Exec Time
discounts-service 31.5%
advertisements... 27.1%
store-frontend 26.8%
postgres 13.1%
store-frontend-... 1.01%
store-frontend-... 0.51%
discounts-service flask.request > GET /discount
4.13 s (44.5% of total duration)
Span: Info Infrastructure Metrics Logs 5 Network Processes Code Hotspots
Trace ID trace_id:3662431455315162039

DATE	SERVICE	HOST
Jan 20 11:40:38.065	discounts-service	dd101-dev-host
Discounts available: 206		
Jan 20 11:40:42.133	discounts-service	dd101-dev-host
Total of 202 influencer specific discounts as of this request		
Jan 20 11:40:42.188	advertisements-service	dd101-dev-host
Total advertisements available: 3		
Jan 20 11:40:44.951	advertisements-service	dd101-dev-host
attempting to grab banner at 3.jpg		
Jan 20 11:40:47.135	store-frontend	61ca85868f2b
Completed #index		

2. Mouse over each line and look at the flamegraph. 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.

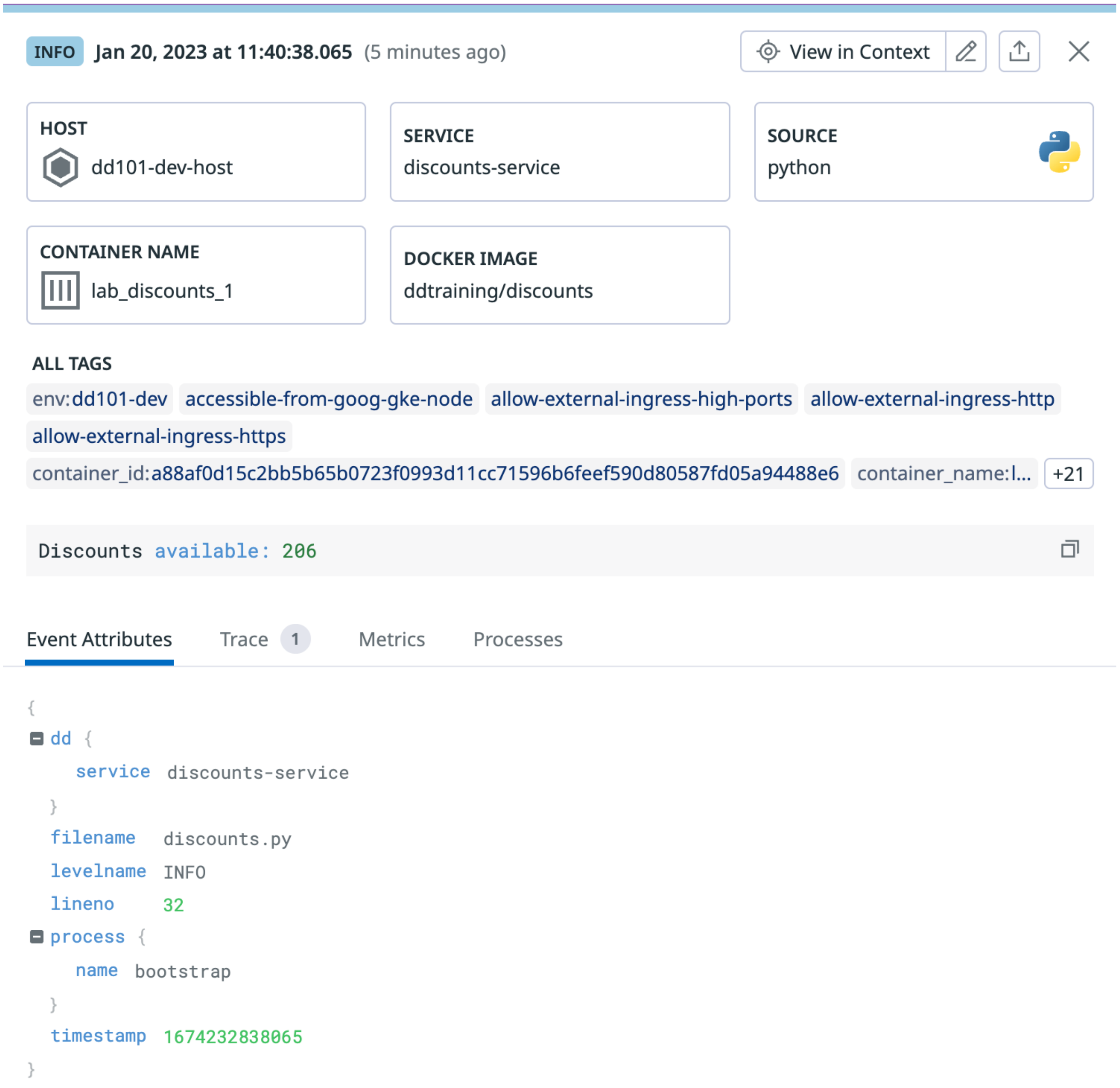
3. In the **Logs** table, click the icon for **Open in Log Explorer** to the far end of the **Hosts** column:


discounts-service flask.request > GET /discount
4.13 s (44.5% of total duration)
Span: Info Infrastructure Metrics Logs 5 Network Processes Code Hotspots
Trace ID trace_id:3662431455315162039
Open in Log Explorer

