Observability with Datadog

Bootcamp Labs

Notice

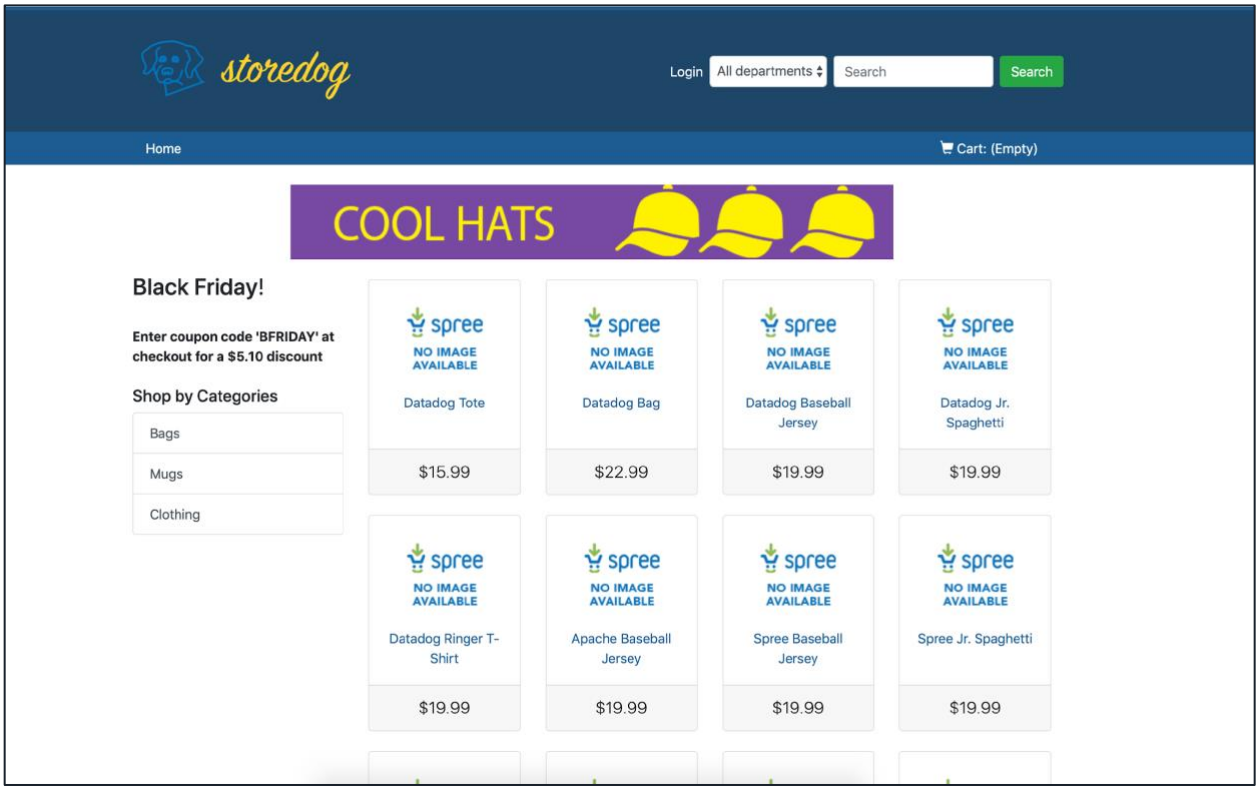
Copyright

Table of Contents

Labs Completion: IMP

This is a course demonstrating applying observability principals to an eCommerce app.

In this hypothetical scenario, we've got a Spree website, that a team has started adding microservices to. In its current state, the application is broken.



We'll take that broken application, instrument it with Datadog, and then deploy a fix. After deploying a fix, we'll look into Datadog to ensure our deploy worked, and that our systems are actually functioning properly.

This course will cover the below aspects of monitoring:

1. Infrastructure
2. Install Integrations
3. Application Performance Monitoring(APM)
4. Log Management
5. Browser Real User Monitoring
6. Synthetics

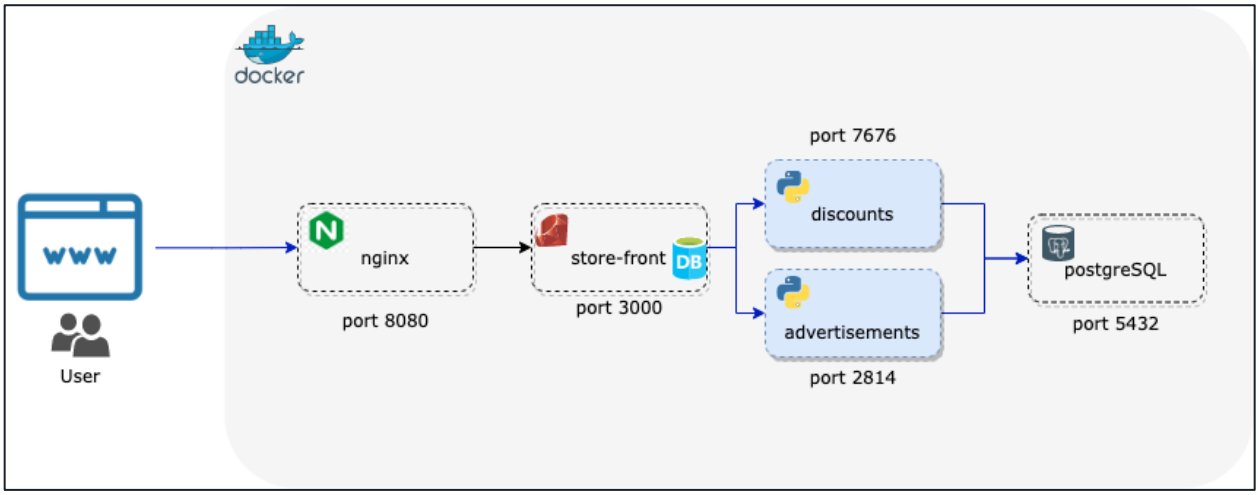
This course also enables you to create Dashboards, Monitors and SLOs.

The eCommerce application is hosted on virtual machines running as docker containers which we access, for the labs in this class.

Lab instructions will tell you which virtual machine to use. Read the lab instructions carefully to make ensure you are following the correct set of instructions
and connecting to the right machine. If you are unsure, please ask your instructor
before beginning the lab.

Make sure that you are typing things like “ (quotation) marks and - (dash) marks.
If you copy and paste from this lab guide you may end up with a notation that is not precisely the same and which could cause issues with this installation and labs further along.

The overall architecture of this lab environment is:

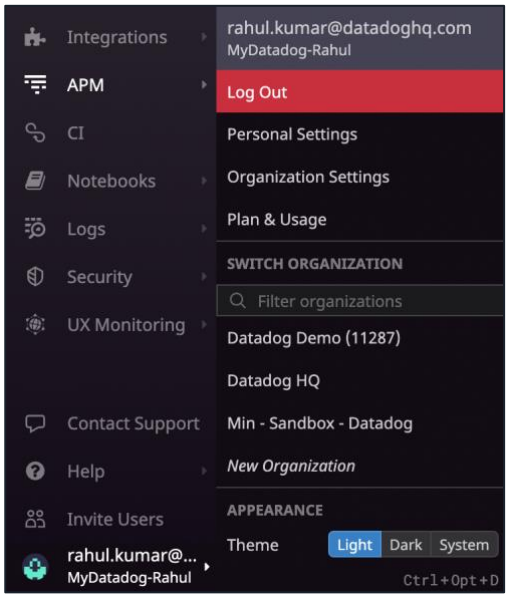


Services	Purpose
nginx	Act as pass through proxy and directs Internet client requests to the backend app store-frontend
store-frontend	Spree, an open-source e-commerce framework written in Ruby
discounts	Python Flask API that serves store item discounts
advertisements	Python Flask API that serves advertisements
db	PostgreSQL database used by discounts and advertisements

Datadog Account

Instructions

1. Launch https://app.datadoghq.com/account/login on your browser
2. Click **Sign in with Google** for authentications
3. Navigate to settings located on the bottom left corner, from the **SWITCH ORGANIZATION** select your dogfood account. Refer below screenshot:

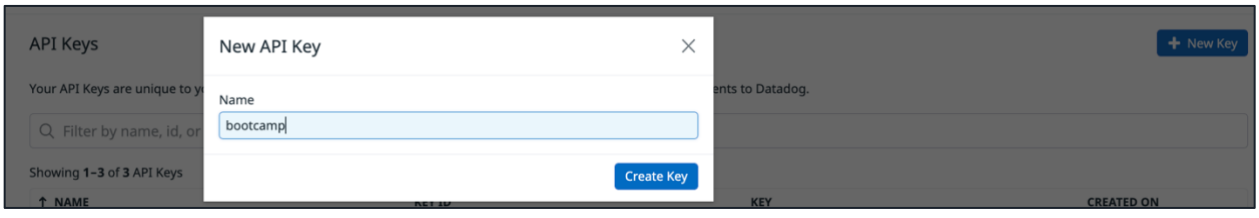


API Keys

Add an API key

To add a Datadog API key or client token:

1. Navigate to **Organization settings**, then click the **API keys** tab
2. Click the **New Key** button
3. Enter a name for your key or token as **bootcamp**
4. Click **Create Key**.



