Install the Agent

Explore Storedog

Open the Storedog e-commerce application by clicking the **Storedog** tab above the terminal.

This application is served from an Nginx container running in your lab environment. It might take a little while to load the first time.

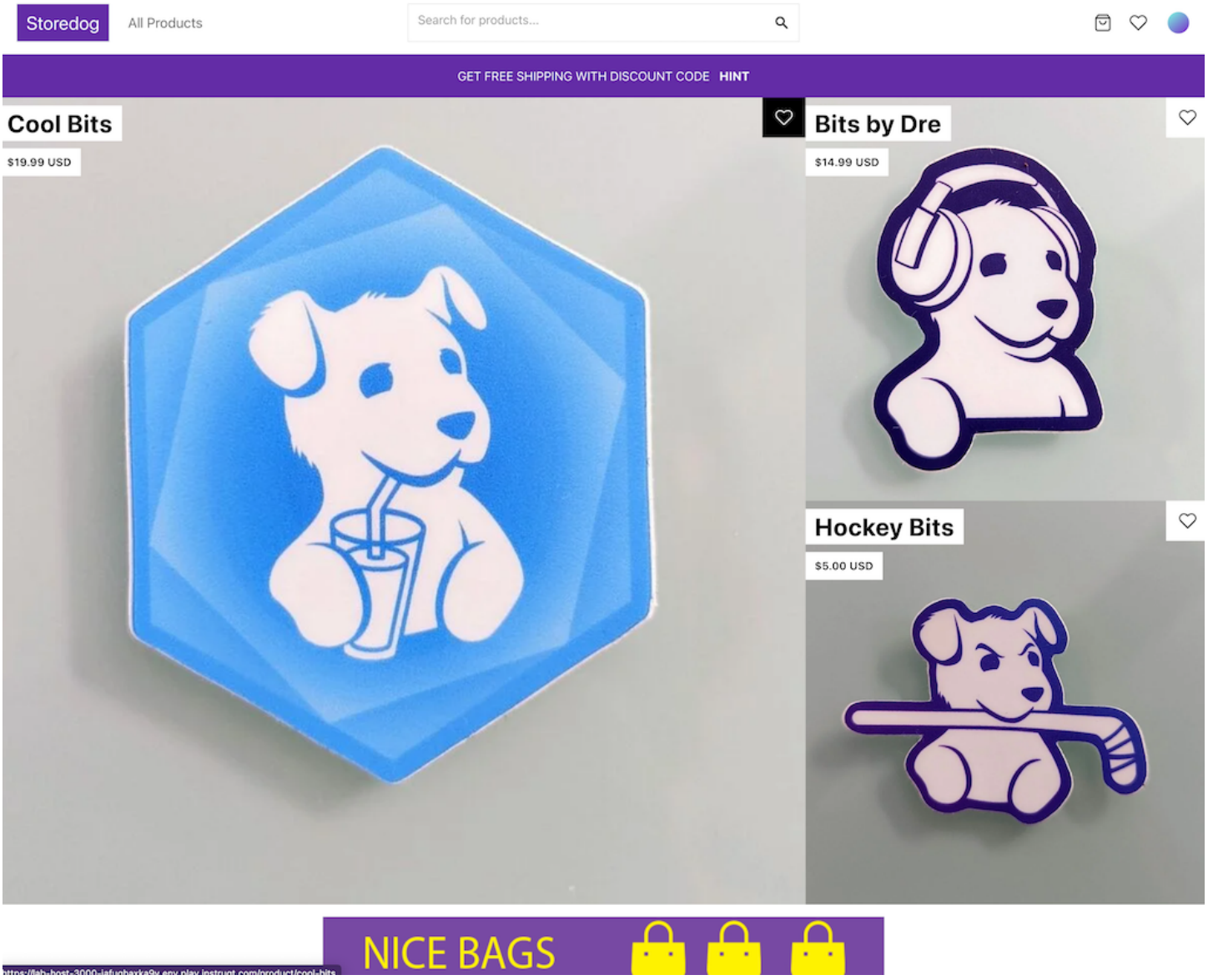


Figure 1: A screenshot of the Storedog web application home page

Storedog is composed of several services running in Docker containers:

- Nginx web server
- NextJS frontend

- Ruby on Rails e-commerce backend and worker
- Python Flask microservice for discount codes
- Python Flask microservice for advertisements

There are also two services that are installed directly on the host:

- Redis cache
- A PostgreSQL database for the backend and microservices

That's a lot of moving pieces for a little web store! But don't worry about remembering the architecture because the Agent will help you keep track of everything.

> **Note:** If you want to see how these services are configured, you can click on the IDE tab and examine the file `docker-compose.yml`.

Feel free to click around to get familiar with Storedog before continuing.

Install the Agent

1. Log in to Datadog using the Datadog training account credentials displayed in the terminal.
2. Navigate to **Integrations** > **Agent** and click on **Ubuntu**.
3. Click the **Select API Key** button and then select your API key. The last four digits of the **Key** (not **Key ID**) should match your Datadog training account credentials.

   Select **Use API Key** to have your API Key automatically populated in the following **one-step install** command.
4. **Important**. Toggle off **Enable APM Instrumentation**:

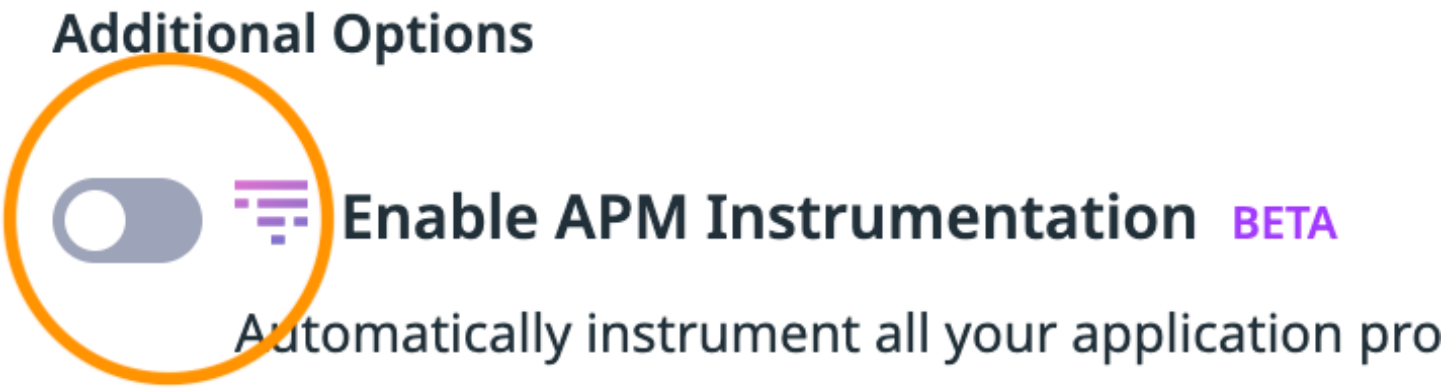


Figure 2: Toggle off APM Instrumentation

   This is a new feature that simplifies instrumenting your applications without altering their code. In this lab, all the services have been manually instrumented with `dd_trace` libraries, so auto-instrumentation is unnecessary.
5. Copy the full, **one-step install** command to install the Agent by clicking the copy icon to the right of the command:
6. Back in the lab terminal, paste the command and press Enter to execute it.

   It should take about a minute to fully install the Agent. When complete, you will see a confirmation message:

In the next challenge you'll use the Agent's command-line interface to ensure that it's running as expected.

Click **Check** to move on to the next activity.

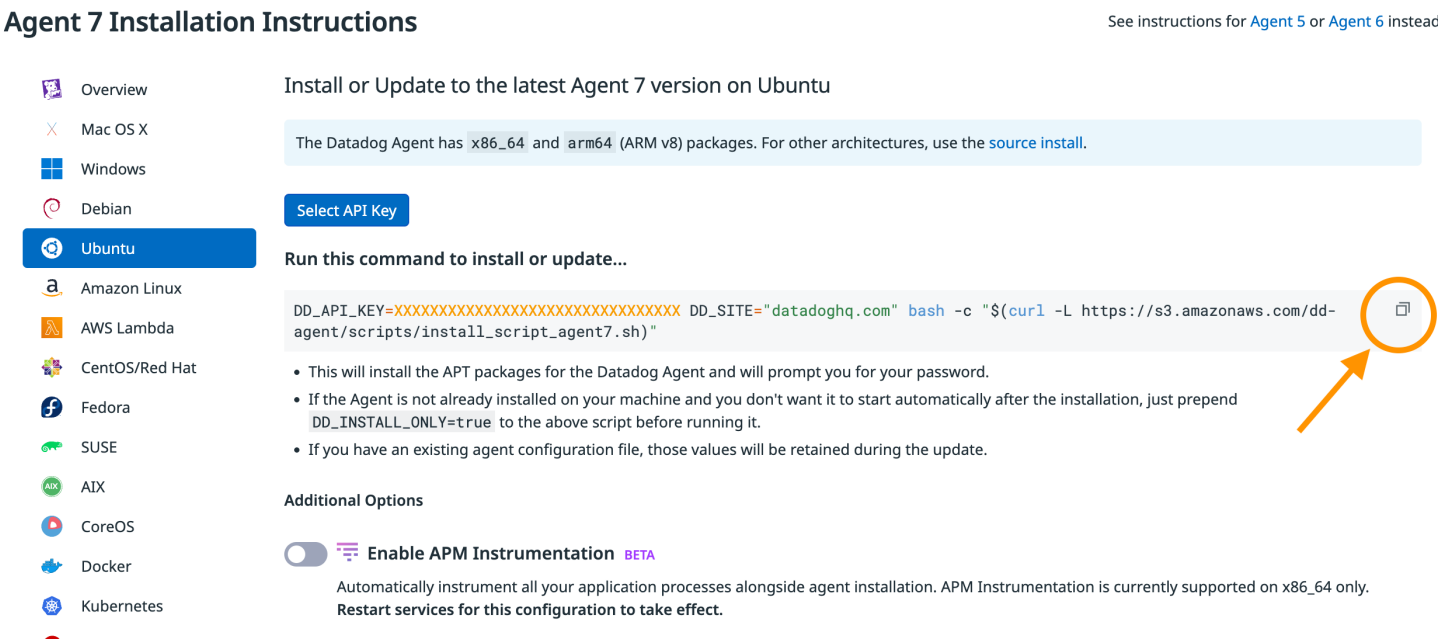


Figure 3: Easy one-step Agent install for Ubuntu

```
Your Datadog Agent is running and functioning properly.
It will continue to run in the background and submit metrics to Datadog.
If you ever want to stop the Datadog Agent, run:

     systemctl stop datadog-agent

And to run it again run:

     systemctl start datadog-agent
```

Figure 4: Confirmation message that Agent is installed

Agent Commands

Now that you have installed the Datadog Agent on the VM running Storedog's services, you can run some commands to inspect its status and configuration.

