

Integrations

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.

Overview

Storedog is running in Docker, and the Agent is collecting data from the services running in containers. Currently, the Agent doesn't know what type of services they are.

Integrations help the Agent be more precise about the services it's monitoring. You'll focus on the PostgreSQL integration in this lab because it's a popular example of a Datadog core integration, and you had experience configuring it in a previous lab, "The Agent on a Host".

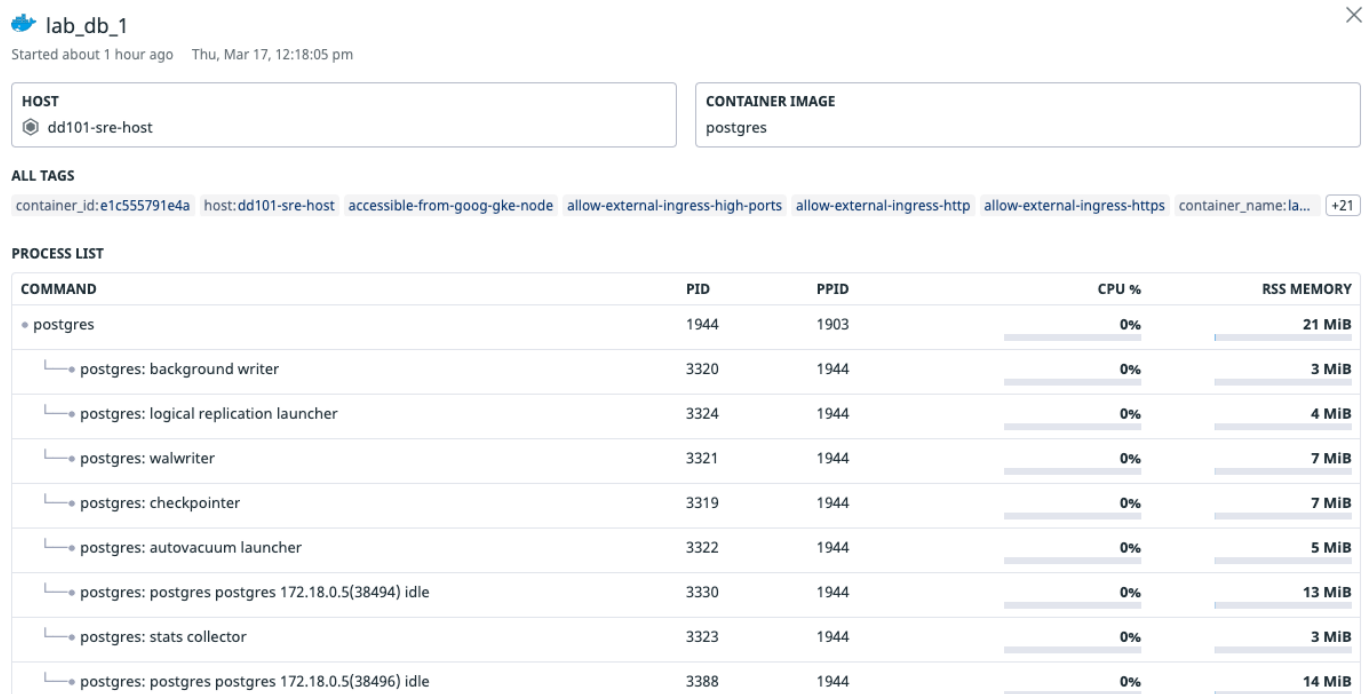
First, take a look at what Datadog knows about the PostgreSQL out-of-the-box:

1. In the terminal, run this command:

```
docker-compose ps
```

You can see that the Storedog containers are already running, and the container named `lab_db_1` is running PostgreSQL.