100 Accessing your Lab Environment

The lab environment for this class is hosted on a virtual machine, which you will need to access to complete the exercises.

Your instructor will provide you with details of your environment.

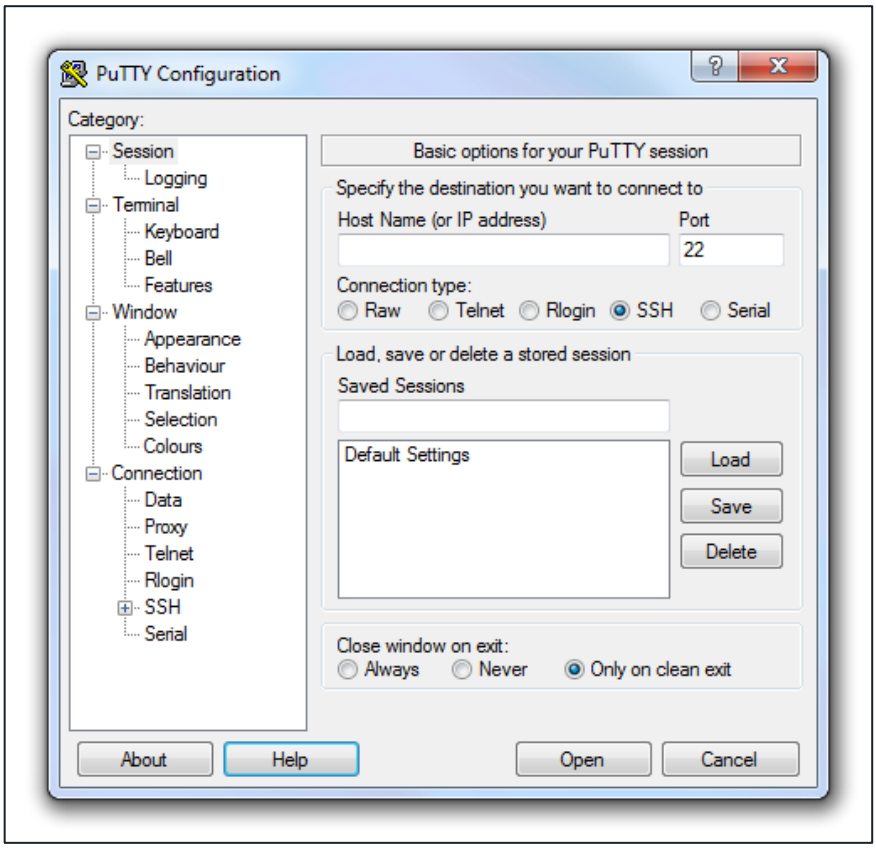
For SSH, if you are logging in from a Mac OS or Linux computer you should already have an SSH client installed. If you are logging in from a Windows computer, you may need to install PuTTY or another SSH client.

SSH Client on Mac OS

1. On Mac OS, OpenSSH is part of the system, for accessing your lab you can open a **Terminal**.
2. Change to the directory where you have saved your **bootcamp.pem** file.
3. Type in **chmod 400 bootcamp.pem**.
4. Type in **ssh-add bootcamp.pem**.
5. You can access your lab by typing in **ssh ubuntu@[Datadog host FQDN]**
6. You'll have to type **yes** to ensure you want to connect to this computer for the first-time connection.
7. You are now connected. Notice that the command prompt changed to your virtual machine instance.

SSH Client on Windows

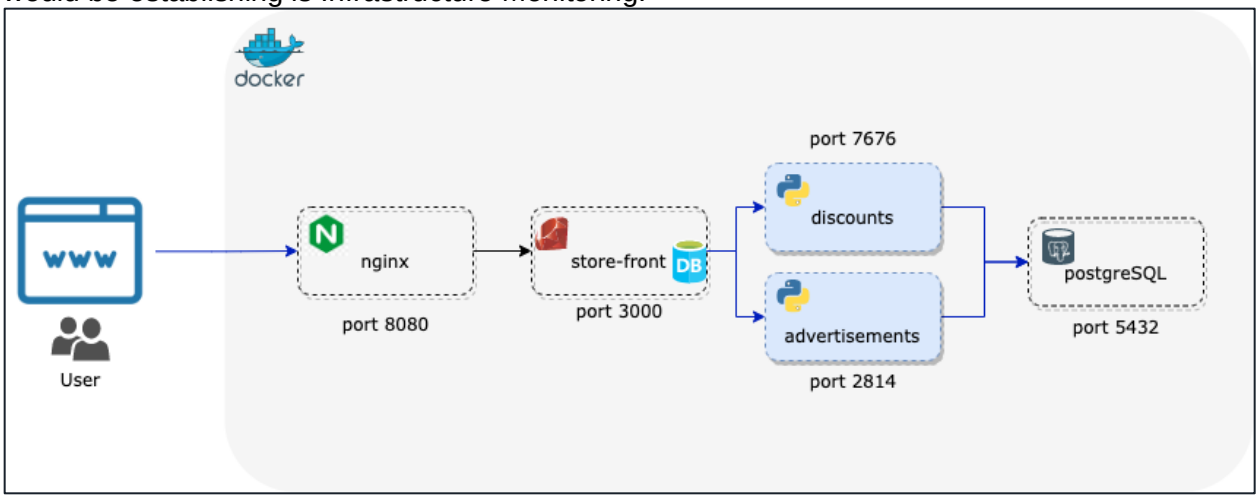
1. Download PuTTY or an equivalent application. PuTTY can be downloaded from: http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe
2. You will also need PuTTYGen, which can be downloaded from: http://the.earth.li/~sgtatham/putty/0.70/w32/puttygen.exe
3. From your Windows Downloads launch PuTTYGen
4. Press the **Load** button to load a private key file. Navigate to the location where you have saved your private key earlier (**bootcamp.pem)** and select it (change the option in the file type drop-down to **All Files**).
5. Notice the popup confirming successful import of the private key (old PEM format) into PuTTY's own format
6. Click the **Save private key** button, at which point you will be asked to provide a passphrase to protect the private key - this is not needed.
7. Save the new PuTTY key as "**Putty-Private-Key**" and make sure **PuTTY (*.ppk)** is selected as the type.
8. Now close PuTTYGen and run PuTTY.



9. Provide the IP address of the instance you want to connect to in the **Host Name (or IP address)** box.
10. Click **Connection > Data** in the left-hand navigation pane and set the **Auto-login username** to the one provided by your instructor.
11. Click **Connection > SSH > Auth** in the left-hand navigation pane and configure the private key to use by clicking **Browse** under Private key file for authentication.
12. Select your newly created **Putty-Private-Key.ppk** file.
13. Click **Open** to begin your session with the server. You may be prompted to check the server's key in case you don't trust it - click **Yes** to continue at this point.

101 Infrastructure Lab

Referring to the architecture diagram for our Spree application the first vital monitoring that we would be establishing is Infrastructure Monitoring.



As we discussed, Datadog's SaaS-based infrastructure monitoring provides metrics, visualizations, and alerting to ensure engineering teams can maintain and optimize cloud or hybrid environments. With extensive coverage of popular technologies, a simple deployment process that requires little maintenance, and an easy-to-use interface.

In this lab, you will install and configure the Agent in a Docker environment. The Agent will run in its own container along with other Docker containers comprising the Storedog ecommerce web application. We will also be enabling Live Process Collection.

101-1 Installing the Agent

1. Examine the docker-compose file (**docker-compose-no-agent.yml**) using below commands.

```
$ cd ~/docker/ecommerce-workshop/deploy
$ more docker-compose-no-agent.yml
```

Notice that you have below services which we discussed earlier in the Lab Completion section as part of docker-compose file. Let's add the Datadog agent to monitor the Infrastructure and live processes of our Spree website.

Service	Purpose
nginx	Act as pass through proxy and directs Internet client requests to the backend app store-frontend
store-frontend	Spree, an open-source e-commerce framework written in Ruby
discounts	Python Flask API that serves store item discounts
advertisements	Python Flask API that serves advertisements
db	PostgreSQL database used by discounts and advertisements

2. We will make a copy of **docker-compose-no-agent.yml** file and name it as **docker-compose-instr-infra.yml** using below command

```
$ cp docker-compose-no-agent.yml docker-compose-instr-infra.yml
```

3. Edit **docker-compose-instr-infra.yml** and paste the following text highlighted in **bold**, such that **agent:** and **discount:** are at the same indentation level

```
$ vi docker-compose-instr-infra.yml
```

```
version: '3'
services:
  agent:
    container_name: dd-agent
    image: "datadog/agent:latest"
    environment:
      - DD_SITE
      - DD_API_KEY
      - DD_HOSTNAME=bootcamp-lab
      - DD_TAGS="project:bootcamp"
      ## LIVE PROCESSES
      - DD_PROCESS_AGENT_ENABLED=true
      ## DOGSTATSD
      - DD_DOGSTATSD_NON_LOCAL_TRAFFIC=true
    ports:
      - 8126:8126/tcp # for APM
      - 8125:8125/udp # for Dogstatsd
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - /proc/:/host/proc/:ro
      - /sys/fs/cgroup/:/host/sys/fs/cgroup:ro
      - /etc/passwd:/etc/passwd:ro # LIVE PROCESSES
```

```
  discounts:
    container_name: discounts
```

Note: Read the public documentation https://docs.datadoghq.com/agent/docker/?tab=standard on usage of environment variables, commands, data collected etc.