Agent `status`

The Agent `status` command displays the Agent's status including the checks it's running, the logs it's collecting, and more.

1. To verify that the Agent is running and sending data to Datadog, run the following command:

   ```
   datadog-agent status
   ```

2. Scroll up to the **Forwarder** section of the command output. The Forwarder is the process that sends everything to Datadog.
3. Near the bottom of this section, find the **Transaction Successes** block. This displays the transactions that the Agent has completed with Datadog API endpoints.

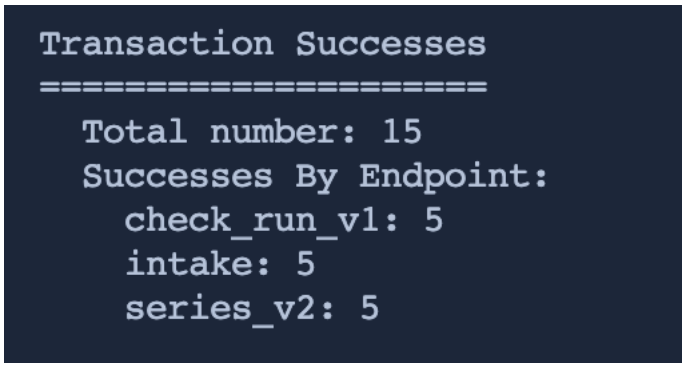


Figure 1: The Transaction Successes block of the Agent status output

   Note: If you don't see this block, run the Agent `status` command again. The Agent might need more time to start communicating with Datadog.

4. Find the last block of the **Forwarder** section, **API Keys status**. It displays the last few characters of your Datadog API key, which was included in the installation command.

Comparing the API key reported by the Agent with those in your Datadog account is a good way to confirm that you're sending data to the right account.

You'll run the `status` command a few more times in this lab as you configure the Agent.

Agent `config`

The Agent `config` command lists the resolved values of every Agent configuration parameter. It includes default values and those set using environment variables.

1. Run the Agent `config` command:

   ```
   datadog-agent config
   ```

   Scroll through the list a bit to get a sense of the Agent's configuration parameters. Most of these parameters are documented in the Agent's configuration template, which you can find in its GitHub repository.

2. Ensure that the Agent will tag all metrics, events, and logs with the correct environment tag. For this lab, it's `agent-host-lab`. Run the following command:

   ```
   datadog-agent config|grep ^env
   ```

   You should see `env: agent-host-lab`. This value originates from an environment variable set in your shell. You can see it by running the following command:

   ```
   env |grep DD_ENV
   ```

   This environment variable was detected by the Agent installer and written to the Agent's configuration. The configuration parameters that can be set using environment variables are documented in the `datadog.yaml` file that you'll work with in the next activities.

As you can see, `config` is a useful command for ensuring that the Agent is configured as expected.

Click **Next** to extend the Agent's capabilities by modifying its configuration file.

Configure Live Processes

The Datadog Agent is configured by editing YAML files. On Linux systems, these files are located in `/etc/datadog-agent` by default. These files were created when you installed the Agent.

> **Note**: In this lab, there is a symbolic link to the Datadog Agent configuration directory at `/root/lab/datadog-agent`. This provides convenient access to the directory in the IDE.

The file that configures every aspect of the Datadog Agent is `datadog.yaml`. It contains hundreds of configuration options and their defaults. In this section, you'll update this file to set an explicit hostname and to enable logging.

Live Processes

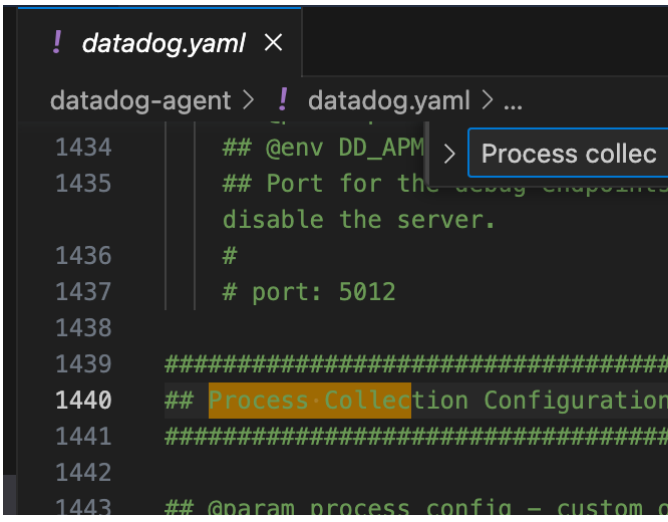
Datadog's Live Processes gives you real-time visibility into the processes running on your infrastructure, presenting them in one place in the Datadog app.

You'll now get familiar with `datadog.yaml` as you enable Live Processes.

1. In the IDE file explorer, expand the `datadog-agent` directory.
2. Open `datadog.yaml`.

   Notice that your API key is already configured on line 11. The one-step Agent installation command you ran included your API key, and the installation process wrote it into this file for you.
3. Scroll down to the **Process Collection Configuration** section, about midway down the file.

   Note: You can use Ctrl/Cmd+F in the lab IDE to search for text.



4. Find the line `# process_config:`.

   Notice the verbose comments in this file. This is typical of Datadog configuration files. If a parameter is commented out (that is, begins with a `#` character), then its default value will be used. Default values are identified in the comments.
5. Remove the `#` character and the extra space in front of `process_config:`
   This will "un-comment" the option and allow you to set a non-default value.

   The IDE will automatically save your changes.
6. The option that enables Live Processes is `process_collection`. Un-comment the `process_collection` line and the `enabled` line.

   Then change `enabled: false` to `enabled: true`.

7. Confirm this section of your the `datadog.yaml` file looks like the following:

```
#####################################
## Process Collection Configuration ##
#####################################

## @param process_config - custom object - optional
## Enter specific configurations for your Process data collection.
## Uncomment this parameter and the one below to enable them.
## See https://docs.datadoghq.com/graphing/infrastructure/process/
#
process_config:

  ## @param process_collection - custom object - optional
  ## Specifies settings for collecting processes.
  process_collection:
    ## @param enabled - boolean - optional - default: false
    ## Enables collection of information about running processes.
    enabled: true
```

Figure 1: Process collection is enabled in the Datadog configuration file

You must restart the Agent for configuration changes to take effect.

1. In the terminal, run the following command:

   `systemctl restart datadog-agent`

2. You can use `status` and `config` to confirm that Live Processes are now enabled.

   Run the `status` command and scroll to the **Process Agent** section. Locate the **Collector** section below it:

   `datadog-agent status`

```
=========
Collector
=========
  Last collection time: 2023-10-02 01:09:37
  Docker socket: /var/run/docker.sock
  Number of processes: 123
  Number of containers: 0
  Process Queue length: 0
  RTProcess Queue length: 0
```

Figure 2: Status output with processes

   Number of processes displays the number of host processes that the Agent has identified. This should be well above zero.

3. If the Agent `status` reports something surprising, you can view the resolved configuration values using the `config` command.

```
datadog-agent config  |grep process_collection -A1
```

```
root@agent-host:~/lab# datadog-agent config  |grep process_collection -A1
  process_collection:
    enabled: true
```

Figure 3: Agent status output confirms that process collection is enabled

If `process.collection.enabled` were `false`, you could start troubleshooting by reviewing `datadog.yaml` to confirm that your changes were saved.

Now that you know how to configure the Agent, you'll configure more useful features.

Click **Next** to configure logging and Docker container metrics.

Configure logging and Docker

Configure Logging

Logging is not enabled by default, so you will edit `datadog.yaml` to enable logging on this host.

1. In the IDE file explorer, expand the `datadog-agent` directory and open `datadog.yaml`.

   Find the section titled **Log Collection Configuration** around line 1033.

2. Locate the line with `# logs_enabled: false`.

   Un-comment the line by removing the leading `#` and space characters and replace `logs_enabled: false` with `logs_enabled: true`.

Most of Storedog's services are running in Docker containers. The Agent won't collect these logs by default, so you must enable this feature.

1. Below the `logs_enabled` line you un-commented in the previous step, find and un-comment `logs_config`.
2. Un-comment `container_collect_all` and set it to `true`.
3. Confirm that this section of `datadog.yaml` looks like the following:
4. In the terminal, run the following command to restart the Agent:

   ```
   systemctl restart datadog-agent
   ```

5. Wait a minute and then run the Agent `status` command:

   ```
   datadog-agent status
   ```

   > **Note**: If running the `status` command results in an error, the error message should indicate the problematic line. Return to `datadog.yaml` in the IDE and confirm that the indication is correct.

6. Scroll through the status output to find the **Logs Agent** section. Notice that it is now populated:

   But what's up with `BytesSent: 0`? There should be lots of logs coming from the containers. You'll fix that next.

Configure Docker

Logs

The Agent collects metrics, events, and logs from containers by reading the Docker daemon socket, `docker.sock`. This also gives it full access to metrics about the Docker daemon.

The Agent runs as the system user `dd-agent` with only the permissions necessary to perform its Agent duties. By default, Docker resources are only accessible to members of the `docker` group. The solution is to add the user `dd-agent` to the group `docker`.