2. In Datadog, navigate to **Infrastructure > Containers**. Click on container `lab_db_1` to examine its details in the side panel:



lab_db_1
Started about 1 hour ago Thu, Mar 17, 12:18:05 pm

HOST
dd101-sre-host

CONTAINER IMAGE
postgres

ALL TAGS
container_id:e1c55791e4a host:dd101-sre-host accessible-from-goog-gke-node allow-external-ingress-high-ports allow-external-ingress-http allow-external-ingress-https container_name:la... +21

PROCESS LIST

COMMAND	PID	PPID	CPU %	RSS MEMORY
postgres	1944	1903	0%	21 MiB
postgres: background writer	3320	1944	0%	3 MiB
postgres: logical replication launcher	3324	1944	0%	4 MiB
postgres: walwriter	3321	1944	0%	7 MiB
postgres: checkpointer	3319	1944	0%	7 MiB
postgres: autovacuum launcher	3322	1944	0%	5 MiB
postgres: postgres postgres 172.18.0.5(38494) idle	3330	1944	0%	13 MiB
postgres: stats collector	3323	1944	0%	3 MiB
postgres: postgres postgres 172.18.0.5(38496) idle	3388	1944	0%	14 MiB

The Agent can see what processes are running in the db container, e.g. `postgres: stats collector`, but it needs some help identifying this as a known integration.

3. Navigate to **Logs > Search**. In the facets panel on the left, filter by the `postgres` service.

Note: If you don't see any entries, increase the timeframe to **Past 1 Hour** in the dropdown at the top of the page.

The logs from the `postgres` service are a bit messy. The Agent collects them as-is, without any special processing. Consequently, many are incorrectly tagged as errors.

4. Click on one of the log entries that has a status of **Error**:

The screenshot shows the Datadog Logs interface. On the left, there's a sidebar with 'Logs' and a search bar containing 'Service:postgres'. Below the search bar are tabs for 'Fields', 'Patterns', and 'Transactions'. A 'List' button is highlighted. The main area shows a timeline with a red bar indicating an error. On the right, a detailed view of a log entry is shown. The entry is an 'ERROR' from 'Mar 17, 2022 at 12:18:30.703'. It includes fields for 'HOST' (dd101-sre-host), 'SERVICE' (postgres), 'SOURCE' (postgres), 'CONTAINER NAME' (lab_db_1), and 'DOCKER IMAGE' (postgres). Below these fields, there's a section for 'ALL TAGS' with various tags like 'accessible-from-goog-gke-node', 'allow-external-ingress-high-ports', etc. The log message itself is '2022-03-17 19:18:16.185 UTC [1] LOG: database system is ready to accept connections'. At the bottom, there's a message: 'No attributes have been extracted from the log message. Set the source value to an integration name to benefit from automatic setup or create a custom pipeline to process this log format.' with a 'Need Help?' button.

There's a helpful note at the bottom of this log line detail:

No attributes have been extracted from the log message. Set the source value to an integration name to benefit from automatic setup or create a custom pipeline to process this log format.