4. Update your Datadog API key in the .env file (look under Datadog section)

```
$ cd ~/docker/ecommerce-workshop/deploy
$ vi .env
```

```
# Datadog
DD_SITE=datadoghq.com
DD_API_KEY= xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

5. Start the application and check the status

```
$ docker-compose -f docker-compose-instr-infra.yml up -d
$ docker ps -a
```

```
CONTAINER ID   IMAGE                  COMMAND                  CREATED
STATUS                            PORTS
NAMES
59c95b06bb4d   nginx:1.21.4           "/docker-entrypoint.…"   15 seconds ago   Up
14 seconds                        80/tcp, 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp
nginx
8e9c8027c2a2   store-frontend:1.0     "sh /opt/storedog/do…"   15 seconds ago   Up
14 seconds                        0.0.0.0:3000->3000/tcp, :::3000->3000/tcp
store-frontend
f9825e1eb1dd   ads:1.0                "/bin/bash -c 'sleep…"   17 seconds ago   Up
16 seconds                        0.0.0.0:7676->7676/tcp, :::7676->7676/tcp
ads
b47ddbc5f5bb   discounts:1.0          "/bin/bash -c 'sleep…"   17 seconds ago   Up
15 seconds                        0.0.0.0:2814->2814/tcp, :::2814->2814/tcp
discounts
8b359ca9eb10   postgres:12-alpine     "docker-entrypoint.s…"   24 seconds ago   Up
17 seconds                        5432/tcp
postgres
25c4d0bd0612   datadog/agent:latest   "/bin/entrypoint.sh"     24 seconds ago   Up
16 seconds (health: starting)   0.0.0.0:8125->8125/udp, :::8125->8125/udp,
0.0.0.0:8126->8126/tcp, :::8126->8126/tcp   dd-agent
```

6. Generate traffic load to the application

```
$ cd ~/docker/ecommerce-workshop/traffic
$ ./generate-traffic.sh
Generate traffic to web app
PID of this process: 3451
pid written to /home/ubuntu/docker/ecommerce-workshop/traffic/status.pid
```

Note down the pid or refer to the pid file, if you wish to stop the traffic

```
kill -9 /home/ubuntu/docker/ecommerce-workshop/traffic/status.pid
```

To validate that the traffic is generated successfully. Tail the status.log and check for response code 200

```
$ cd ~/docker/ecommerce-workshop/traffic
$ tail -f status.log
Generate traffic to web app
1: 200 http://localhost:8080/t/bags
2: 200 http://localhost:8080
3: 200 http://localhost:8080/t/clothing
4: 200 http://localhost:8080
5: 200 http://localhost:8080/cart
6: 200 http://localhost:8080
```

7. Check the Datadog agent status using below command