1. In the terminal, run the following command:

   ```
   usermod -a -G docker dd-agent
   ```

2. Restart the Agent:

   ```
   systemctl restart datadog-agent
   ```

3. Wait a bit and then run the `status` command again.

   Find the **Logs Agent** section and notice that it now has a value for `BytesSent`, and that the containers it's collecting from are listed under the **docker** section of **Integrations**:

```
###################################
## Log collection Configuration ##
###################################

## @param logs_enabled - boolean - optional - default: false
## @env DD_LOGS_ENABLED - boolean - optional - default: false
## Enable Datadog Agent log collection by setting logs_enabled to
true.
#
logs_enabled: true

## @param logs_config - custom object - optional
## Enter specific configurations for your Log collection.
## Uncomment this parameter and the one below to enable them.
## See https://docs.datadoghq.com/agent/logs/
#
logs_config:

  ## @param container_collect_all - boolean - optional - default:
  false
  ## @env DD_LOGS_CONFIG_CONTAINER_COLLECT_ALL - boolean - optional
  - default: false
  ## Enable container log collection for all the containers (see
  ac_exclude to filter out containers)
  #
  container_collect_all: true
```

Figure 1: Confirm the log collection configuration

```
==========
Logs Agent
==========
    Reliable: Sending compressed logs in HTTPS to agent-http-intake.logs.datadoghq.com on port 443
    BytesSent: 0
    EncodedBytesSent: 0
    LogsProcessed: 0
    LogsSent: 0
    CoreAgentProcessOpenFiles: 26
    OSFileLimit: 524288
```

Figure 2: Logs Agent section populated in status output

```
============
Integrations
============

docker
------
  - Type: docker
    Service: discounts-service
    Source: python
    Status: OK
      The log file tailer could not be made, falling back to socket
    Inputs:
      02deb2d1a5109ef1ba3830cc06bb193776ae8526602225b784af5f202fe441f7
    Bytes Read: 4159268
    Pipeline Latency:
      Average Latency (ms): 8
      24h Average Latency (ms): 8
      Peak Latency (ms): 148
      24h Peak Latency (ms): 148
  - Type: docker
    Service: worker-service
    Source: ruby
    Status: OK
      The log file tailer could not be made, falling back to socket
    Inputs:
      417d4bb60cb287e1dc5168fe41520f486023a683920856bf0f9b2dc2779c1417
    Bytes Read: 21784
```

Figure 3: Agent status displaying collected container logs

You are now collecting metrics, events, and logs from the Storedog services running in containers.

APM traces

Out of the box, the Datadog Agent is ready to accept APM traces from instrumented applications on localhost. However, containers run on their own network. You'll configure the Agent to accept traces from other networks.

1. Open `datadog.yaml` in the IDE and find `apm_config:`. Un-comment this parameter.

   That line should look like this:

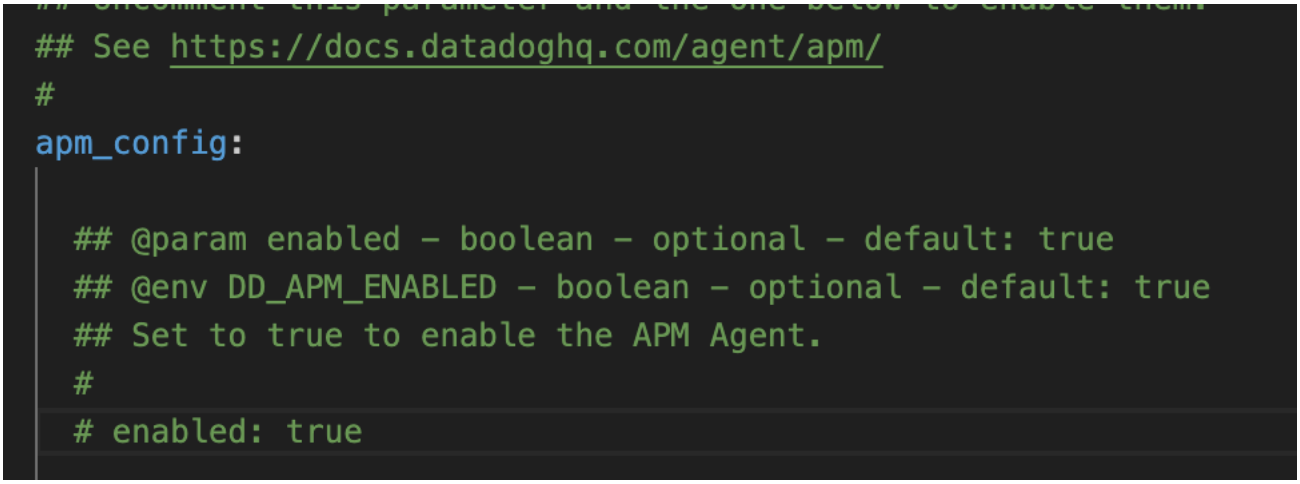


Figure 4: Un-commented apm_config

   Notice that, unlike the `logs_config` section, the default value for `enabled` is `true`. You don't need to change this.

2. Under `apm_config:`, find `apm_non_local_traffic`. Un-comment this parameter and set it to `true`.

   That line should look like this:

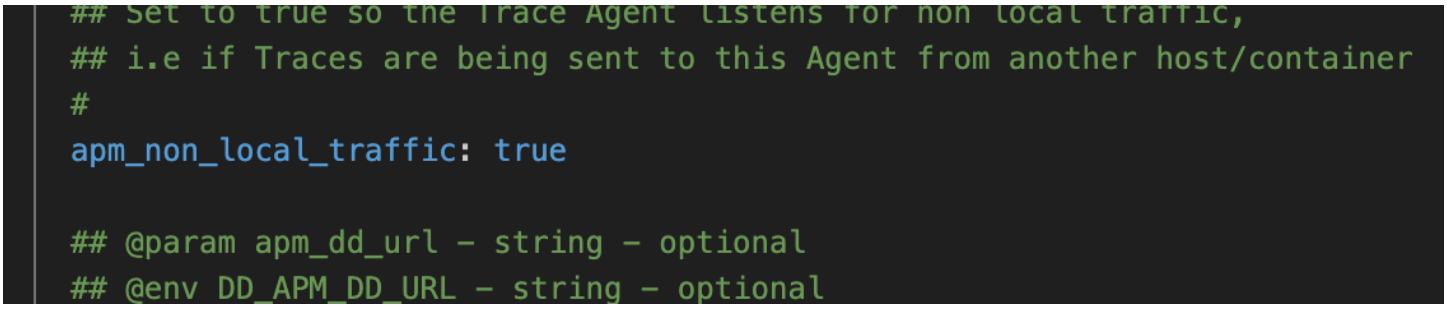


Figure 5: Enable accepting APM traces from non-local traffic

3. Restart the Agent:

   ```
   systemctl restart datadog-agent
   ```

   The Storedog services are already instrumented with `ddtrace` libraries, so they have been trying to send traces to an Agent all this time. Now the Agent will finally receive them.

4. Confirm that the Agent is receiving traces with the `status` command:

   ```
   datadog-agent status
   ```

   Scroll up to the **APM Agent** section, and notice that it's now receiving Ruby and Python traces:

   You might have to wait a couple minutes and run `status` again to see both languages under **Receiver**.

That takes care of all the configuration for the host, Docker, and the Docker containers.