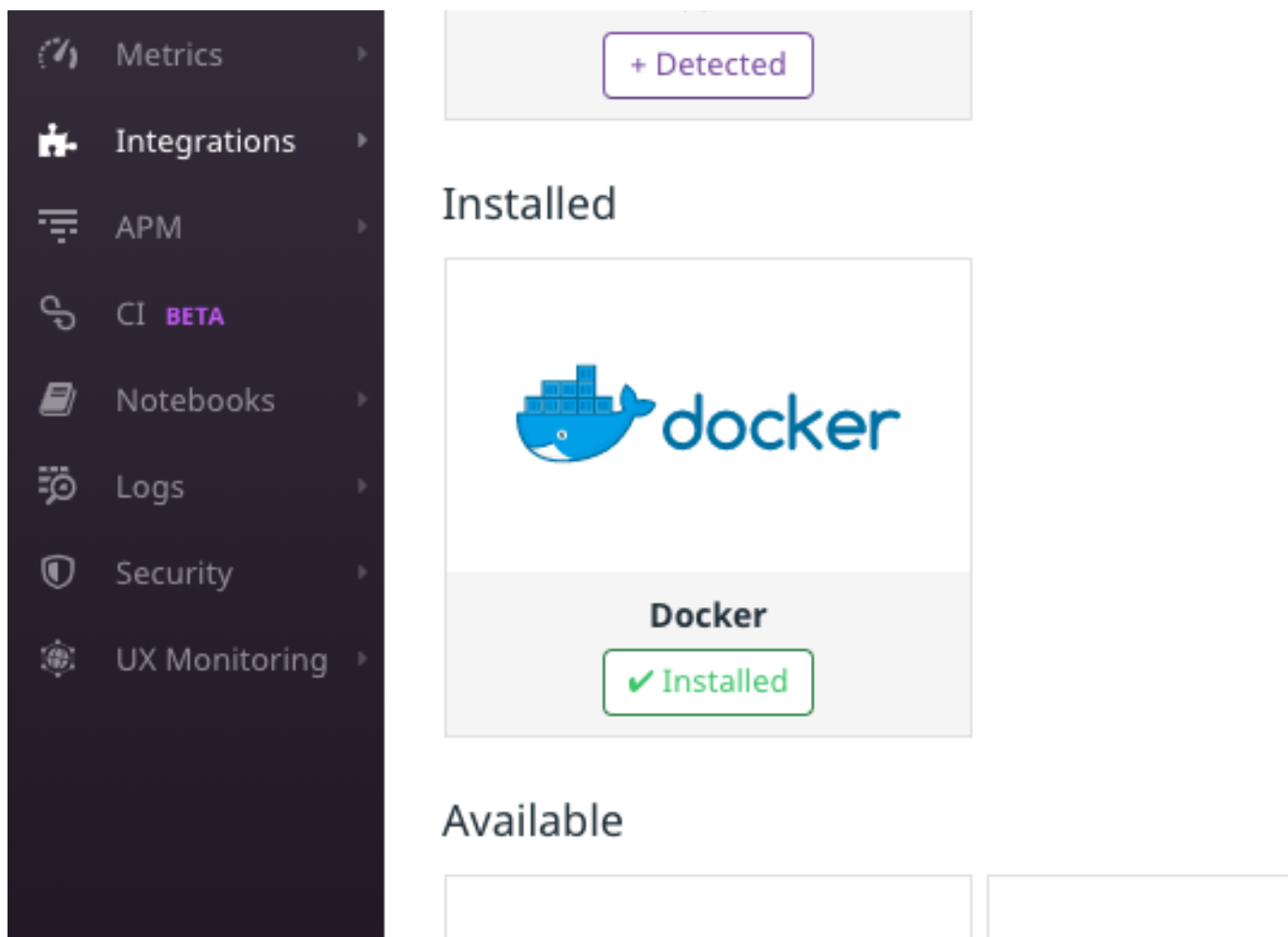
You'll do that shortly.

Furthermore, the `service` in log lines is identified as `postgres`, even though it is tagged as `database` in the `docker-compose.yml` file. This is because the Agent will use a container's `short_image` name for the `service` tag by default. You'll fix that, too.

Viewing existing integrations

Before configuring the PostgreSQL service, take a look at an integration that Datadog has installed for you automatically.

1. Navigate to **Integrations**.
2. Under **Installed**, you should see **Docker**, with an indicator that it's installed. This is one of the integrations that Datadog can install without configuration. The Agent recognizes it as soon as it mounts the `docker.sock` file from the host that we mentioned when starting the lab, and which you configured in "The Agent in a Container" lab previously.



Mounting `docker.sock` in the Agent container also enables Autodiscovery for all the containers that the Docker Daemon is running. The Agent will use each container's Autodiscovery labels to configure checks for it. Autodiscovery label keys start with `com.datadoghq.ad`, as you'll see below.

If you see PostgreSQL listed under **Installed** already, it's likely left over from the "The Agent on a Host" lab, where you configured the PostgreSQL integration in a *host* environment. Integrations will remain installed unless you manually uninstall them by clicking the **Uninstall Integration** button at the bottom of the integration's **Configuration** tab. While technically installed, this integration will not be used again until you configure the *Docker* Datadog Agent in the following steps.