DATE	SERVICE	HOST
Jan 20 11:40:38.065	discounts-service	dd101-dev-host
Discounts available: 206		
Jan 20 11:40:42.133	discounts-service	dd101-dev-host
Total of 202 influencer specific discounts as of this request		
Jan 20 11:40:42.188	advertisements-service	dd101-dev-host
Total advertisements available: 3		
Jan 20 11:40:44.951	advertisements-service	dd101-dev-host
attempting to grab banner at 3.jpg		

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 to open the log entry details panel.


INFO
Jan 20, 2023 at 11:40:38.065 (5 minutes ago)
View in Context
HOST
dd101-dev-host
SERVICE
discounts-service
SOURCE
python
CONTAINER NAME
lab_discounts_1
DOCKER IMAGE
ddtraining/discounts
ALL TAGS
env:dd101-dev
accessible-from-goog-gke-node
allow-external-ingress-high-ports
allow-external-ingress-http
allow-external-ingress-https
container_id:a88af0d15c2bb5b65b0723f0993d11cc71596b6feef590d80587fd05a94488e6
container_name:l...
+21
Event Attributes
Trace 1
Metrics
Processes

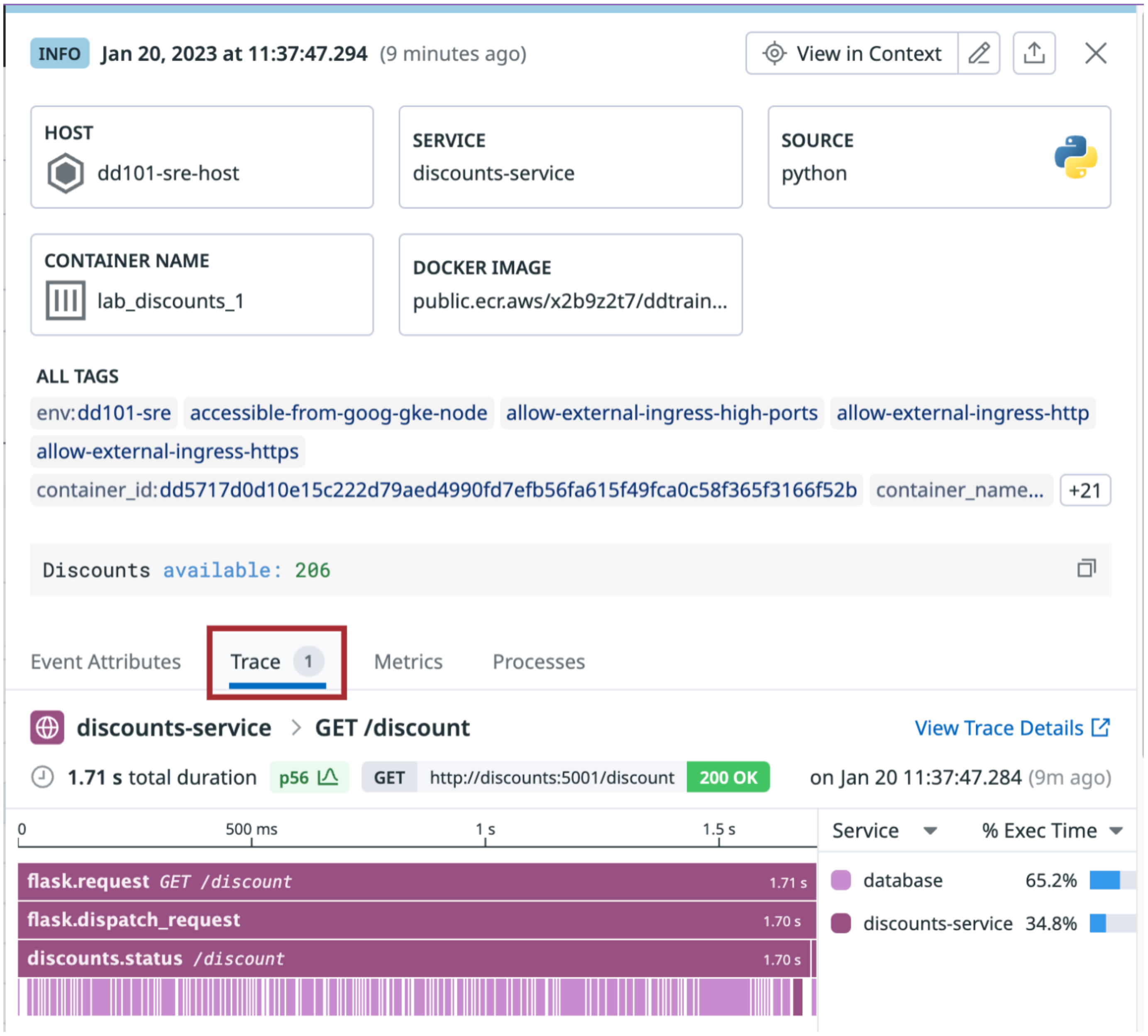
```
Discounts available: 206
```

```
{
  dd {
     service  discounts-service
  }
  filename   discounts.py
  levelname  INFO
  lineno     32
  process {
     name  bootstrap
  }
  timestamp  1674232838065
}
```