```
$ docker exec -it dd-agent agent status
```

101-2 Troubleshooting

Datadog public documentations can be handy to resolve any issues that you may come across, please find below some of the materials for your help.

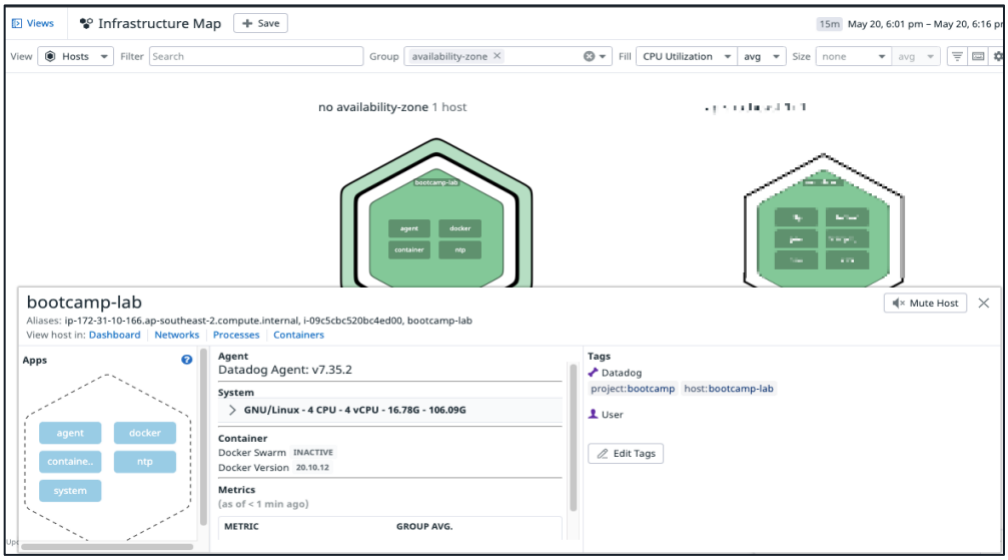
Agent Guides
Docker Agent

101-3 Visualising the Infrastructure Metrics

You can confirm that the Datadog agent is running and reporting to the Datadog Platform.

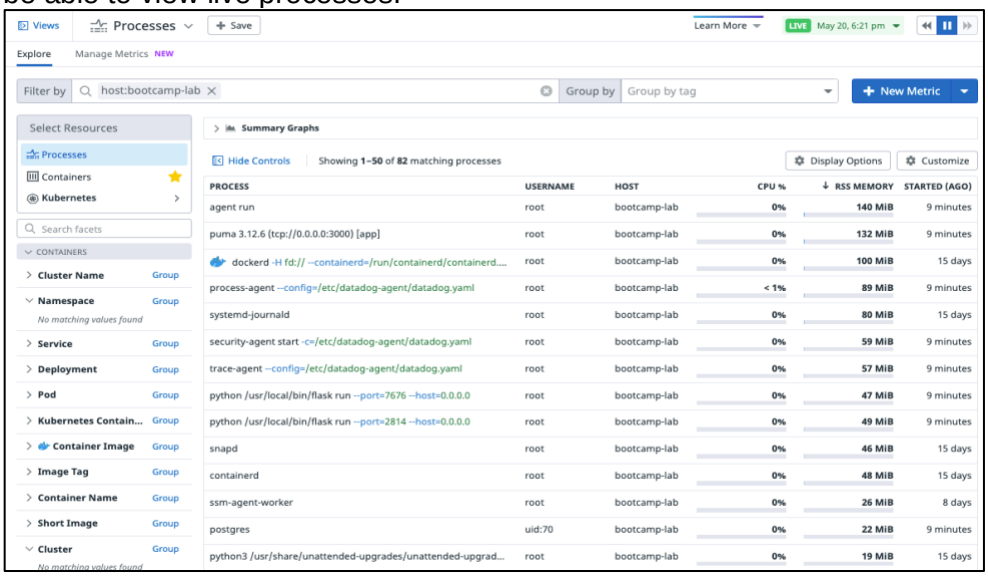
Host Maps:

1. Switch to Datadog UI
2. Navigate to **Infrastructure->Host Map**
3. You should see host with name "**bootcamp-lab**" reporting
4. Click on the hexagon to explore more on the infrastructure metrics.



Processes:

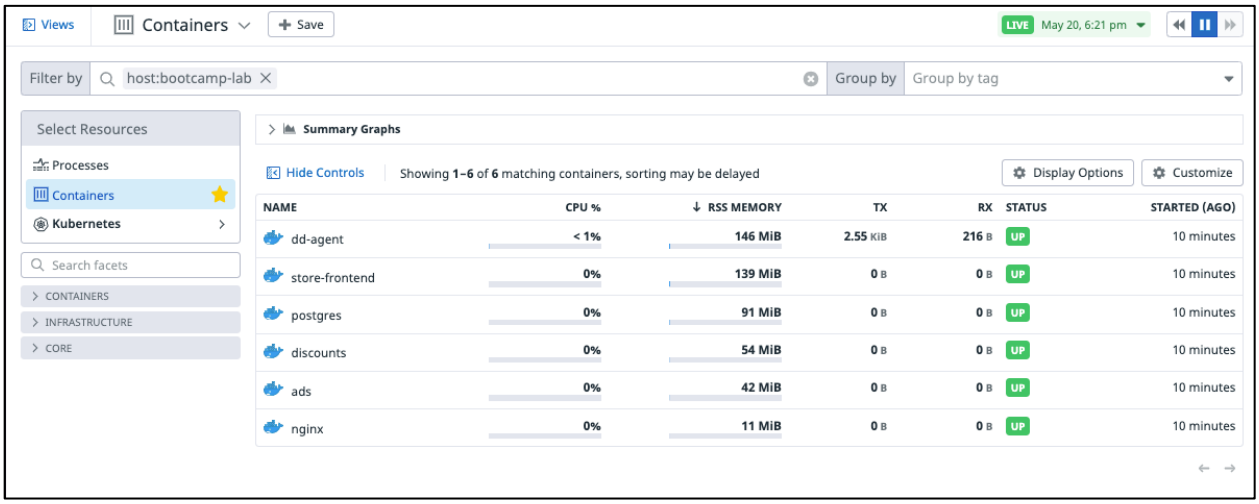
Navigate to **Infrastructure->Processes** and apply Filter by to **host:bootcamp-lab**. You should be able to view live processes.



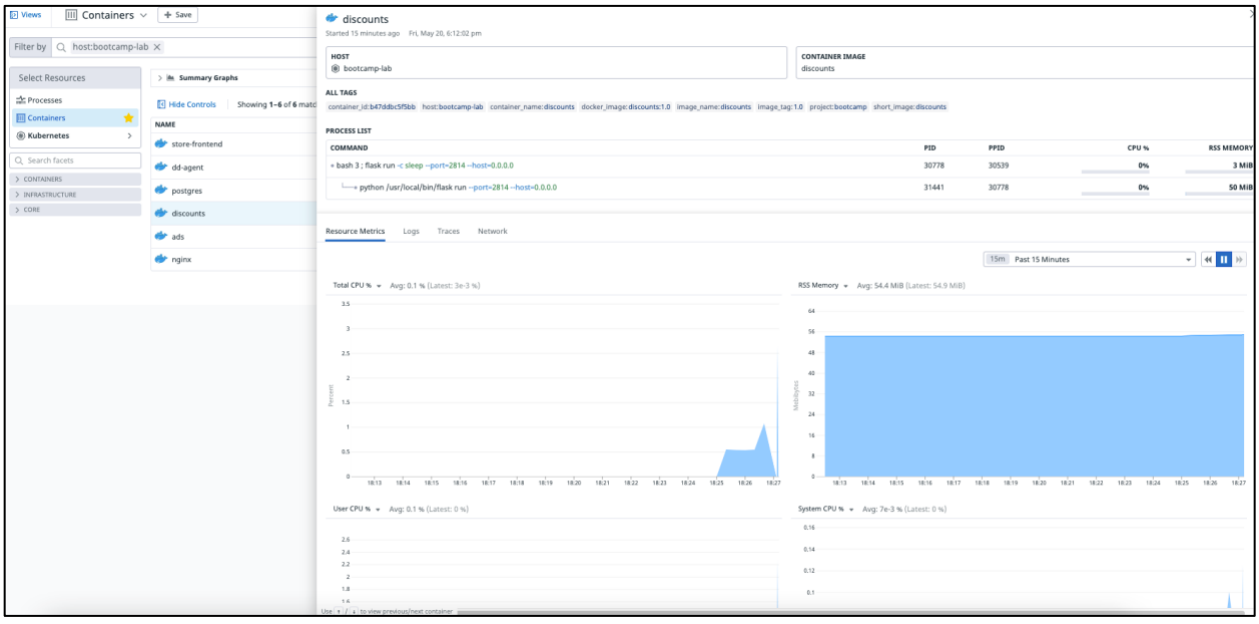
Passing DD_PROCESS_AGENT_ENABLED=true enables Live Processes to give real-time visibility into the process running on your infrastructure.

Containers:

Navigate to **Infrastructure->Containers** and apply Filter by to **host:bootcamp-lab**. You should be able to view all containers with the status for our lab applications



From the container list click on any container to view **Resource Metrics** and other metadata associated:



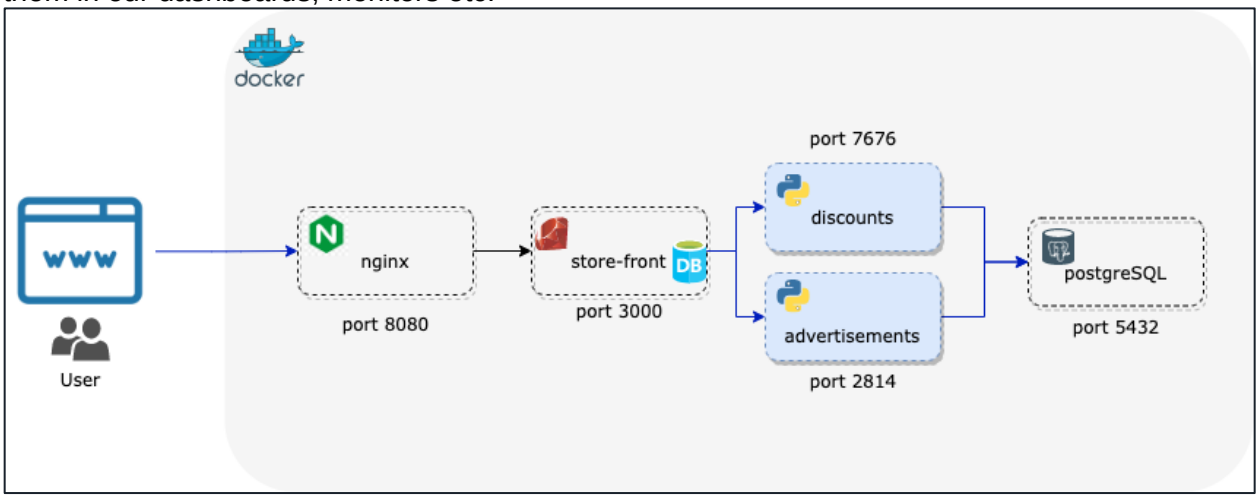
As you would notice that **Logs** and **Traces** sub-tab doesn't have any data reporting. This is because those configurations are yet to be made.

As we proceed further in our bootcamp we will learn how to enable **Logs** and **Traces** for our **Spree** website. We would also learn and explore the seamless co-relation Datadog provides between the 3 core Pillars of Observability- Metrics, Traces and Logs.

Further exploration on the infrastructure metrics, dashboards etc. is left to you. Play around or ask your instructor about any questions you may have.

102 Integrations Lab

Referring to the architecture diagram again, you would observe that we have **nginx** and **postgresSQL** for which we can leverage Datadog integration to monitor the key metrics, use them in our dashboards, monitors etc.



There are more than 500 integrations available to connect your systems, services, and applications to Datadog. You can use integrations to bring together all the metrics and logs from your infrastructure and gain insight into the unified system as a whole—you can see pieces individually and also how individual pieces are impacting the whole.

Datadog Integrations fall into three categories:

1. **Agent-based:** These integrations are installed with the Datadog Agent and use a Python class method called “check” to define the metrics to collect. These include core checks such as CPU, disk, and memory, as well as checks for popular services such as PostgreSQL, Redis, Apache, and many more.
2. **Authentication (crawler) based:** These integrations are set up in the Datadog App and enable communication between Datadog and external services. They require credentials for the external services they communicate with, and many of those services require your Datadog credentials. For example, Slack, AWS, Azure, and PagerDuty.
3. **Library:** Datadog API client code that you can import into your applications. There are libraries for Node.js, Python, .Net, and many more.

Note: Also refer to https://docs.datadoghq.com/getting_started/integrations/ for more details.

You’ll focus on Agent-based integrations in the following lab.

102-1 Pre-Requisites

In the previous lab we have setup infrastructure monitoring for our Spree application. But first, stop the application by using below command as we will be setting up Infrastructure and Integration monitoring together in this lab for our application.

```
$ cd ~/docker/ecommerce-workshop/deploy
$ docker-compose -f docker-compose-instr-infra.yml down
```

We will be making a copy of **docker-compose-instr-infra.yml** as **docker-compose-instr-infra-integration.yml** and make appropriate changes for the integration setup.

Remember you already have the **docker-compose-instr-infra.yml** with the Infrastructure Monitoring configurations.

102-2 Postgres Integration

In the lab exercises, we will take you through the steps on how to setup integration for database (postgres). The below link outlines in details the steps to setup integration for db service (postgres) https://docs.datadoghq.com/integrations/postgres/?tab=docker#setup

1. **Prepare Postgres**
   To get started with the PostgreSQL integration, create a read-only Datadog user with proper access to your PostgreSQL server.

   In this case, we will create a sql file where we define the the databaser user and password for Datadog and the necessary grant permission to be added.

```
$ cd ~/docker/ecommerce-workshop/db
$ cat dd_monitor.sql

  create user datadog with password 'ddpassword';
  grant pg_monitor to datadog;
  grant SELECT ON pg_stat_database to datadog;
```

   Here is the public documentation for detailed information:
   https://docs.datadoghq.com/integrations/postgres/?tab=docker#configuration

2. Make a copy of **docker-compose-instr-infra.yml** and name as **docker-compose-instr-infra-integration.yml** using below command

```
$ cd ~/docker/ecommerce-workshop/deploy
$ cp docker-compose-instr-infra.yml docker-compose-instr-infra-
integration.yml
```

3. Edit the **docker-compose-instr-infra-integration.yml** file and replace the **db:** section with below, make sure to follow the proper indentation.