Click **Next** to enable Agent checks for other services that are running directly on the host.

```
=========
APM Agent
=========
  Status: Running
  Pid: 25638
  Uptime: 105 seconds
  Mem alloc: 16,334,112 bytes
  Hostname: agent-lab-host
  Receiver: 0.0.0.0:8126
  Endpoints:
    https://trace.agent.datadoghq.com

  Receiver (previous minute)
  ==========================
    From python 3.9.6 (CPython), client 1.7.2
      Traces received: 1 (2,003 bytes)
      Spans received: 15

    From ruby 3.0.3 (ruby-x86_64-linux), client 1.4.1
      Traces received: 351 (4,649,986 bytes)
      Spans received: 5419


    Priority sampling rate for 'service:store-backend,env:agent-host-lab': 100.0%
    Priority sampling rate for 'service:store-worker,env:agent-host-lab': 100.0%

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

Figure 6: APM Agent is receiving Python and Ruby traces

Configure checks

Agent checks

The Datadog Agent runs a core set of checks by default. These include `cpu`, `disk`, `memory`, `uptime`, and more. When you run the `datadog-agent status` command, these appear in the **Collector > Running Checks** section of the output.

Each of these checks has a corresponding configuration file located in a subdirectory of `datadog-agent/conf.d/`.

1. To see a concise list of the checks the Agent is running and their configuration file paths, run the following command in the terminal.

   ```
   datadog-agent configcheck
   ```

   You should see the **Configuration source** for each of the checks. Notice the subdirectory and filename, such as the following for **disk check**:

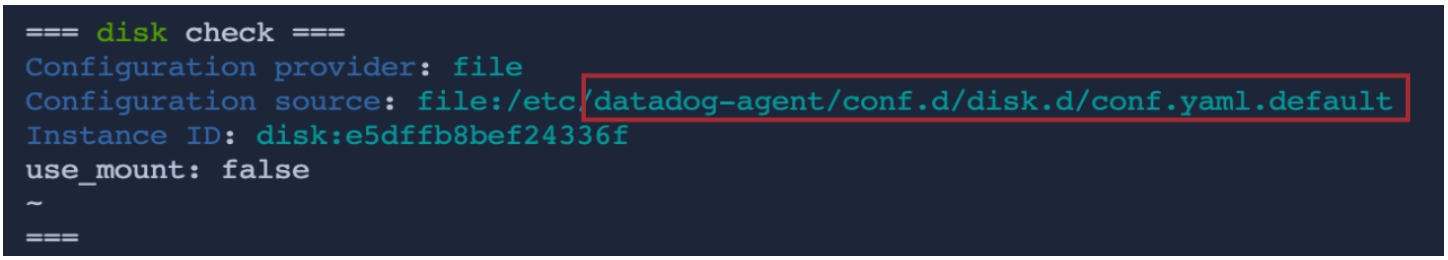


Figure 1: Agent configcheck

2. In the IDE file explorer, open that file, `datadog-agent/conf.d/disk.d/conf.yaml.default`.

   Note: Configuration files that end with `.default` are core Agent checks that run automatically.

   As in the main Agent configuration file, the default configuration values for the `disk` check are listed in this file. If you want to customize the `disk` check for your system, you would un-comment an option and change its value.

   For example, if you wanted the Agent to check only physical storage devices, you would un-comment `include_all_devices` and set its value to `false`.

   Note: Like the Agent configuration, if you update Agent check configuration files, you must restart the Agent for those changes to take effect.

All the checks in `datadog-agent/conf.d/` are known as Agent-based Integrations, a subset of the massive Datadog Integrations ecosystem. As such, you can find documentation for all of them in the Integrations documentation.

If there is a check that you would like the Agent to run that is not provided by a Datadog Integration, you can create your own. See the Writing a Custom Agent Check documentation for details.

Configure the Redis check

Now you'll learn how to configure an Agent Integration for a service running on the same host, Redis.

1. First, search for the Redis documentation on the main Integrations docs page:

1. Click the **Redis** card to see its documentation.

   Overview describes what the integration will provide, and **Setup** explains how to install and configure the integration. Scroll further down the page to **Data Collected** to see the metrics that the integration will collect.

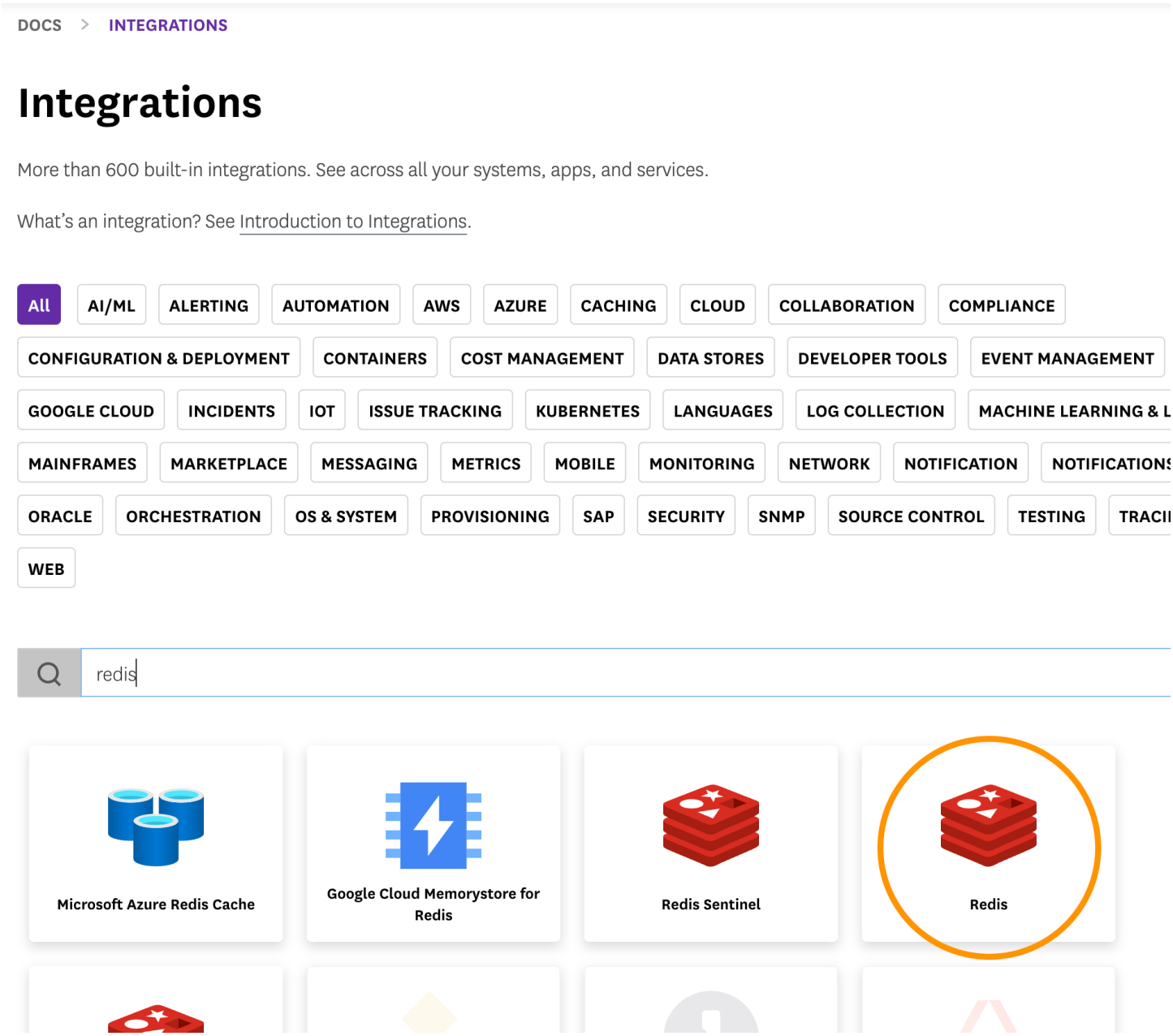


Figure 2: Searching for redis on the main integration docs page

The following steps will walk you through the Setup instructions.

2. In the IDE file explorer, open the directory `datadog-agent/conf.d/redisdb.d/`.

   This directory contains two files, `auto-conf.yaml` and `conf.yaml.example`.

   `auto-conf.yaml` is used for Autodiscovery, which is a feature useful for dynamic environments such as containers. You can ignore that file in this lab.

   You'll work with `conf.yaml.example`.

3. The Agent will run any check that has a `conf.yaml` file in its `conf.d` subdirectory.

   Right-click on `conf.yaml.example` and select **Rename**:

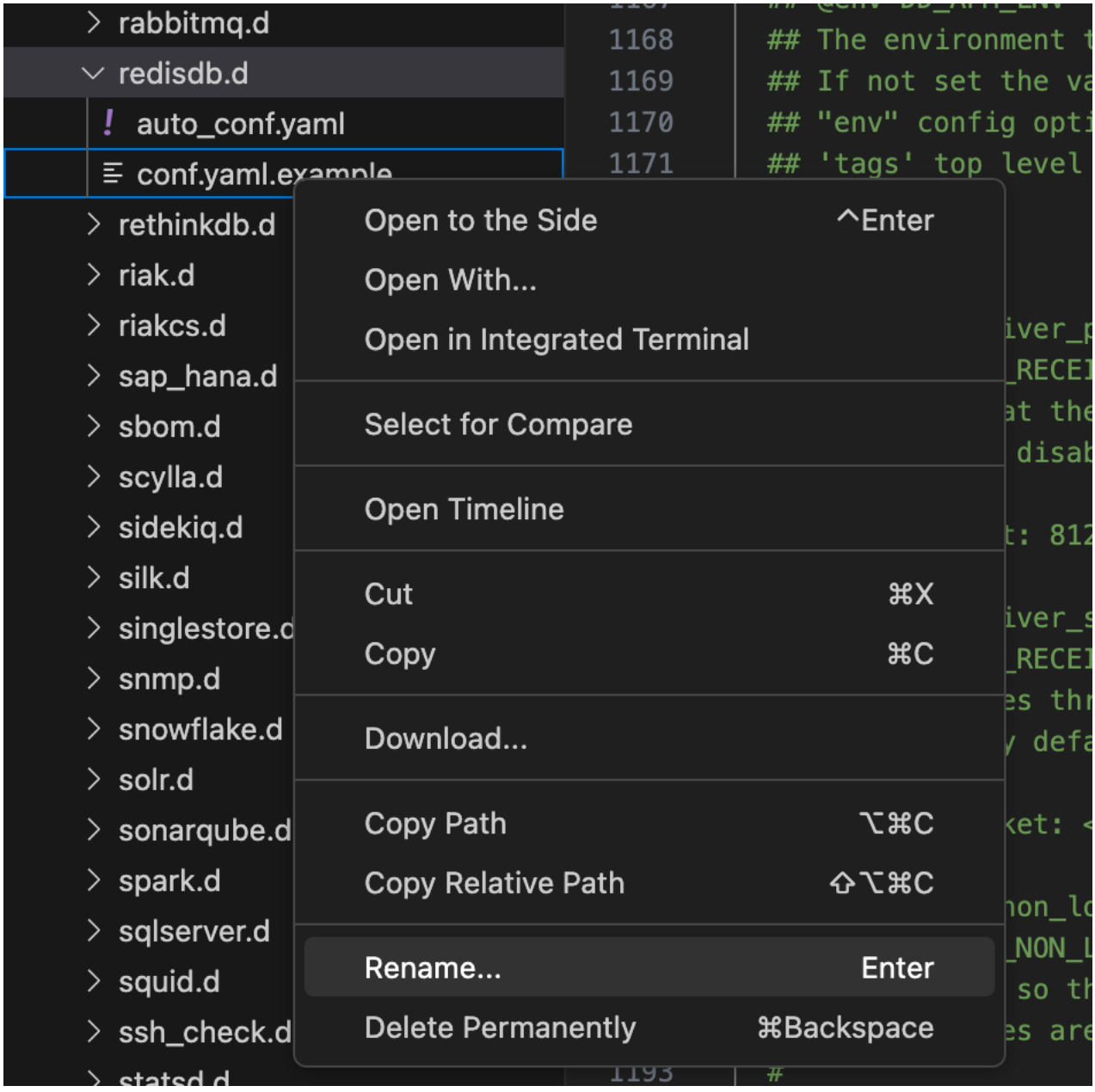


Figure 3: Renaming the redis conf.yaml.example file in the IDE file explorer

   Remove `.example` so the file name becomes `conf.yaml`. This is how you enable checks

4. Open `conf.yaml`. Notice that it is well documented like `datadog.yaml`.

5. Name this service so you can identify its metrics.

   Under `init_config:`, un-comment `service:` and replace `<SERVICE>` with `redis`.

6. The Redis server in this lab isp protected by a password. The Agent will use this password to connect to Redis for metric collection.

   Un-comment `password:` and change `<PASSWORD>` to the following:

   `[[ Instruqt-Var key="LABVAR_REDIS_PASSWORD" hostname="agent-host"]]`

7. Scroll to the bottom of the file and un-comment the entire `logs:` section.
8. Update `path` to the path of the Redis log on this host:

   `/var/log/redis/redis-server.log`

9. Update `service` to `redis`.
10. Confirm that the `logs:` section looks like this:

```
## Discover Datadog log collection: https://docs.data
#
logs:
  - type: file
    path: /var/log/redis/redis-server.log
    source: redis
    service: redis
```

Figure 4: Redis configuration logs block

11. By default, redis logs are readable only by the `redis` user and `adm` group.

    In the terminal, add the Datadog Agent user to the `adm` group by running the following command:

    `usermod -a -G adm dd-agent`

12. Restart the Agent:

    `systemctl restart datadog-agent`

13. Run the `status` command and scroll up to the **Checks** section. You should see that the `redisdb` check is running OK:

```
redisdb (4.8.0)
---------------
  Instance ID: redisdb:590592b5ba6697d9 [OK]
  Configuration Source: file:/etc/datadog-agent/conf.d/redisdb.d/conf.yaml
  Total Runs: 3
  Metric Samples: Last Run: 54, Total: 158
  Events: Last Run: 0, Total: 0
  Service Checks: Last Run: 2, Total: 5
  Average Execution Time : 131ms
  Last Execution Date : 2023-10-03 02:42:35 UTC (1696300955000)
  Last Successful Execution Date : 2023-10-03 02:42:35 UTC (1696300955000)
  metadata:
    version.major: 5
    version.minor: 0
    version.patch: 7
    version.raw: 5.0.7
    version.scheme: semver
```

Figure 5: Agent status command shows that the redis check is running

14. Scroll down to the **Logs Agent** section and confirm that the Agent is collecting **redisdb** logs:

```
redisdb
-------
  - Type: file
    Path: /var/log/redis/redis-server.log
    Service: redis
    Source: redis
    Status: OK
    Inputs:
      /var/log/redis/redis-server.log
    Bytes Read: 0
    Pipeline Latency:
      Average Latency (ms): 0
      24h Average Latency (ms): 0
      Peak Latency (ms): 0
      24h Peak Latency (ms): 0
```

Figure 6: Agent status redis logs

Configure the PostgreSQL check

For your last configuration task, you will configure the PostgreSQL Integration to read the logs of a local PostgreSQL server, and to connect to the server to query metadata. You don't need to focus on the specifics of this Integration, only the concepts.

You can read the docs for the PostgreSQL integration, or rely on the comments in its configuration file.

1. In the IDE file explorer, open the directory `datadog-agent/conf.d/postgres.d/`.
2. Right-click on `conf.yaml.example` and rename it to `conf.yaml`.
3. Around lines 21 to 42 is where you will find the most common configuration parameters for a PostgreSQL database service. You'll see that `host` and `username` are required for the Datadog Agent to connect to the service and query metadata.
4. When this lab started, the `datadog` user was granted access to the local PostgreSQL server with the password `datadog`.

   Update line 36 to remove the comment and replace `<PASSWORD>` with `datadog`:

   Ensure that key `password` is aligned with `username`, as shown in the above screenshot.
5. Scroll to the bottom of the file to find ## **Log Section** around line 539. This is where you will configure the Agent to collect logs for the PostgreSQL service.
6. Un-comment all the lines in the `logs` block from lines 554-562.
7. Set `path` to the following value:

   `/var/log/postgresql/postgresql-12-main.log`

   This is where the PostgreSQL server is writing its logs in the filesystem.
8. Set `service` to `database`. This will add the tag `service:database` to the log entries collected by the Agent.

   Your `logs:` block should look like the following:
9. By default, PostgreSQL logs are readable only by the `postgres` user and group.

   In the terminal, add the Datadog Agent user to the `postgres` group by running the following command:

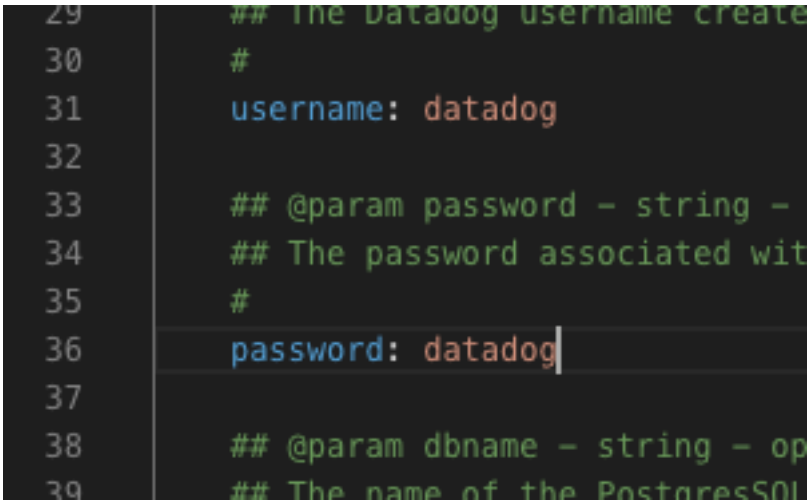


Figure 7: Set password for the Datadog Postgres user

```
##
## Discover Datadog log collection: https://docs.datadoghq.com/logs/log
#
logs:
  - type: file
    path: /var/log/postgresql/postgresql-12-main.log
    source: postgresql
    service: database
    log_processing_rules:
    - type: multi_line
      pattern: \d{4}\-(0?[1-9]|1[012])\-(0?[1-9]|[12][0-9]|3[01])
      name: new_log_start_with_date
```

Figure 8: PostgreSQL configuration logs block

```
usermod -a -G postgres dd-agent
```

10. Restart the Agent:

```
systemctl restart datadog-agent
```

11. After a few seconds, run the following command:

```
datadog-agent status
```

12. Scroll up to the **Running Checks** section and find the **postgres** entry, which should indicate that it's running OK:

```
postgres (13.7.0)
-----------------
  Instance ID: postgres:9e1d0b83682b440f [OK]
  Configuration Source: file:/etc/datadog-agent/conf.d/postgres.d/conf.yaml
  Total Runs: 1
  Metric Samples: Last Run: 38, Total: 38
  Events: Last Run: 0, Total: 0
  Service Checks: Last Run: 1, Total: 1
  Average Execution Time : 67ms
  Last Execution Date : 2023-07-11 19:34:53 UTC (1689104093000)
  Last Successful Execution Date : 2023-07-11 19:34:53 UTC (1689104093000)
  metadata:
    resolved_hostname: dd101-sre-host
    version.major: 12
    version.minor: 15
    version.patch: 0
    version.raw: 12.15 (Ubuntu 12.15-0ubuntu0.20.04.1)
    version.scheme: semver
```

Figure 9: Agent status PostgreSQL check

> **Note**: If you don't see an entry for **postgres**, run the `status` command after a minute. It can take a little while for the Agent to run checks after restarting.

13. Scroll down to the **Logs Agent** section and confirm that the Agent is collecting **postgres** logs:

You have fully configured the Agent for Storedog! Click **Next** to explore the results in Datadog.