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 details side panel, click on the **Trace** tab.


INFO Jan 20, 2023 at 11:37:47.294 (9 minutes ago)

HOST dd101-sre-host

SERVICE discounts-service

SOURCE python

CONTAINER NAME lab_discounts_1

DOCKER IMAGE public.ecr.aws/x2b9z2t7/ddtrain...

ALL TAGS env:dd101-sre accessible-from-goog-gke-node allow-external-ingress-high-ports allow-external-ingress-http allow-external-ingress-https container_id:dd5717d0d10e15c222d79aed4990fd7efb56fa615f49fca0c58f365f3166f52b container_name... +21

```
Discounts available: 206
```

discounts-service > GET /discount

1.71 s total duration p56 GET http://discounts:5001/discount 200 OK on Jan 20 11:37:47.284 (9m ago)

```
flask.request GET /discount      1.71 s
flask.dispatch_request           1.70 s
discounts.status /discount       1.70 s
```

Service	% Exec Time
database	65.2%
discounts-service	34.8%

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.

Examine APM Monitors

In a previous step, you noticed that there was an alert for the discounts service. This alert came from an APM monitor that was created by one of Storedog's SREs to alert you when the discounts service is experiencing performance issues, allowing you to quickly take action to fix the issue.

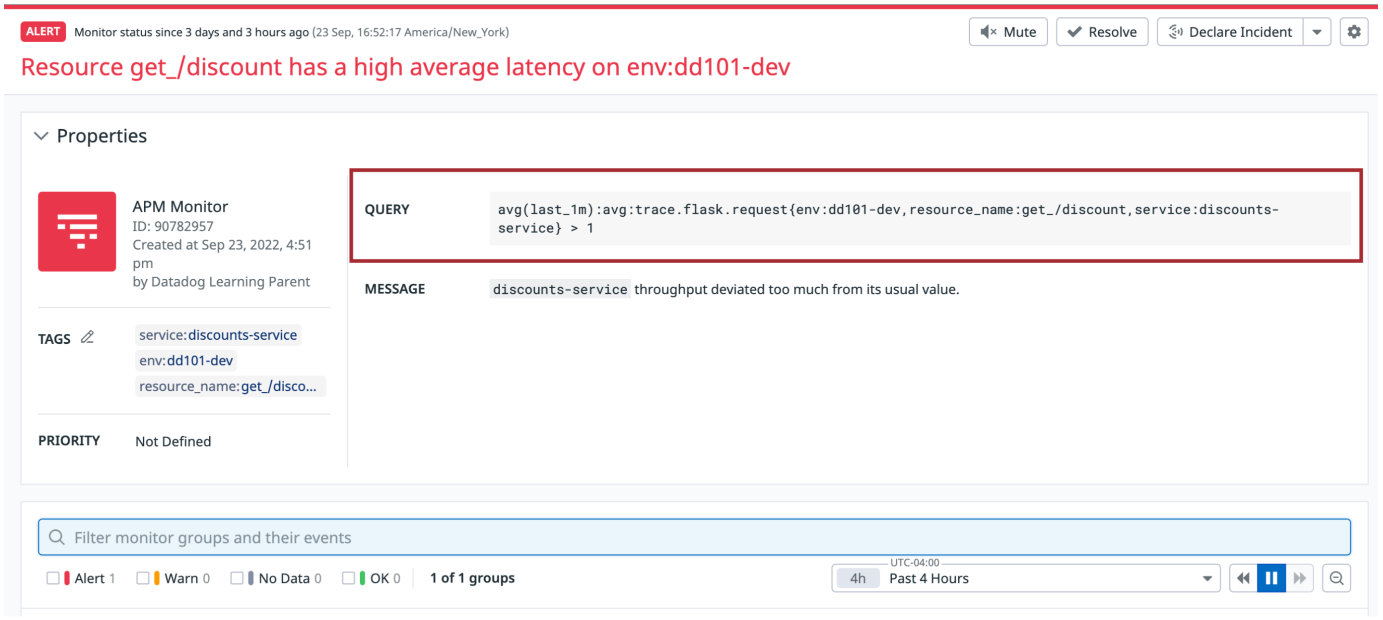
In Datadog, monitors can be created to track any part of an application to detect changes in behavior or performance that you want to be alerted to. For example, you can create a monitor to track the number of requests to a specific endpoint over a period of time, or you can create a monitor to track the number of errors that occur on a specific endpoint.