```
$ vi docker-compose-instr-infra-integration.yml
```

```
db:
    container_name: postgres
    image: postgres:12-alpine
    restart: always
    environment:
      - POSTGRES_PASSWORD
      - POSTGRES_USER
      - DD_ENV=dev
    volumes:
      - ../db/initdb.sql:/docker-entrypoint-initdb.d/initdb.sql
      - ../db/dd_monitor.sql:/docker-entrypoint-initdb.d/dd_monitor.sql
    labels:
      com.datadoghq.ad.check_names: '["postgres"]'
      com.datadoghq.ad.init_configs: '[{}]'
      com.datadoghq.ad.instances: '[{"host":"%%host%%",
"port":5432,"username":"datadog","password":"ddpassword"}]'
      com.datadoghq.ad.logs: '[{"source": "postgresql", "service": "postgres"}]'
      my.custom.label.team: "db"
```

```
      my.custom.label.app: "spree"
```

Notice the additional environment variable with DD prefix. See documentation for more details https://docs.datadoghq.com/integrations/postgres/?tab=docker#configuration

Add in the dd_monitor.sql file location under db volume label and place it under `/docker-entrypoint-initdb.d`, so that the sql command is executed during database container startup.

Check that you have enter the correct password under `com.datadoghq.ad.instances` label if you choose to use a different password.

102-3 Nginx Integration

In the lab exercises, we will take you through the steps on how to setup integration for nginix. The below link outlines in detail the steps to setup integration for nginx
https://docs.datadoghq.com/integrations/nginx/?tab=host#installation

1. **Prepare Nginx**
   On each NGINX server, append the following in your nginx configuration file

```
$ cd ~/docker/ecommerce-workshop/nginx
$ vi default.conf

location /nginx_status {
        stub_status;
        server_tokens on;
```

   Hint: refer to `default.conf.instr` *for the changes required.*

2. The default NGINX log format does not have a request response time. To include it into your logs, update the NGINX log format by adding `$request_time` into the log_format_main of the http section of your NGINX configuration file (/etc/nginx/nginx.conf):

```
$ cd ~/docker/ecommerce-workshop/nginx
$ vi nginx.conf


log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent $request_time "$http_referer"'
                      '"$http_user_agent" "$http_x_forwarded_for"';
```

   Hint: refer to `nginx.conf.instr` *for the changes required.*

   Here is the public documentation for detailed information:
   https://docs.datadoghq.com/integrations/nginx/?tab=host#configuration

3. Edit the **docker-compose-instr-infra-integration.yml** file and replace the **nginx:** section with below, make sure to follow the proper indentation.

```
$ vi docker-compose-instr-infra-integration.yml
```

```
nginx:
   container_name: nginx
   image: nginx:1.21.4
   restart: always
   environment:
     - DD_ENV=dev
   volumes:
     - ../nginx/default.conf:/etc/nginx/conf.d/default.conf
     - ../nginx/nginx.conf:/etc/nginx/nginx.conf
   ports:
     - "8080:8080"
   depends_on:
     - store-frontend
```

```
    labels:
      com.datadoghq.ad.check_names: '["nginx"]'
      com.datadoghq.ad.init_configs: '[{}]'
      com.datadoghq.ad.instances: '[{"nginx_status_url":
"http://%%host%%:8080/nginx_status/"}]'
      my.custom.label.team: "nginx"
      my.custom.label.app: "spree"
```

Notice the additional environment variable with DD prefix. See documentation for more details.
https://docs.datadoghq.com/integrations/nginx/?tab=docker#installation

102-4 Validating the Integration

Now that we have successfully edited our **docker-compose-instr-infra-integration.yml** and made appropriate changes on the application side, lets run the application using below command

```
$ cd ~/docker/ecommerce-workshop/deploy
$ docker-compose -f docker-compose-instr-infra-integration.yml up -d
$ docker ps -a
```

```
CONTAINER ID   IMAGE                  COMMAND                  CREATED
STATUS                  PORTS
NAMES
a18ae55b5d08   nginx:1.21.4           "/docker-entrypoint.…"   6 hours ago   Up 6
hours               80/tcp, 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp
nginx
f9946aaeacfa   store-frontend:1.0     "sh /opt/storedog/do…"   6 hours ago   Up 6
hours               0.0.0.0:3000->3000/tcp, :::3000->3000/tcp
store-frontend
9183652eaff6   ads:1.0                "/bin/bash -c 'sleep…"   6 hours ago
Exited (1) 6 hours ago
ads
28ec9345e4db   discounts:1.0          "/bin/bash -c 'sleep…"   6 hours ago
Exited (1) 6 hours ago
discounts
d38b81fc0a10   postgres:12-alpine     "docker-entrypoint.s…"   6 hours ago   Up 6
hours               5432/tcp
postgres
c968c1e9b227   datadog/agent:latest   "/bin/entrypoint.sh"     6 hours ago   Up 6
hours (healthy)     0.0.0.0:8125->8125/udp, :::8125->8125/udp, 0.0.0.0:8126-
>8126/tcp, :::8126->8126/tcp   dd-agent
```

Check the Datadog agent status using below command

```
$ docker exec -it dd-agent agent status
```

The output of the command should show the nginx and postgres status as [**OK]** as below:

```
    nginx (5.2.1)
    -------------
      Instance ID: nginx:c6bb35114ac88105 [OK]
      Configuration Source:
container:docker://a18ae55b5d082d1c448a175213964039c880a14bb9c077c3264e8e38b5739325
      Total Runs: 1,380
      Metric Samples: Last Run: 7, Total: 9,660
      Events: Last Run: 0, Total: 0
      Service Checks: Last Run: 1, Total: 1,380
      Average Execution Time : 3ms
      Last Execution Date : 2022-05-18 09:02:20 UTC (1652864540000)
      Last Successful Execution Date : 2022-05-18 09:02:20 UTC (1652864540000)
      metadata:
        version.major: 1
        version.minor: 21
        version.patch: 4
        version.raw: 1.21.4
        version.scheme: semver
```

```
    ntp
    ---
      Instance ID: ntp:d884b5186b651429 [OK]
      Configuration Source: file:/etc/datadog-agent/conf.d/ntp.d/conf.yaml.default
      Total Runs: 23
      Metric Samples: Last Run: 1, Total: 23
      Events: Last Run: 0, Total: 0
      Service Checks: Last Run: 1, Total: 23
      Average Execution Time : 0s
      Last Execution Date : 2022-05-18 08:47:33 UTC (1652863653000)
      Last Successful Execution Date : 2022-05-18 08:47:33 UTC (1652863653000)