Configure an Integration for a Service

Installing the PostgreSQL integration involves the same configuration you performed in the host environment, but done differently.

1. Navigate to **Integrations**.
2. In the search field, type `postgres` to find the Postgres installation instructions.
3. Click on the PostgreSQL card, and then click on the **Configure** tab.

Under **Prepare Postgres**, there are `psql` commands to create a user for the Agent to query statistics. These commands were already run for you when the lab started.

As you scroll further, you will see the steps you performed in "The Agent on a Host" lab.

4. Scroll down to the **Docker** section and look at the **Metric collection**, **Log collection**, and **Trace collection** instructions. You'll perform these steps next.

Configure the Postgres Integration

To configure the PostgreSQL integration, follow these steps:

1. In the IDE, open `docker-compose.yml` and scroll down to the `db` service section.
2. The following Autodiscovery labels have been adapted for `docker-compose.yml`. Add them to the `labels` block of the `db` service under the comment `# postgres integration template here`:

```
com.datadoghq.ad.check_names: '["postgres"]'  
com.datadoghq.ad.init_configs: '[]'  
com.datadoghq.ad.instances: ' [{"host": "%host%", "port": 5432, "username": "datadog", "password": "datadog"} ]'
```

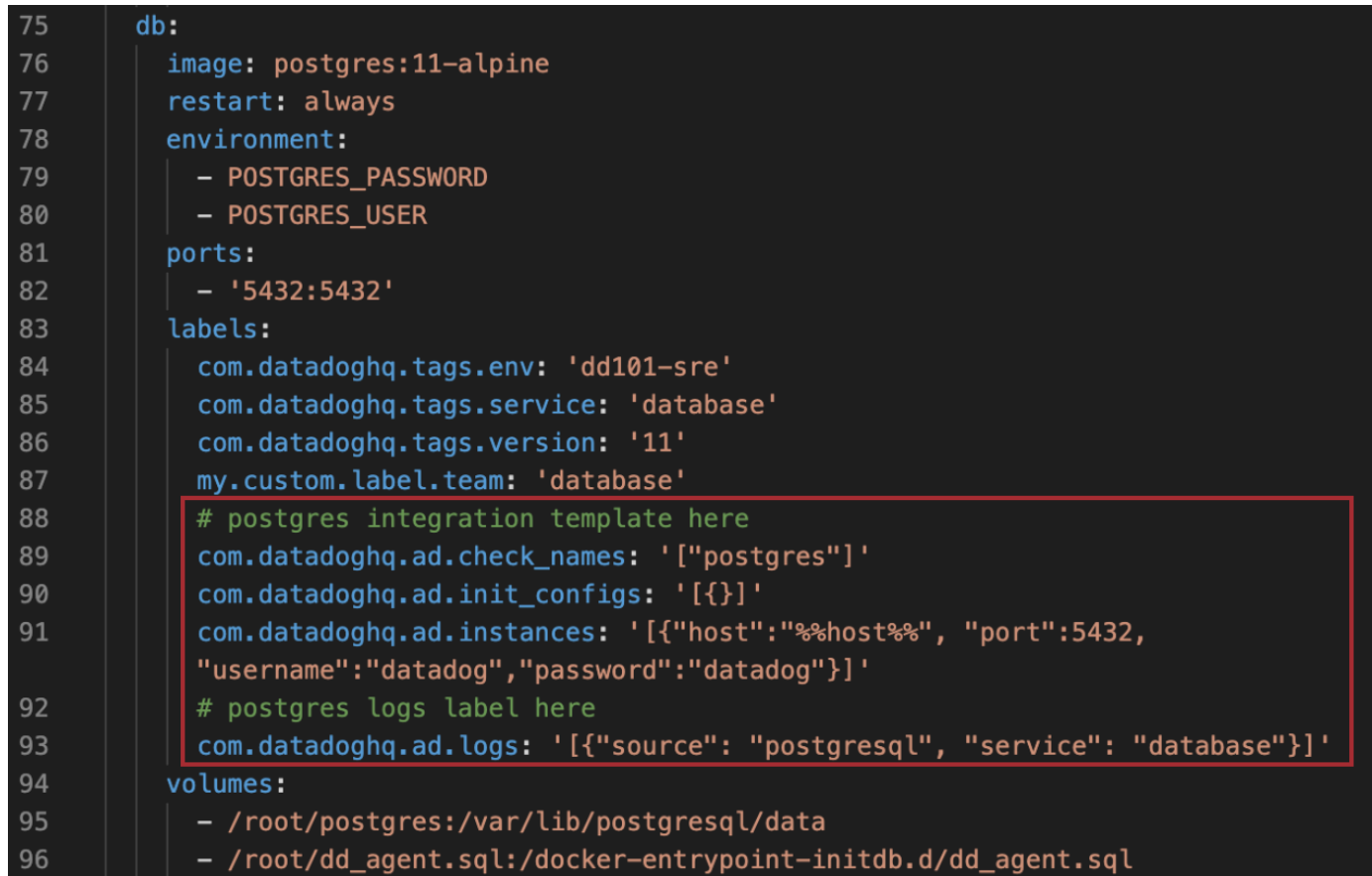
These Autodiscovery labels tell the Agent to run the `postgres` check on this container, and provide the credentials for querying metrics.