Check the monitor

1. Navigate to **APM > Service Catalog** to see a list of the services.

   The discounts service has a monitor associated with it on the right.

2. Hover over the monitor's status and select the **View Monitors** link to see the monitor details.
3. On the **Monitors** page, click on the monitor with an **Alert** status to see what this monitor was set up to alert you to.
4. Here you'll find the query that was used to create the monitor, which is to check the average latency of the discounts service over the span of a minute. It is currently around 4 seconds, but the monitor is set to alert you if the latency is greater than 1 second.



ALERT Monitor status since 3 days and 3 hours ago (23 Sep, 16:52:17 America/New_York)

Resource get_/discount has a high average latency on env:dd101-dev

Properties

APM Monitor
ID: 90782957
Created at Sep 23, 2022, 4:51 pm
by Datadog Learning Parent

TAGS: service:discounts-service, env:dd101-dev, resource_name:get_/disco...

PRIORITY: Not Defined

QUERY
```
avg(last_1m):avg:trace.flask.request{env:dd101-dev,resource_name:get_/discount,service:discounts-service} > 1
```

MESSAGE: `discounts-service` throughput deviated too much from its usual value.

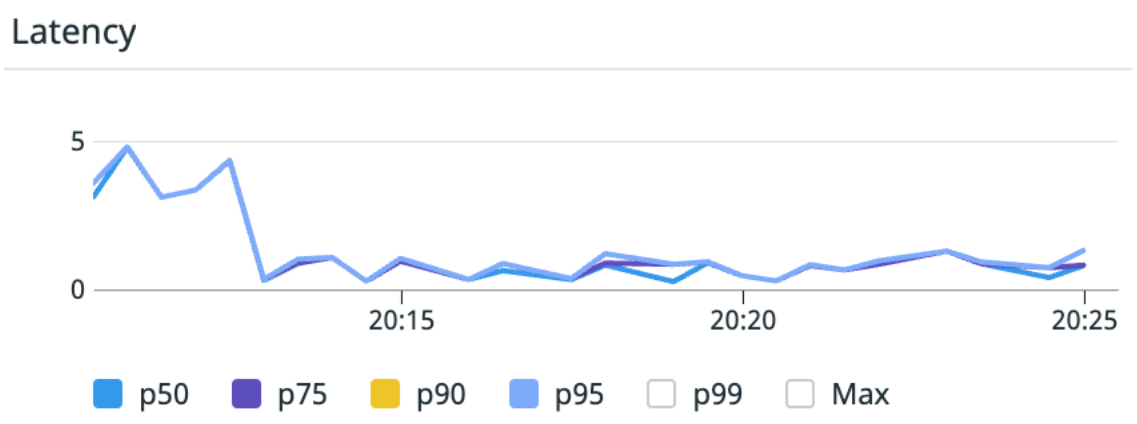
5. Also notice how you can view a list of events around this monitor, so you know when it changes status.

This monitor is set to alert you when the `/discounts` resource is slow, so it's safe to assume that there is an issue in that service's code that you should go look at.

Fix the latency issue

1. In the IDE, open `discounts-service/discounts.py`.
2. Find the `time.sleep()` methods that were previously left in the file by accident.