    postgres (12.1.1)
    -----------------
      Instance ID: postgres:b718d4e6e7897786 [OK]
      Configuration Source:
container:docker://d38b81fc0a107402439a95ba9f6bacba73a7ddd8db741bbcb07a5c97b56e7b43
      Total Runs: 1,379
      Metric Samples: Last Run: 45, Total: 62,055
      Events: Last Run: 0, Total: 0
      Service Checks: Last Run: 1, Total: 1,379
      Average Execution Time : 21ms
      Last Execution Date : 2022-05-18 09:02:13 UTC (1652864533000)
      Last Successful Execution Date : 2022-05-18 09:02:13 UTC (1652864533000)
      metadata:
        version.major: 12
        version.minor: 10
        version.patch: 0
        version.raw: 12.10
        version.scheme: semver
```

[May be needed] Follow below steps in case you don't see Integrations installed:

1. Login to your **Datadog account.**
2. Navigate to **Integrations > Integrations** List.
3. Under Installed, you may see tiles for Nginx and PostgreSQL. Integrations are automatically installed when the Datadog agent sends data from these services, but this can take some time. If you don't see tiles for Nginx and PostgreSQL yet, you can install them manually:

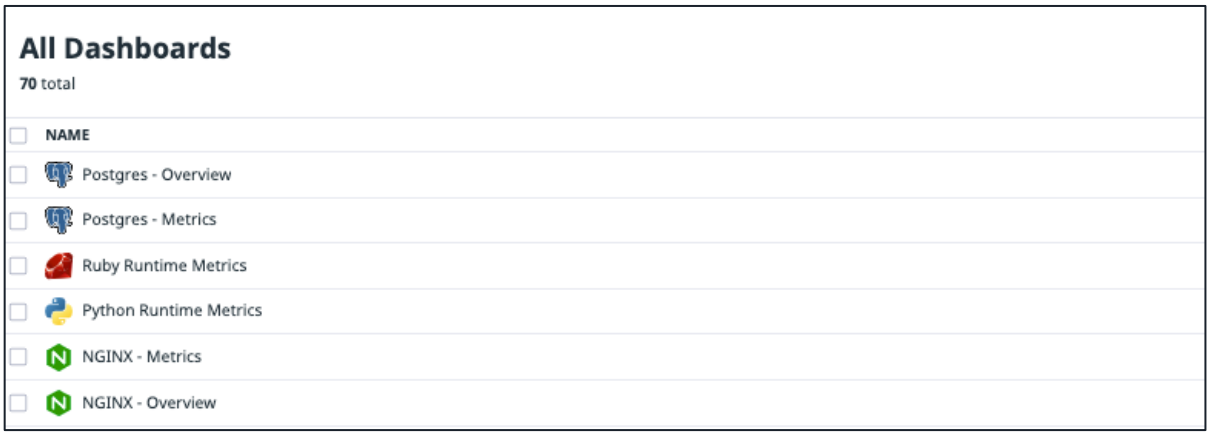
    I. Hover over the Nginx and PostgreSQL tile(one by one) and click the **Install** button.
    II. Click **Install Integration** on the Configuration tab.

102-5 Visualising the Integration Metrics

Dashboards:

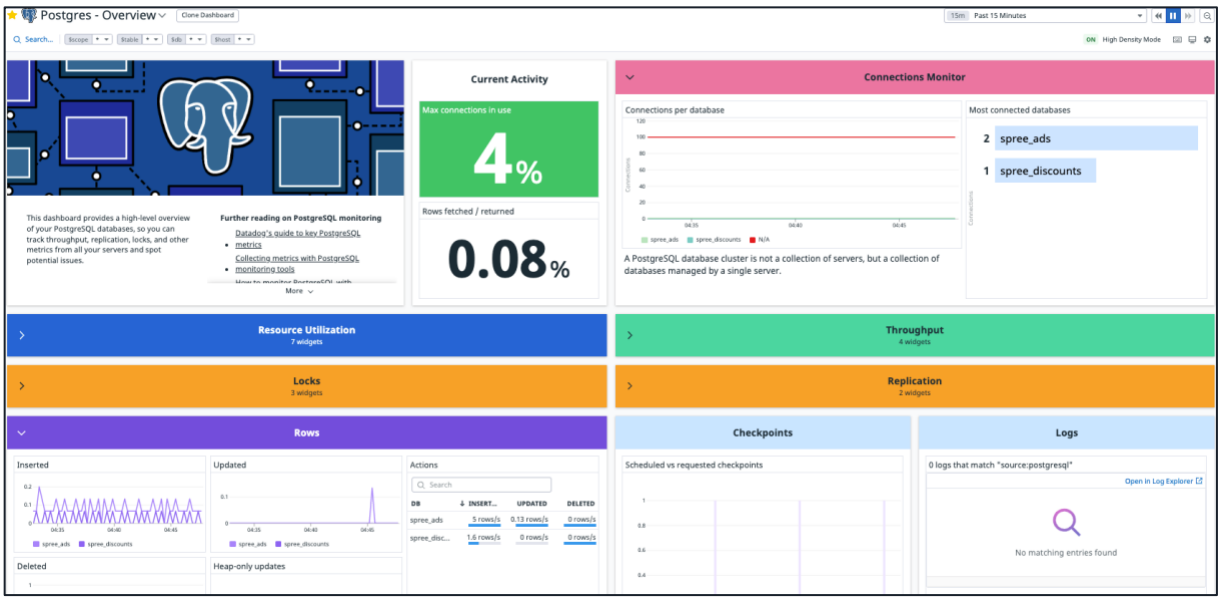
Navigate to the **Dashboards ->Dashboards List**. You should see several new dashboards, including **Nginx – Overview**, **Postgres – Overview**, **Postgres – Metrics** etc.

https://app.datadoghq.com/dashboard/lists



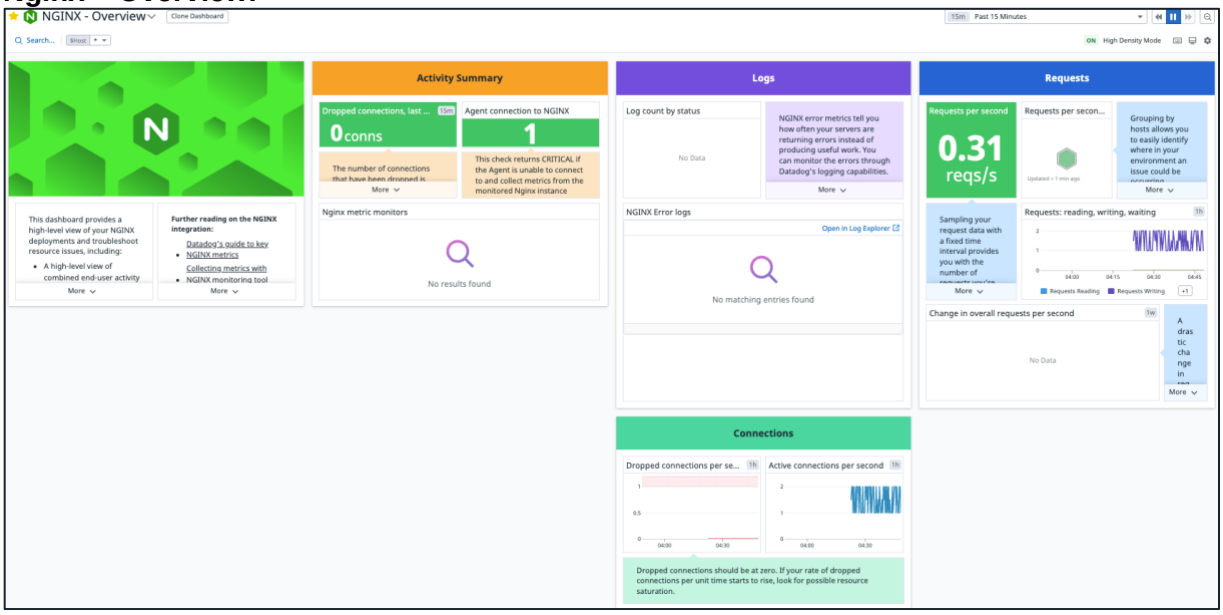
Click on the link for the **Nginx - Overview** dashboard. Also, explore the **Postgres – Overview** and **Postgres – Metrics.** *Note that there's not much here because they're not handling any requests.*

Postgres – Overview:



Postgres – Metrics:

Nginx – Overview:



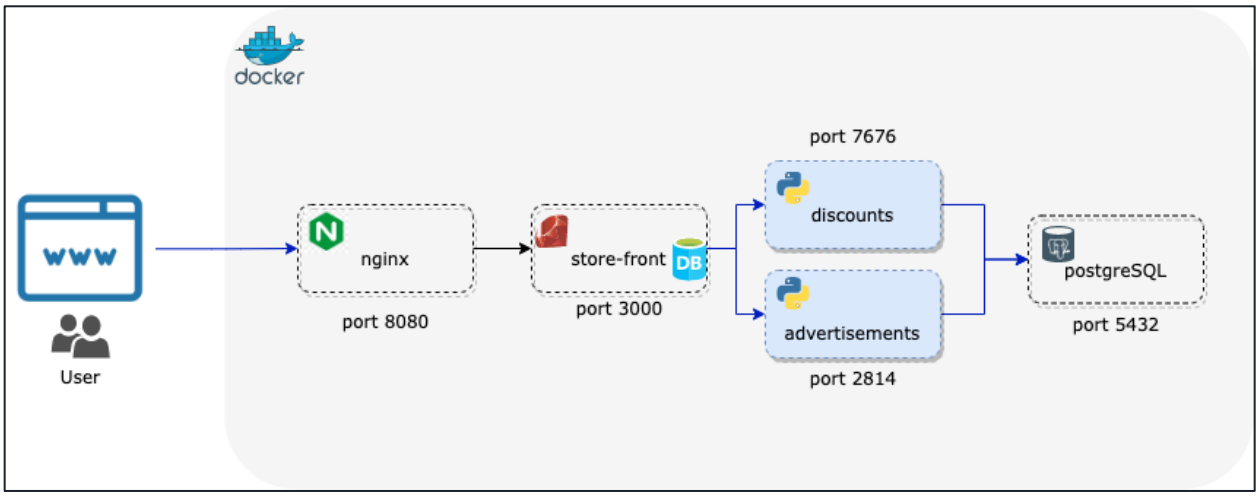
Nginx – Metrics:

Further exploration on the integration metrics, dashboards etc. is left to you. Play around or ask your instructor about any questions you may have.

103 APM and Logs Collection

With Infrastructure and Integration Monitoring in place, it is now good time for us to setup Application Performance and Log Monitoring.

Looking at the Spree e-commerce shopping website architecture diagram we understand that the web frontend, which is the store-frontend runs the main e-commerce shopping platform which is hosted behind the nginx instance.



The below table details about what each service is responsible for:

Service	Purpose
nginx	Act as pass through proxy and directs Internet client requests to the backend app store-frontend
store-frontend	Spree, an open-source e-commerce framework written in Ruby
discounts	Python Flask API that serves store item discounts
advertisements	Python Flask API that serves advertisements
db	PostgreSQL database used by discounts and advertisements

In order to boost sales and retain customer loyalty, the business owner has asked the app team to include advertisements and discount services to the shopping platform.

As the team started adding the microservices during development. In its current state, the application is broken.

The goal is to help the team to triage and pinpoint to the root cause of the problem using Datadog APM

In the previous lab we have setup infrastructure and integration monitoring for our Spree application. But first, stop the application using below command:

```
$ cd ~/docker/ecommerce-workshop/deploy
$ docker-compose -f docker-compose-instr-infra-integration.yml down
```

103-1 Prepare the lab

Instructions

In your lab environment, these services are running as containers. Check that the images are available

1. Check required docker images