3. Add the following line to the same `labels` section under the comment `# postgres logs label here`:

```
com.datadoghq.ad.logs: ' [{"source": "postgresql", "service": "database"} ]'
```

This tells Datadog to use the PostgreSQL integration's log pipeline to parse this service's logs more intelligently, and to tag the log lines with `service:database`.

Make sure that it lines up with the other labels as shown in this screenshot:



```
75 db:  
76   image: postgres:11-alpine  
77   restart: always  
78   environment:  
79     - POSTGRES_PASSWORD  
80     - POSTGRES_USER  
81   ports:  
82     - '5432:5432'  
83   labels:  
84     com.datadoghq.tags.env: 'dd101-sre'  
85     com.datadoghq.tags.service: 'database'  
86     com.datadoghq.tags.version: '11'  
87     my.custom.label.team: 'database'  
88     # postgres integration template here  
89     com.datadoghq.ad.check_names: '["postgres"]'  
90     com.datadoghq.ad.init_configs: '[]'  
91     com.datadoghq.ad.instances: ' [{"host": "%host%", "port": 5432,  
92     "username": "datadog", "password": "datadog"} ]'  
93     # postgres logs label here  
94     com.datadoghq.ad.logs: ' [{"source": "postgresql", "service": "database"} ]'  
95   volumes:  
96     - /root/postgres:/var/lib/postgresql/data  
97     - /root/dd_agent.sql:/docker-entrypoint-initdb.d/dd_agent.sql
```

4. Finally, you need to add an environment variable to the `agent` service. Add the following line to the `environment` block under the comment `# agent non-local apm here`:

```
- DD_APM_NON_LOCAL_TRAFFIC=true
```

This allows the Agent to accept APM traces from other containers. You'll see later how APM traces PostgreSQL through instrumented applications that connect to the database.

See the Results

1. In the terminal, restart the application stack by running the following command:

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

2. After Docker Compose finishes restarting, check the Datadog Agent status by running the following command:

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

3. Scroll up to the **Running Checks** section and find the new **postgres** section:

```
postgres (9.0.2)
-----
Instance ID: postgres:2fc2b794990322e2 [OK]
Configuration Source: docker:docker://c95704a8e6beed69b79332c2ab4722ff807d0af25ce177794f30aad15ffa978c
Total Runs: 14
Metric Samples: Last Run: 15, Total: 30
Events: Last Run: 0, Total: 0
Service Checks: Last Run: 1, Total: 14
Average Execution Time : 14ms
Last Execution Date : 2021-10-01 11:56:17 UTC (1633089377000)
Last Successful Execution Date : 2021-10-01 11:56:17 UTC (1633089377000)
metadata:
  version.major: 11
  version.minor: 12
  version.patch: 0
  version.raw: 11.12
  version.scheme: semver
```

Note: If you don't see the **postgres** section under **Running Checks**, wait a few seconds and run the Agent **status** command again.