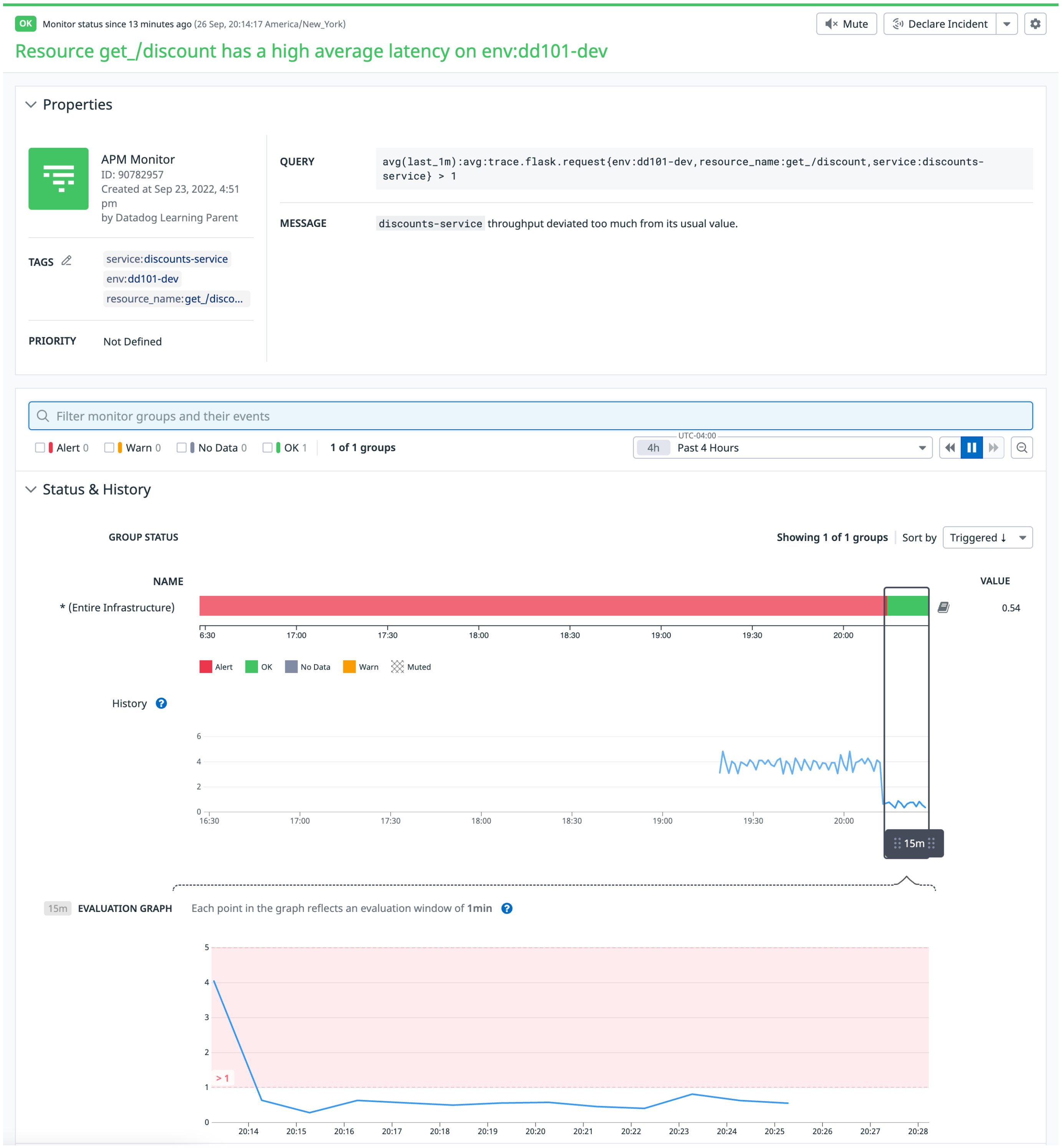
   Remove or comment out those two lines and the IDE automatically saves for you. The changes will immediately update in the running container.
3. It will take a moment or two for the monitor to leave its **Alert** status and return to its **OK** status, so use this time to see if the latency is now less in the **APM > Traces** page.
4. Click on the `discounts-service` facet on the left to see only those from the `discounts-service`.
5. Notice how the latency is now going down. The list of traces may show a difference, but you can also see it in the **Latency** graph towards the top of the page.



6. While you're here, take a moment and explore some of the other services that are also being monitored.

   You'll find that even though the `store-frontend-sqlite` and `store-frontend-cache` services weren't explicitly configured for APM, they are set up within the `store-frontend` service, which we did configure for APM.

7. After a moment or two, the monitor should be in an **OK** status. Navigate to it and you should see something like this image:



OK Monitor status since 13 minutes ago (26 Sep, 20:14:17 America/New_York)

Resource get_/discount has a high average latency on env:dd101-dev

Properties

APM Monitor
ID: 90782957
Created at Sep 23, 2022, 4:51 pm
by Datadog Learning Parent

TAGS: service:discounts-service, env:dd101-dev, resource_name:get_/disco...

PRIORITY: Not Defined

QUERY:
```
avg(last_1m):avg:trace.flask.request{env:dd101-dev,resource_name:get_/discount,service:discounts-service} > 1
```

MESSAGE: `discounts-service` throughput deviated too much from its usual value.