```
============
Integrations
============

postgres
--------
  - Type: file
    Path: /var/log/postgresql/postgresql-12-main.log
    Service: database
    Source: postgresql
    Status: OK
    Inputs:
      /var/log/postgresql/postgresql-12-main.log
    Bytes Read: 365
    Lines Combined: 0
    MultiLine matches: 3
    Pipeline Latency:
      Average Latency (ms): 0
      24h Average Latency (ms): 0
      Peak Latency (ms): 0
      24h Peak Latency (ms): 0
```

Figure 10: Agent status PostgreSQL logs

Explore Datadog

You've done a lot of work installing the Agent and configuring it for Storedog running on the lab host. Now you can explore the wealth of metrics, events, traces, and logs that it's sending to your lab account!

Log in to Datadog

Make sure you are logged into Datadog using the Datadog training account credentials provisioned for you. Your trial username are displayed at the top of the terminal:

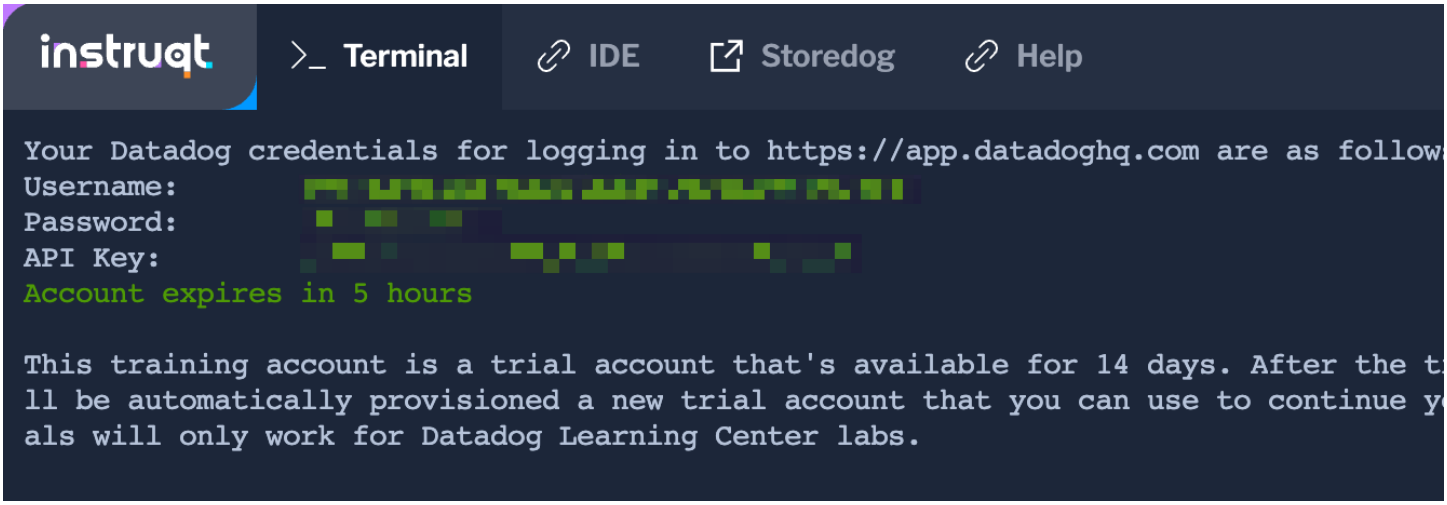


Figure 1: Datadog trial account credentials displayed in the terminal

You can display these credentials any time by running `creds` in the lab terminal.

Enable Log Management

Earlier, you confirmed that the Agent is sending logs to Datadog from the host by using the `status` command. You must also opt-in to using Log Management in your Datadog account:

1. Navigate to **Logs**. If you see Get Started content instead of actual logs, do the following:
    1. Click one of the **Get Started** buttons.
    2. On the modal that appears, click **Get Started** again.

    You should now see the Logs Explorer displaying many logs from Storedog services.

Explore Datadog

Start your tour by looking at Storedog's infrastructure, which consists of the virtual machine host and several Docker containers.

1. Navigate to **Infrastructure**.

    You will see `agent-host`, which is the hostname for your lab VM.

2. In the Infrastructure List, click `agent-host` to open the host details panel.