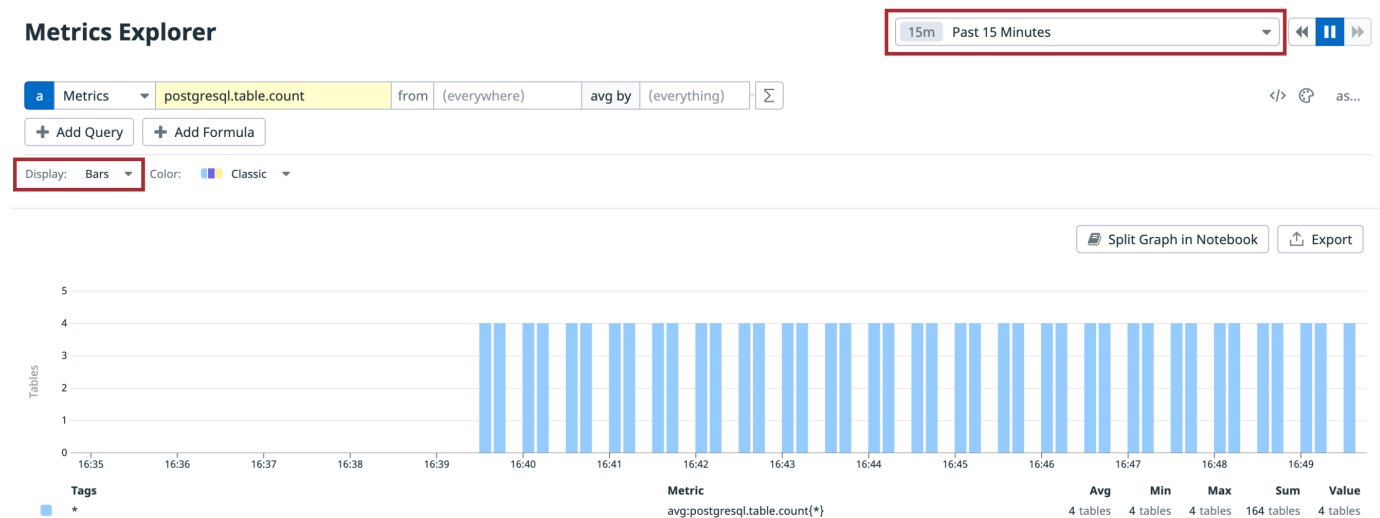
4. In Datadog, navigate back to **Integrations**. You should see that the Postgres integration is now installed (if it wasn't already).
5. Click the card and open the **Data Collected** tab. These are the new metrics available, thanks to this integration.
6. Navigate to **Metrics > Explorer**.
7. For the **Metrics** query, type the following:

```
postgresql.table.count
```

The resulting graph should show that there are 4 tables.

Note: To visualize it more clearly, try changing the **Display** from **Lines** to **Bars** and changing the timeframe selector to **Past 15 Minutes**.

Metrics Explorer



8. Navigate to **Dashboards > Dashboard List** and find **Postgres - Overview** and **Postgres - Metrics**. These are out-of-the-box Dashboards provided by the integration.
Open them up to see what they provide.
9. Navigate to **Logs > Search** and click the new **database** service facet on the left-hand side to filter the log lines for that service tag. This is the service formerly tagged as **postgres**. You will still see the old **postgres** service tags if you set the timeframe dropdown to **Past 1 Hour** or more.
10. Click on a log line and look at the detail panel. You can see that it's correctly identified as the **database** service and that the **Event Attributes** are formatted as JSON. This is a benefit of the logs pipeline that was installed when Datadog detected this integration.