Status & History

NAME	VALUE
* (Entire Infrastructure)	0.54

EVALUATION GRAPH: Each point in the graph reflects an evaluation window of 1min

Note: The monitor may take a few minutes to update its status. Feel free to move onto the next step and check back in a few minutes.

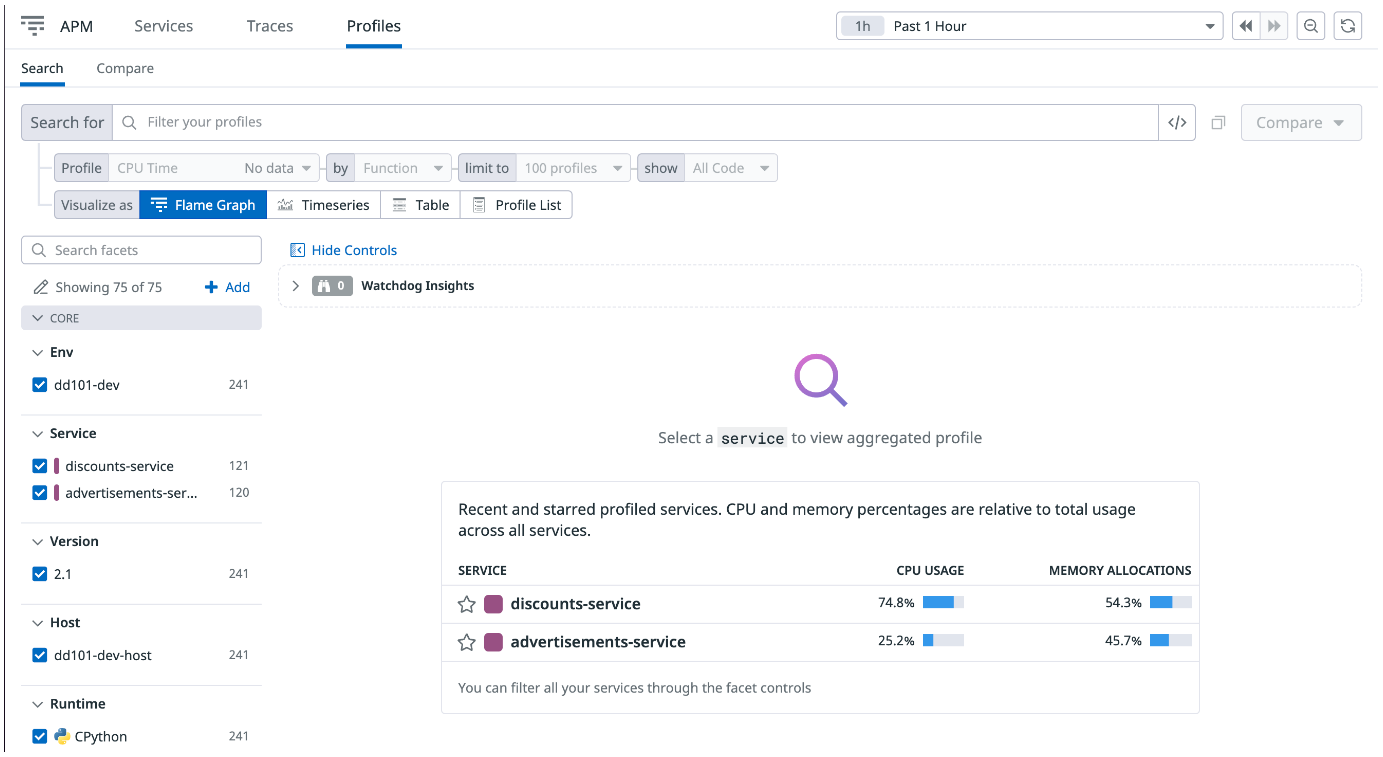
Great work on using an APM monitor to quickly detect performance issues! Having the ability to see the performance of a service in your application is a great way to detect issues that may not be immediately apparent.

Observe Continuous Profiling

Datadog's Continuous Profiler 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.

Profiling is enabled automatically for Storedog's Python services that send APM traces. You can learn more about configuring profiling for different programming languages in the **Profiling Documentation**.

1. Navigate to **APM > Profiles** to see the `advertisements-service` and `discounts-service` profiling. You may have to refresh the page if you see a **Getting Started** page instead.