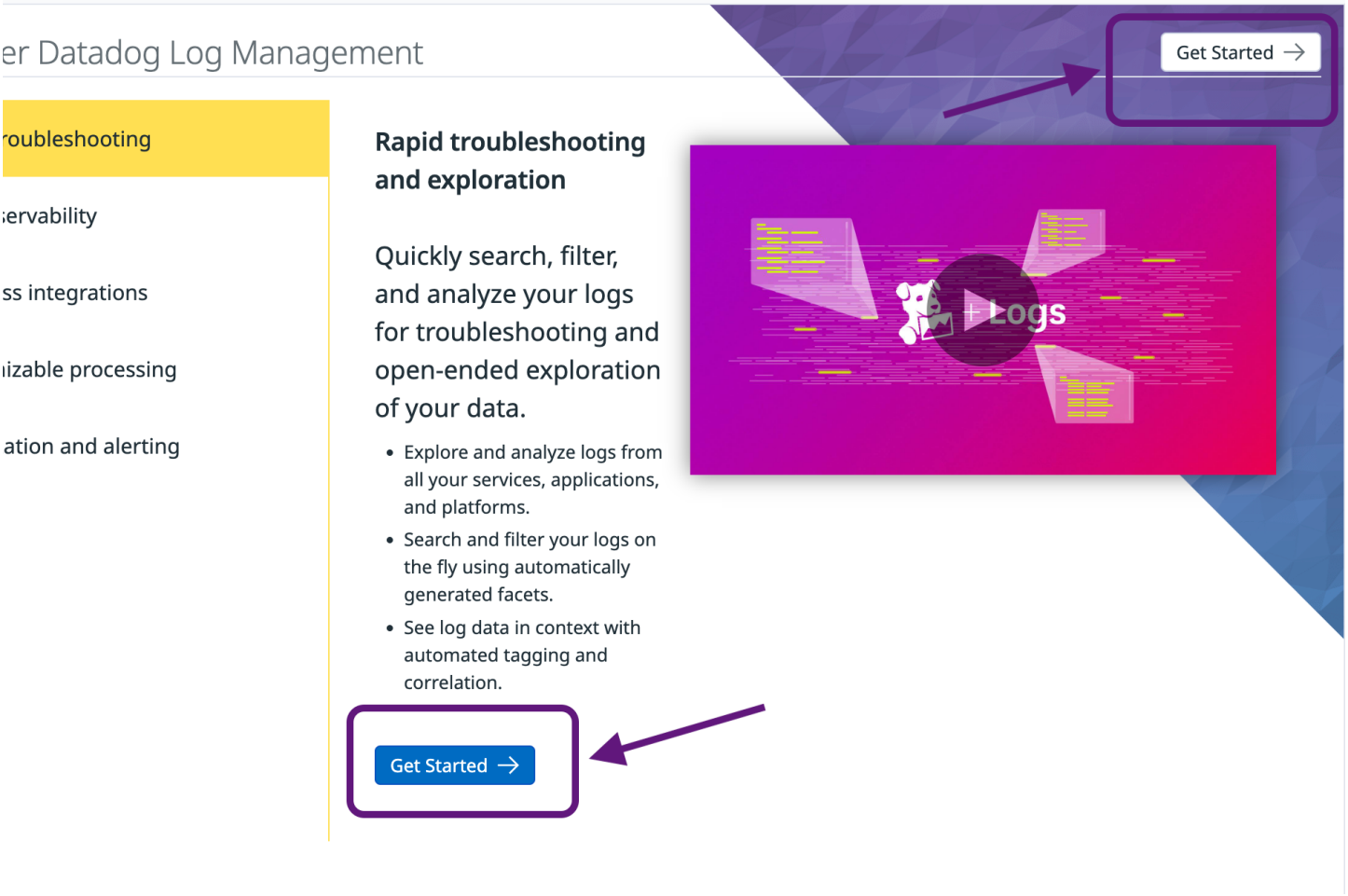


Figure 2: Opt into Log Management by clicking a Get Started button

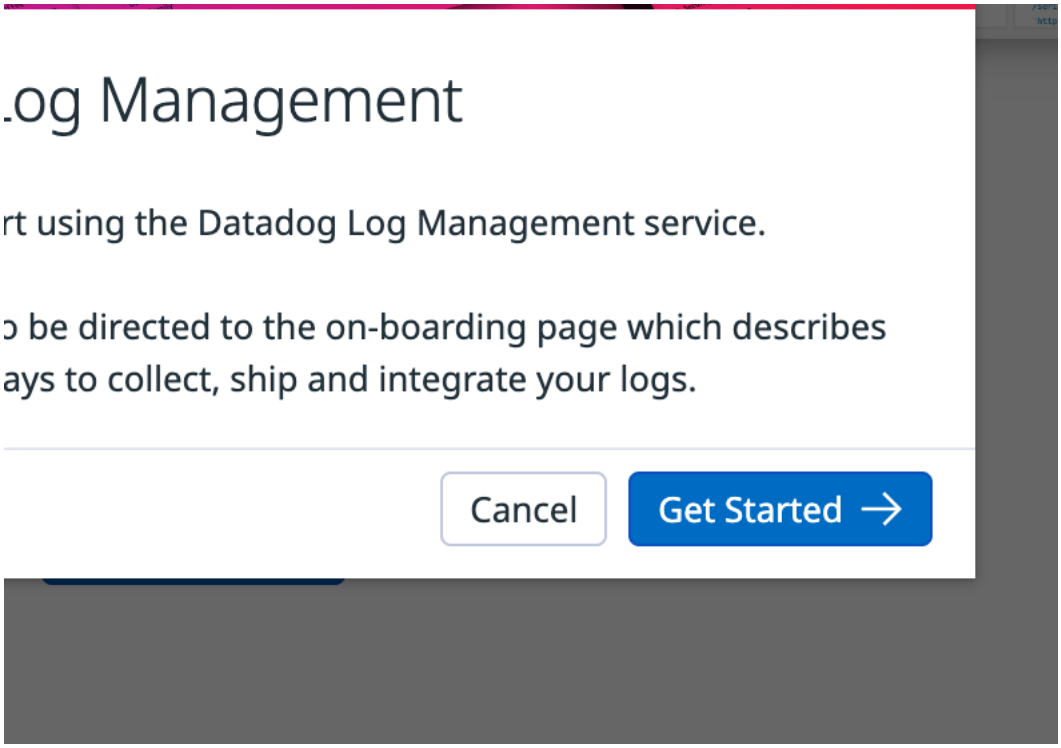


Figure 3: To enable Log Management, click Get Started again

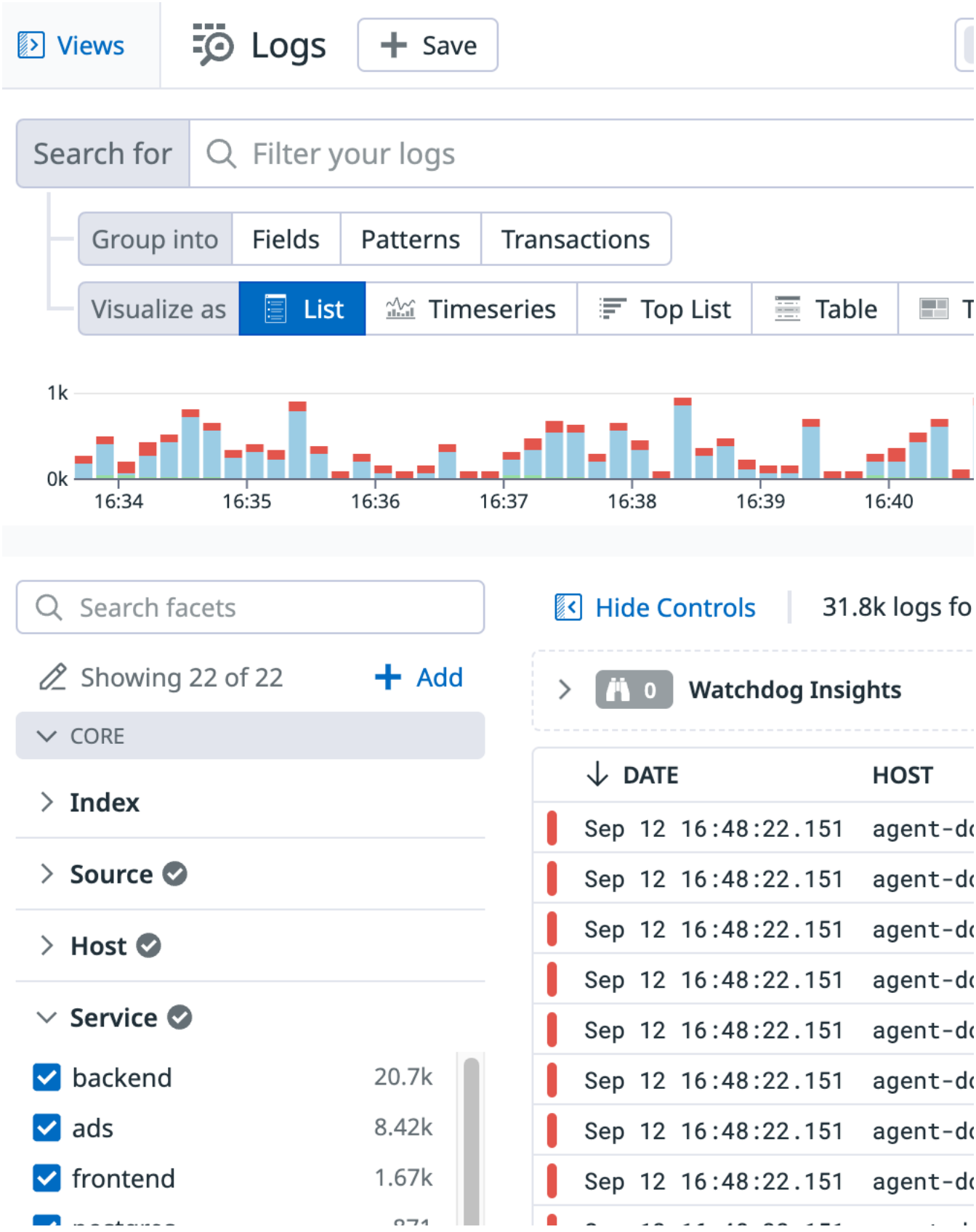


Figure 4: Logs Explorer showing logs from Storedog services

3. Click on the **Host Info** tab if it is not already selected.

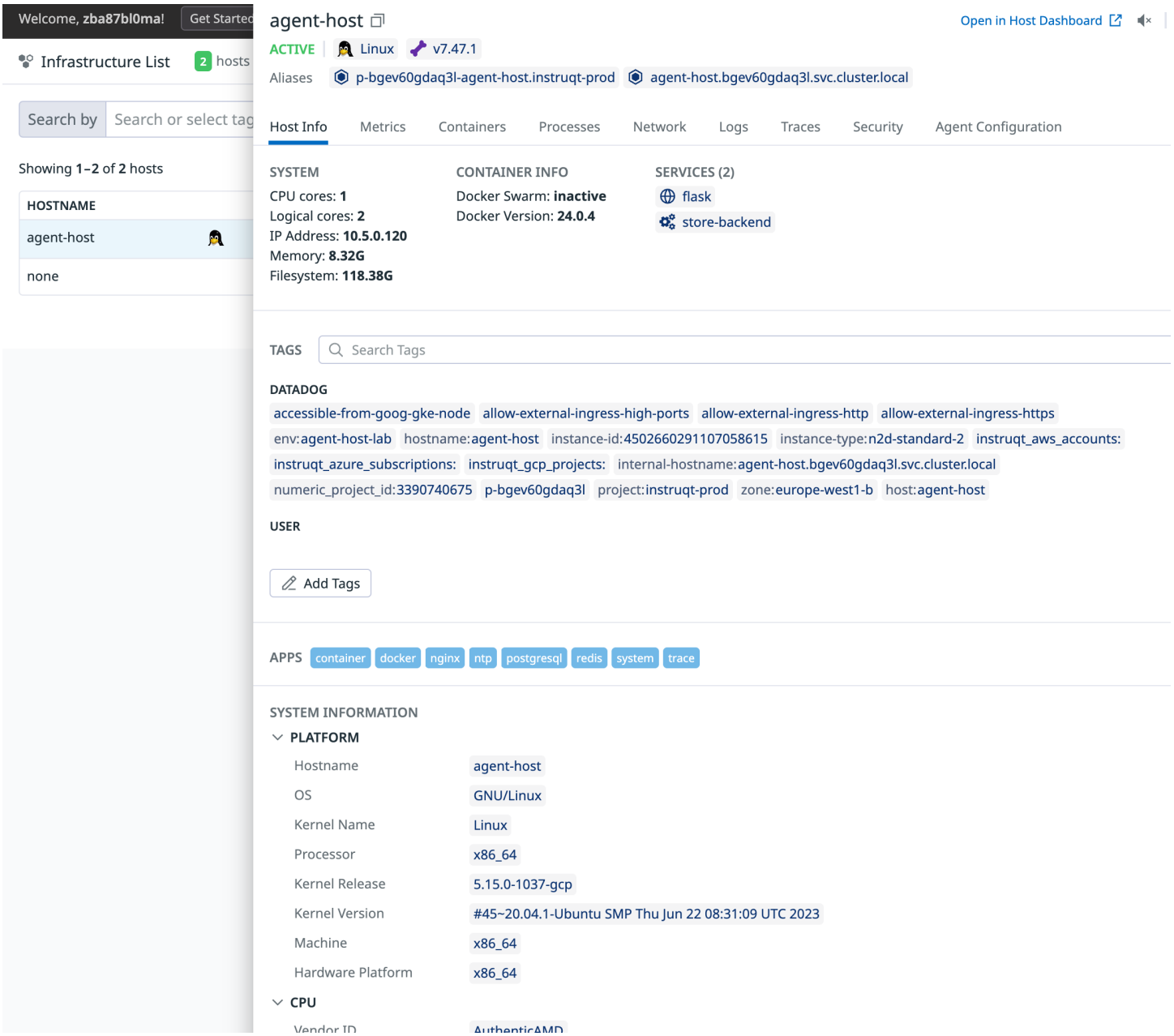


Figure 5: Infrastructure host details panel

The host details panel displays lots of information about the host that the Agent is running on.

4. Click on the **Processes** tab. These are the Live Processes that you configured the Agent to collect.
5. Navigate to **Infrastructure > Containers**. Here you can see all the service containers because you configured the Agent to collect metrics from Docker.

   Note: If you don't see any containers yet, you might have to wait a few minutes for the initial data from the Agent to be processed by Datadog.
6. Click on `lab_advertisements_1` and notice the metadata the Agent captures by default under **ALL TAGS**. You may need to expand the overflow list to see all the tags.
7. Find the tag `env:agent-lab`, which was applied to all events, metrics, and logs collected by the Agent.

   This is the value that you saw when using the `config` command. It is now a tag on all the data the Agent has collected. If you had multiple environments, this tag would help you filter and query resources accordingly.
8. Click on the **Logs** tab in this side panel to see the logs for this container.

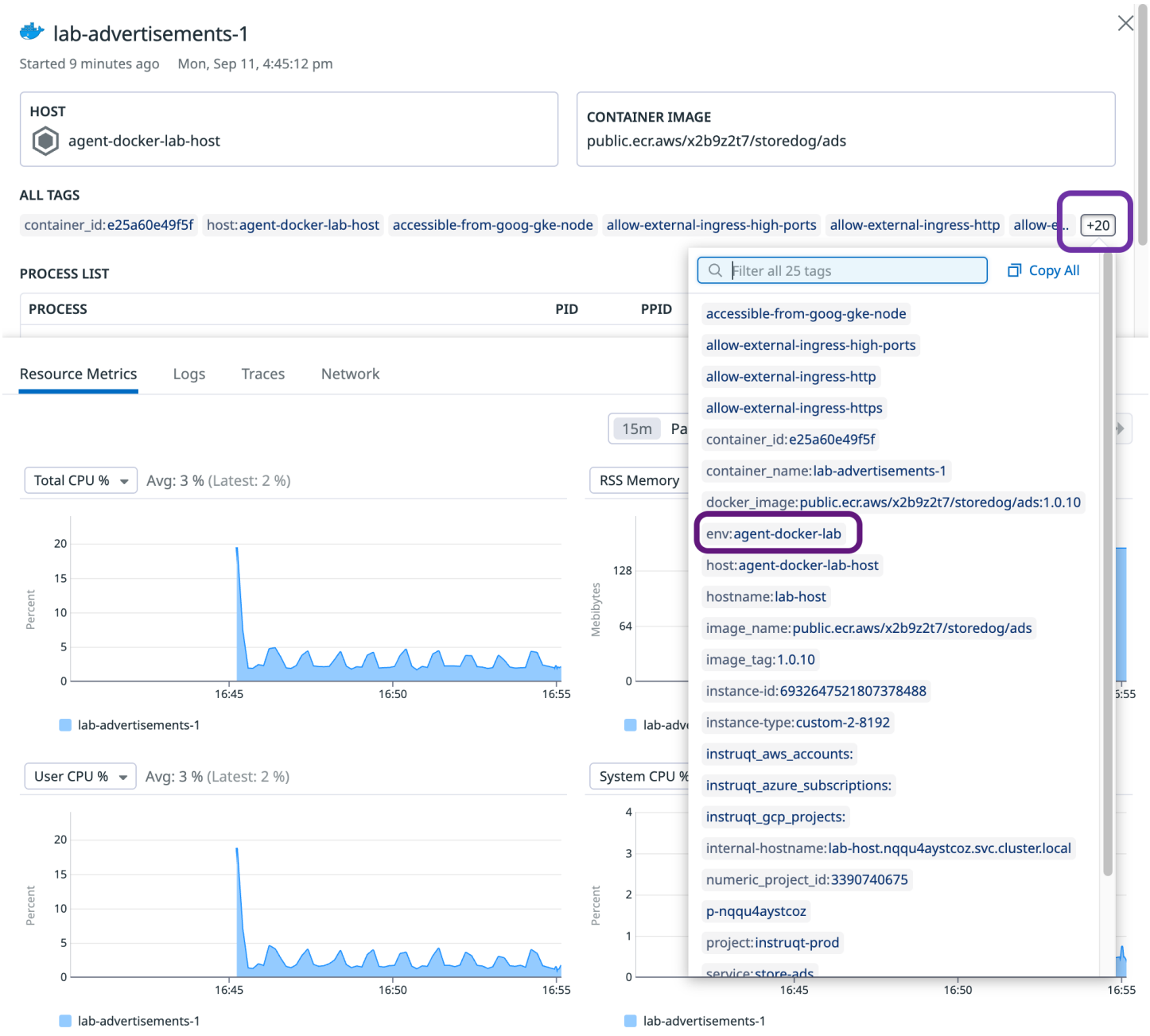


Figure 6: Advertisements container

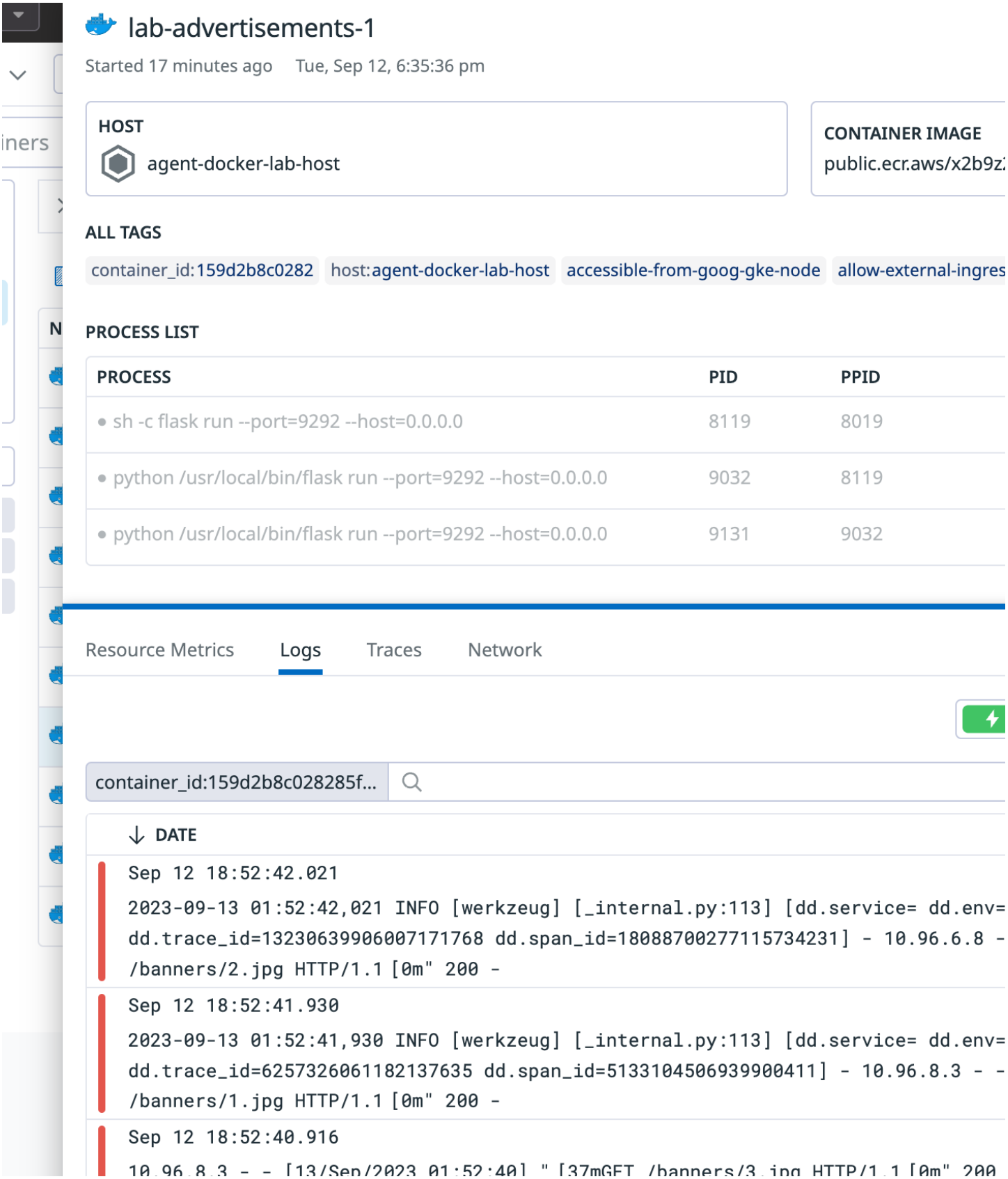


Figure 7: Logs tab of advertisements service container

These are the Live Tail logs emitted by the `lab_advertisements_1` container. If you don't see any logs, change the timespan selector to `Past 15 Minutes`:

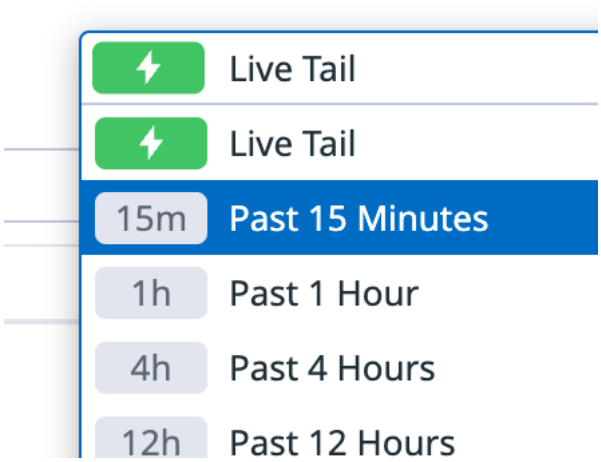


Figure 8: Selecting Past 15 from the container logs timeframe selector

9. Navigate to **Logs** to see all Storedog logs.
10. In the facets panel on the left, expand the **Service** section and select `discounts-service` to show only logs for the discounts service.

    You are able to see logs from Storedog services because you enabled the `container_collect_all` parameter while configuring the Agent.
11. Click on a `discounts-service` log entry. In addition to the log content, the Agent was able to capture a lot of metadata, such as the **CONTAINER NAME**, **DOCKER IMAGE**, **SERVICE**, and **SOURCE**.

    When Datadog parses a `source` tag, it tries to map the value to sources it knows about. Because all the services in Storedog are already tagged correctly, Datadog can indicate that they're coming from Python, Ruby, Nginx, Redis, an so on.

    If you go back to the IDE and look at the PostgreSQL and Redis `config.yaml` files, you'll see the these integrations have `source` parameters.
12. The Agent is also sending lots of traces from each Storedog service. You can examine them by navigating to **AMP > Traces**.

    You can use the facets panel on the left to filter traces, and click on traces to drill down into the details.

Explore on your own

There is so much data coming in from Storedog that you can spend lots of time exploring it all. Feel free to dig into areas that you're interested in.

Most importantly, look at the Dashboards that come with the Agent Integrations that you have worked with. For example:

- Docker - Overview
- NGINX - Overview
- NGINX - Metrics
- Postgres - Overview
- Postgres - Metrics
- Redis - Overview

You've completed the Agent on a Host lab! Click **Next** and then click **Mark Lesson Complete & Continue** to wrap up this course.

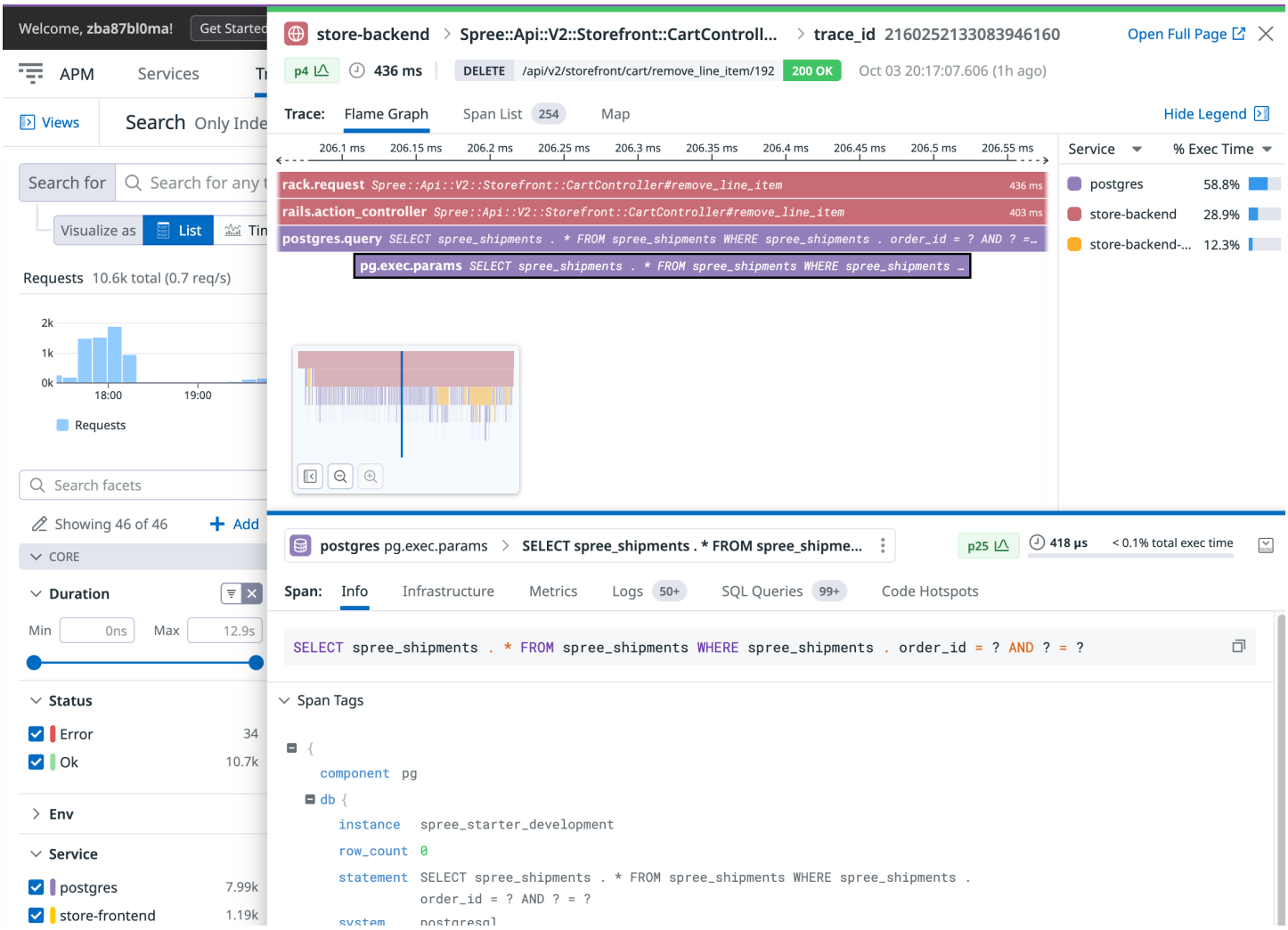


Figure 9: Storedog APM traces featuring a Postgres trace detail panel