INFO

Mar 17, 2022 at 15:21:46.495 (4 minutes ago)

View in Context

Export

×

HOST


dd101-sre-host

SERVICE

database

SOURCE

postgresql



CONTAINER NAME

lab_db_1

DOCKER IMAGE

postgres

ALL TAGS

accessible-from-goog-gke-node

allow-external-ingress-high-ports

allow-external-ingress-http

allow-external-ingress-https

container_id:bbdaba3daaa5942e6615d6e17d17b63932f3645c27c4319b39a355c4f30bc9c5

container_name:lab_db_1

datadog.index:main

docker_image:postgres:11-alpine

env:dd101-sre

hostname:dock...

+19

2022-03-17 22:21:46.495 UTC [21] LOG: database system was shut down at 2022-03-17 22:21:40 UTC

📄

Event Attributes

Trace (0)

Metrics

Processes

```

{
  db {
    date      1647555706495
    severity  LOG

    }
    msg      database system was shut down at 2022-03-17 22:21:40 UTC
    postgres {
      proc_id 21
    }
  }
}

```

SERVICE

1-sre-host	database
1-sre-host	database
1-sre-host	database
1-sre-host	database
1-sre-host	database
1-sre-host	database
1-sre-host	database
1-sre-host	database
1-sre-host	database

11. Navigate to **Logs** > **Configuration** and click on the new **Postgresql** pipeline. You'll learn more about how log pipelines work in the "Logs" lab.

Configure All the Services

The remaining Storedog services run either Ruby or Python, both of which have Datadog integrations. However, unlike popular and well-defined applications such as PostgreSQL, Apache, Nginx, and others, Ruby and Python are processes that simply run applications. These applications might be your own, or frameworks such as Rails or Flask. It is up to the application developers to instrument these applications to talk to Datadog using Application Performance Monitoring (APM).

You'll learn more about APM later in this course. For now, you can enable the log pipeline features of the Ruby and Python integrations. This is achieved by adding a single label to each service in the `docker-compose.yml` file.

Label the Discounts Service

Start by adding an Autodiscovery label for the discounts service:

1. In the IDE, open `docker-compose.yml`.
2. Add the following line to the `discounts` service labels under the comment `# discounts log label here:`

```
com.datadoghq.ad.logs: '[{"source": "python", "service": "discounts-service"}]'
```

This label tells Datadog to parse logs from this service using the Python pipeline. It will also tag each log line with `service:discounts-service`.

3. In the terminal, restart the stack by running the following command:

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

Wait for Docker Compose to complete before proceeding.

4. In Datadog, navigate to **Logs > Search** and set the timeframe selector to **Past 15 Minutes**
5. In the **Service** section of the facets panel, click on the new **discounts-service** facet to filter the logs to that service.
6. Click on a **discounts-service** log line to view the details in the side panel:

ERROR Mar 17, 2022 at 15:40:45.739 (1 minute ago) [View in Context](#) [Export](#) [X](#)

HOST
dd101-sre-host

SERVICE
discounts-service

SOURCE
python

CONTAINER NAME
lab_discounts_1

DOCKER IMAGE
ddtraining/discounts

ALL TAGS
accessible-from-goog-gke-node allow-external-ingress-high-ports allow-external-ingress-http
allow-external-ingress-https container_id:5f9b37942f8345839d0548f0d5d3821384a92c72c01bda655268eb37b1b8147c
container_name:lab_discounts_1 datadog.index:main datadog.pipelines:false docker_image:ddtraining/discount... +21

INFO:bootstrap:Discounts available: 206

Event Attributes Trace (0) Metrics Processes

No attributes have been extracted from the log message. Set the source value to [an integration name](#) to benefit from automatic setup or create a custom pipeline to [process](#) this log format.

[Need Help?](#)

You can see that Datadog correctly identified the source by the Python logo in the upper-right corner. Also, the service is now correctly tagged as **discounts-service**, rather than the default **discounts** taken from the container's **short_image** name.

However, the logs are not getting parsed into structured JSON. Looking at the **Logs > Configuration** page, you'll see that the Python pipeline is indeed active.

The problem is that the discounts service doesn't write logs in a standard format. It's up to the application developer to emit logs that Datadog knows how to parse. There are ways around this that you'll learn about in the "Logs" and "APM" labs. The Python integration documentation also provides a solution.