Profile: CPU Time — No data; by: Function; limit to: 100 profiles; show: All Code

Visualize as: Flame Graph | Timeseries | Table | Profile List

Env
- dd101-dev 241

Service
- discounts-service 121
- advertisements-ser... 120

Version
- 2.1 241

Host
- dd101-dev-host 241

Runtime
- CPython 241

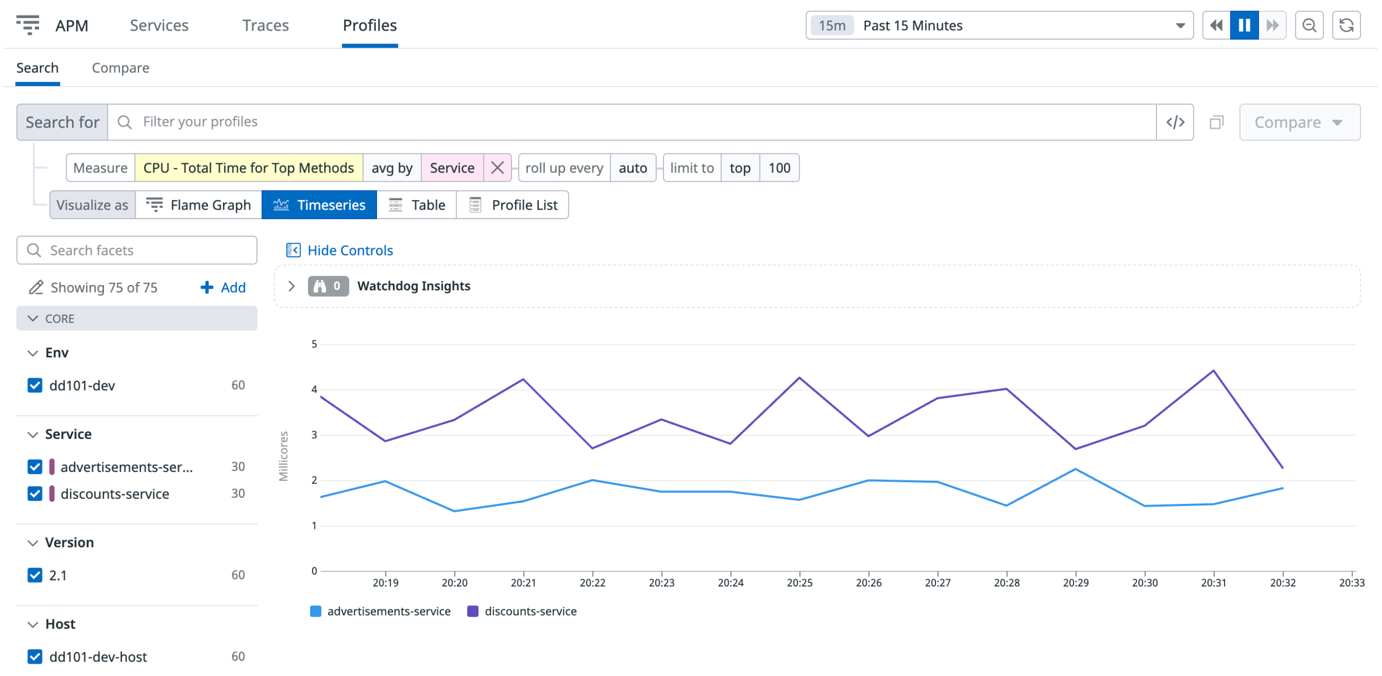
Select a `service` to view aggregated profile

Recent and starred profiled services. CPU and memory percentages are relative to total usage across all services.

SERVICE	CPU USAGE	MEMORY ALLOCATIONS
discounts-service	74.8%	54.3%
advertisements-service	25.2%	45.7%

You can filter all your services through the facet controls

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, make sure `discounts-service` and `advertisements-service` are selected.
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.



Search for: Filter your profiles

Measure: CPU - Total Time for Top Methods, avg by: Service, roll up every: auto, limit to: top 100