```
$ docker images

REPOSITORY       TAG                 IMAGE ID       CREATED          SIZE
discounts        1.0                 070a6e4055d5   3 minutes ago    426MB
ads              1.0                 f29b854e3c1a   13 minutes ago   382MB
store-frontend   1.0                 4dd41722552b   15 minutes ago   886MB
postgres         12-alpine           d8e3d21aa571   3 weeks ago      206MB
nginx            1.21.4              f6987c8d6ed5   4 months ago     141MB
python           3.9.6-slim-buster   e18d3088c48c   8 months ago     115MB
ruby             2.7.2-slim-buster   9427c7849cdc   13 months ago    149MB
```

2. Start the application

```
$ cd ~/docker/ecommerce-workshop/deploy
$ docker-compose -f docker-compose-no-agent.yml up -d
Creating network "deploy_default" with the default driver
Creating postgres ...
Creating postgres ... done
Creating discounts ...
Creating ads ...
Creating discounts
Creating ads ... done
Creating store-frontend ...
Creating store-frontend ... done
Creating nginx ...
Creating nginx ... done
```

3. Validate that all the following containers are running as expected

```
$ docker ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED
STATUS          PORTS                                                  NAMES
c505cf65add2   nginx:1.21.4         "/docker-entrypoint.…"   36 seconds ago   Up 35
seconds   80/tcp, 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp   nginx
3f1ab563d61d   store-frontend:1.0   "sh /opt/storedog/do…"   36 seconds ago   Up 35
seconds   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp          store-frontend
234073a2fd30   ads:1.0              "/bin/bash -c 'sleep…"   37 seconds ago   Up 36
seconds   0.0.0.0:7676->7676/tcp, :::7676->7676/tcp          ads
f7120e77d784   discounts:1.0        "/bin/bash -c 'sleep…"   37 seconds ago   Up 36
seconds   0.0.0.0:2814->2814/tcp, :::2814->2814/tcp          discounts
772c3f63591c   postgres:12-alpine   "docker-entrypoint.s…"   37 seconds ago   Up 36
seconds   5432/tcp postgres
```

4. Tail each individual container log

```
$ docker logs nginx
```

```
022/04/28 12:19:57 [notice] 1#1: using the "epoll" event method
2022/04/28 12:19:57 [notice] 1#1: nginx/1.21.4
2022/04/28 12:19:57 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2022/04/28 12:19:57 [notice] 1#1: OS: Linux 5.4.0-1072-aws
2022/04/28 12:19:57 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2022/04/28 12:19:57 [notice] 1#1: start worker processes
2022/04/28 12:19:57 [notice] 1#1: start worker process 34
2022/04/28 12:19:57 [notice] 1#1: start worker process 35
2022/04/28 12:19:57 [notice] 1#1: start worker process 36
2022/04/28 12:19:57 [notice] 1#1: start worker process 37
```

```
$ docker logs store-frontend
Puma starting in single mode...
* Version 3.12.6 (ruby 2.7.2-p137), codename: Llamas in Pajamas
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://0.0.0.0:3000
Use Ctrl-C to stop
```

```
$ docker logs discounts
* Serving Flask app "discounts.py"
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:2814/ (Press CTRL+C to quit)
```

```
$ docker logs ads
* Serving Flask app "ads.py"
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:7676/ (Press CTRL+C to quit)
```

```
$ docker logs postgres
2022-04-28 12:19:55.888 UTC [1] LOG:  starting PostgreSQL 12.10 on x86_64-pc-linux-
musl, compiled by gcc (Alpine 10.3.1_git20211027) 10.3.1 20211027, 64-bit
2022-04-28 12:19:55.888 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port
5432
2022-04-28 12:19:55.888 UTC [1] LOG:  listening on IPv6 address "::", port 5432
2022-04-28 12:19:55.893 UTC [1] LOG:  listening on Unix socket
"/var/run/postgresql/.s.PGSQL.5432"
2022-04-28 12:19:55.915 UTC [22] LOG:  database system was shut down at 2022-04-28
12:06:13 UTC
2022-04-28 12:19:55.921 UTC [1] LOG:  database system is ready to accept
connections
```

5. Generate traffic load to the application

```
$ cd ~/docker/ecommerce-workshop/traffic
$ ./generate-traffic.sh
Generate traffic to web app
PID of this process: 3451
pid written to /home/ubuntu/docker/ecommerce-workshop/traffic/status.pid
```

Note down the pid or refer to the pid file, if you wish to stop the traffic

```
kill -9 /home/ubuntu/docker/ecommerce-workshop/traffic/status.pid
```

To validate that the traffic is generated successfully. Tail the status.log and check for response code 200

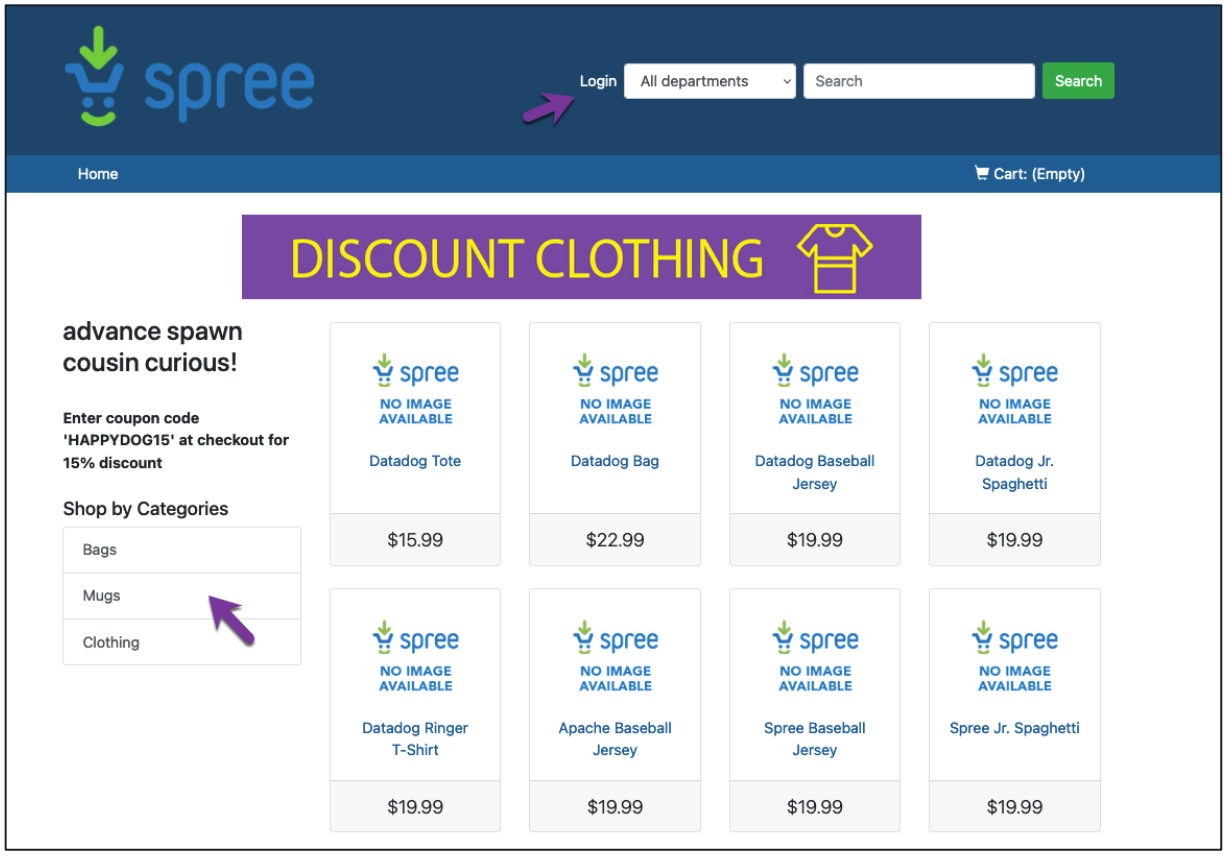
```
$ cd ~/docker/ecommerce-workshop/traffic
$ tail -f status.log
Generate traffic to web app
1: 200 http://localhost:8080/t/bags
2: 200 http://localhost:8080
3: 200 http://localhost:8080/t/clothing
4: 200 http://localhost:8080
5: 200 http://localhost:8080/cart
6: 200 http://localhost:8080
```

6. Browse the spree ecommerce website

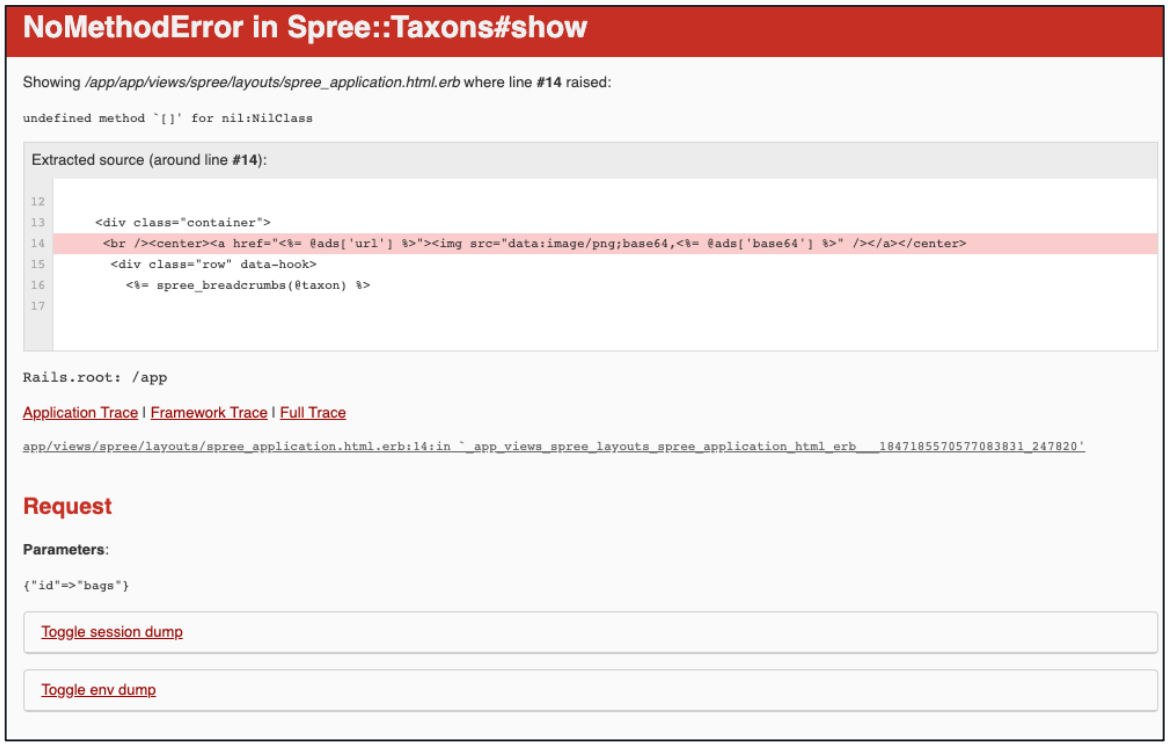
   Go to http:<your-public-hostname>:8080

   To find out your public hostname, do the following