Label All the Services

Now you can add Autodiscovery labels to the other services.

1. In the IDE, open `docker-compose.yml` and add the following line to label the **advertisements** service under the comment `# advertisements log label here`:

```
com.datadoghq.ad.logs: '[{"source": "python", "service": "advertisements-service"}]'
```

2. Add the following line to the **frontend** service under the comment `# frontend log label here`

```
com.datadoghq.ad.logs: '[{"source": "ruby", "service": "store-frontend"}]'
```

Note that the **service** value is the same as each service's `com.datadoghq.tags.service` label value.

3. In the terminal, restart the stack one more time by running the following command:

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

4. After Docker Compose finishes restarting, check the Datadog Agent status by running the following command:

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


5. Scroll up to the **Logs Agent** section and notice that each container you added the `com.datadoghq.ad.logs` label to has its own entry, displaying its individual status and statistics:

```
=====
Logs Agent
=====

Sending compressed logs in HTTPS to agent-http-intake.logs.datadoghq.com on port 443
BytesSent: 746522
EncodedBytesSent: 49336
LogsProcessed: 1360
LogsSent: 1359

docker
-----
- Type: docker
  Status: OK
  Inputs:
    621daa8e292863d8ca905d2d51a66a600ce334767ab8f110e251f8317f822cf5
  BytesRead: 145193
  Average Latency (ms): 38
  24h Average Latency (ms): 38
  Peak Latency (ms): 256
  24h Peak Latency (ms): 256
- Type: docker
  Status: OK
  Inputs:
    81d16ca05ad2fe9edd07f80e351f405574c9db12d09638cf7c7d71ee2645416a
  BytesRead: 13466
  Average Latency (ms): 0
  24h Average Latency (ms): 0
  Peak Latency (ms): 6
  24h Peak Latency (ms): 6
- Type: docker
  Status: OK
  Inputs:
    a8f527f56c855260f883496c543cc05288b4d79717e7fc54b0fb6f44fc6deca0
  BytesRead: 15142
  Average Latency (ms): 0
  24h Average Latency (ms): 0
  Peak Latency (ms): 1
  24h Peak Latency (ms): 1
- Type: docker
  Status: OK
  Inputs:
    f6757f91bb8e2f7540250e3d9a19ec12d6be14b0b2ce0fa932235f465b9e35dc
  BytesRead: 1266
  Average Latency (ms): 0
  24h Average Latency (ms): 0
  Peak Latency (ms): 0
  24h Peak Latency (ms): 0

container_collect_all
-----
- Type: docker
  Status: OK
  Inputs:
    c2429104daf6c552e92b4fa920d5a673d2176080d3c3501f25566ad0bb126111
    576ddd668a6a6a8fd8bfc42cfb244dffcc00351d4cb60ea5a914b23a1d8893d65
  BytesRead: 54353
  Average Latency (ms): 0
  24h Average Latency (ms): 0
```

Previously, these containers had been grouped under `container_collect_all` with aggregated statistics.

6. You can confirm this by comparing the value listed under `Inputs:` with the `CONTAINER ID` values displayed by this

command.

```
docker ps
```

7. Back in Datadog, navigate to **Integrations** to see the newly-installed Python and Ruby integrations.
8. Navigate to **Dashboards > Dashboard List** and examine the new **Python Runtime Metrics** and **Ruby Runtime Metrics** dashboards. They're not graphing anything yet, but they will when you configure APM later in the course.
9. Navigate to **Logs > Search** and look at the latest **advertisements-service** and **store-frontend** log lines.

Click on the log entries to observe the details. The log lines are now correctly identified as Python and Ruby sources, and tagged with the correct **service:** tags.

As with the **discounts-service**, the Python and Ruby log pipelines are activated for these services, but the log output is not formatted uniformly. The **store-frontend** logs will look much better when you configure APM later in the course.

Lab Conclusion

Congratulations! You learned how to install Datadog integrations. You also observed the label values represented as tags in the Datadog App, in both the container details and log entries.

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.