Visualize as: Flame Graph | **Timeseries** | Table | Profile List

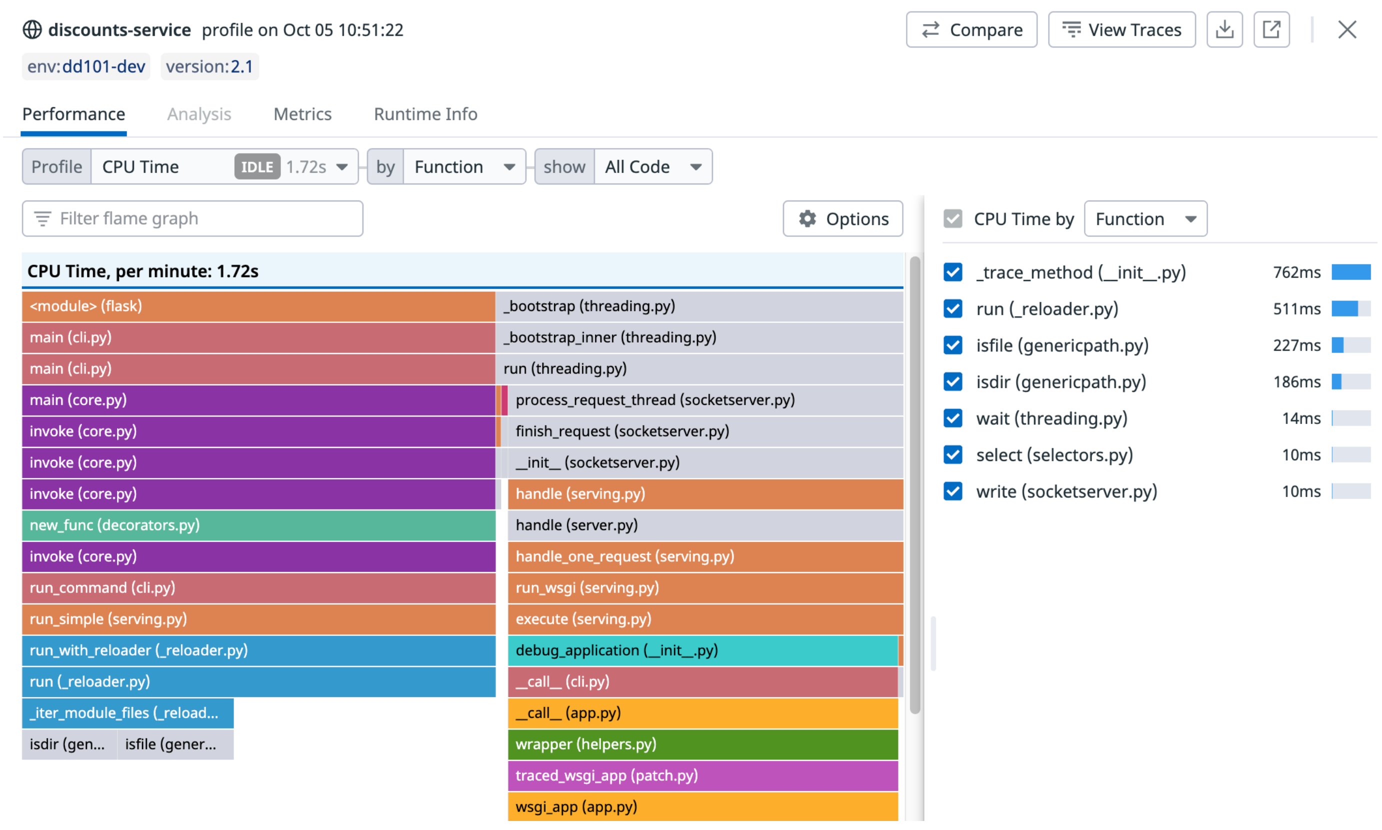
Search facets

Showing 75 of 75 + Add

CORE

Env	
✓ dd101-dev	60
Service	
✓ advertisements-ser...	30
✓ discounts-service	30
Version	
✓ 2.1	60
Host	
✓ dd101-dev-host	60

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



discounts-service profile on Oct 05 10:51:22

env:dd101-dev version:2.1

Profile: CPU Time IDLE 1.72s, by: Function, show: All Code

CPU Time, per minute: 1.72s

```
<module> (flask)                     _bootstrap (threading.py)
main (cli.py)                        _bootstrap_inner (threading.py)
main (cli.py)                        run (threading.py)
main (core.py)                         process_request_thread (socketserver.py)
invoke (core.py)                       finish_request (socketserver.py)
invoke (core.py)                       __init__ (socketserver.py)
invoke (core.py)                       handle (serving.py)
new_func (decorators.py)               handle (server.py)
invoke (core.py)                       handle_one_request (serving.py)
run_command (cli.py)                   run_wsgi (serving.py)
run_simple (serving.py)                execute (serving.py)
run_with_reloader (_reloader.py)       debug_application (__init__.py)
run (_reloader.py)                     __call__ (cli.py)
_iter_module_files (_reload...         __call__ (app.py)
isdir (gen...  isfile (gener...        wrapper (helpers.py)
                                       traced_wsgi_app (patch.py)
                                       wsgi_app (app.py)
```

CPU Time by: Function

✓ _trace_method (__init__.py)	762ms
✓ run (_reloader.py)	511ms
✓ isfile (genericpath.py)	227ms
✓ isdir (genericpath.py)	186ms
✓ wait (threading.py)	14ms
✓ select (selectors.py)	10ms
✓ write (socketserver.py)	10ms

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.

As a developer, this is a valuable tool for finding ways to optimize applications by improving performance and potentially lower resource costs.

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 now also know about how monitors can be set up and used to alert you on potential issues in your application and how you can proactively monitor your application's performance. 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.

For more information on Datadog APM, head over to our Introduction to Application Performance Monitoring Course on the Learning Center.

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.