```
$ curl http://169.254.169.254/latest/meta-data/public-hostname; echo
ec2-13-236-6-50.ap-southeast-2.compute.amazonaws.com
```



Homepage load successfully but there's error as you click the "Shop by Categories", or "Login"



NoMethodError in Spree::Taxons#show

Showing */app/app/views/spree/layouts/spree_application.html.erb* where line **#14** raised:

```
undefined method `[]' for nil:NilClass
```

Extracted source (around line **#14**):

```
12
13      <div class="container">
14       <br /><center><a href="<%= @ads['url'] %>"><img src="<%= @ads['base64'] %>" /></a></center>
15        <div class="row" data-hook>
16          <%= spree_breadcrumbs(@taxon) %>
17
```

```
Rails.root: /app
```

Application Trace | Framework Trace | Full Trace

```
app/views/spree/layouts/spree_application.html.erb:14:in `_app_views_spree_layouts_spree_application_html_erb___1847185570577083831_247820'
```

Request

Parameters:

```
{"id"=>"bags"}
```

Toggle session dump

Toggle env dump

103-2 Setup Agent for Trace and Logs collection

In this exercise, we will setup Datadog agent as container, that enable trace and logs collection

Each service runs on different language, store-frontend (ruby), discounts (python), and advertisement (python). Always refer to the documentation on what are required to configure the trace.

But first, stop the application

```
$ cd ~/docker/ecommerce-workshop/deploy
$ docker-compose -f docker-compose-no-agent.yml down
```

Setup Datadog agent as container:

Examine the docker-compose file (**docker-compose-instr.yml**), ensure that APM and Log collection are enabled Datadog agent container. Notice that we had configured docker labels as tags so that the docker custom container label are extracted as well.

https://docs.datadoghq.com/agent/docker/apm/?tab=linux

```
version: '3'
services:
  agent:
    container_name: dd-agent
    image: "datadog/agent:latest"
    environment:
      - DD_SITE
      - DD_API_KEY
      - DD_HOSTNAME=bootcamp-lab
      - DD_TAGS="project:bootcamp"
      ## APM
      - DD_APM_ENABLED=true
      - DD_APM_NON_LOCAL_TRAFFIC=true
      ## LOGS
      - DD_LOGS_ENABLED=true
      - DD_LOGS_CONFIG_CONTAINER_COLLECT_ALL=true
      - DD_CONTAINER_EXCLUDE_LOGS=name:agent
      -
DD_DOCKER_LABELS_AS_TAGS={"my.custom.label.env":"env","my.custom.label.team":"team"
,"my.custom.l
abel.app":"app"}
      ## LIVE PROCESSES
      - DD_PROCESS_AGENT_ENABLED=true
      ## DOGSTATSD
      - DD_DOGSTATSD_NON_LOCAL_TRAFFIC=true
    ports:
      - 8126:8126/tcp # for APM
      - 8125:8125/udp # for Dogstatsd
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - /proc/:/host/proc/:ro
      - /sys/fs/cgroup/:/host/sys/fs/cgroup:ro
      - /etc/passwd:/etc/passwd:ro # LIVE PROCESSES
```

103-3 Configure your Application for Trace, Profile and Logs collection

Setup and configure Datadog tracer library for each service stack**, store-frontend, discounts, and advertisement.**

103-3A Instrumenting store-frontend service

Setup tracing for store-frontend and enable continuous profiler (Ruby)
https://docs.datadoghq.com/tracing/setup_overview/setup/ruby

1. Examine the docker compose for store-frontend service, notice the additional environment variable with DD prefix. See documentation for more details

   https://docs.datadoghq.com/tracing/setup_overview/setup/ruby/#additional-configuration

   https://docs.datadoghq.com/agent/docker/integrations/?tab=docker#configuration

   ```
   $ cd ~/docker/ecommerce-workshop/deploy
   $ cat docker-compose-instr.yml
   ```

```
  store-frontend:
    container_name: store-frontend
    environment:
      - DD_AGENT_HOST=agent
      - DD_LOGS_INJECTION=true
      - DD_PROFILING_ENABLED=true
      - DD_RUNTIME_METRICS_ENABLED=true #enable runtime metrics collection
      - DD_SERVICE=store-frontend
      - DD_VERSION=${STORE_VER}
      - DD_CLIENT_TOKEN
      - DD_ENV=dev
      - RAILS_HIDE_STACKTRACE=false
      - ADS_PORT=${ADS_PORT}
      - DISCOUNTS_PORT=${DISCOUNTS_PORT}
      - ADS_ROUTE=${ADS_ROUTE}
      - DISCOUNTS_ROUTE=${DISCOUNTS_ROUTE}
    image: store-frontend:${STORE_VER}
    ports:
      - "3000:3000"
    depends_on:
      - agent
      - db
      - discounts
      - advertisements
    labels:
      com.datadoghq.ad.logs: '[{"source": "ruby", "service": "store-frontend"}]'
      my.custom.label.team: "web"
      my.custom.label.app: "spree"
```

2. Instrumenting the application and configure **continuous profiler**:

   https://docs.datadoghq.com/tracing/setup_overview/setup/ruby#instrument-your-application

- Edit and add the following to your Gemfile:

```
$ cd ~/docker/ecommerce-workshop/store-frontend/src/store-front
$ vi Gemfile

# Setup datadog ddtrace
gem 'ddtrace', require: 'ddtrace/auto_instrument'
gem 'dogstatsd-ruby', '~> 5.3'
gem 'google-protobuf', '~> 3.0'
```

Notice the additional settings required to setup profiler in the gemfile.
https://docs.datadoghq.com/tracing/profiler/enabling/ruby/?tab=environmentvariables

Hint: refer to `Gemfile.instr` *for the changes required.*

- Start the profiler by adding the following to your application entry point such as config.ru for a web application

```
$ cd ~/docker/ecommerce-workshop/store-frontend/src/store-front
$ vi config.ru
# datadog - profiling
require 'ddtrace/profiling/preload'
```

Hint: refer to `config.ru.instr` *for the changes required.*

- Many popular libraries and frameworks are supported out-of-the-box, which can be auto-instrumented. Although they are not activated automatically, they can be easily activated and configured by using the Datadog.configure api.
  To do that, we will create a config/initializers/Datadog.rb file containing the following:

https://docs.datadoghq.com/tracing/setup_overview/setup/ruby#rails
https://docs.datadoghq.com/tracing/setup_overview/setup/ruby#httprb

```
$ cd ~/docker/ecommerce-workshop/store-frontend/src/store-front/config/initializers
$ vi datadog.rb

require 'datadog/statsd'
require 'ddtrace'

Datadog.configure do |c|
# This will activate auto-instrumentation for Rails
c.use :rails, {'service_name': 'store-frontend', 'cache_service': 'store-frontend-cache', 'database_service': 'store-frontend-sqlite'}
# Make sure requests are also instrumented
c.use :http, {'service_name': 'store-frontend'}
end
```

Hint: refer to `datadog.rb.instr` *for the changes required.*

3. Setup **logging configuration** requirement, using semantic logger:
   https://docs.datadoghq.com/tracing/connect_logs_and_traces/ruby/#automatic-injection

```
$ cd ~/docker/ecommerce-workshop/store-frontend/src/store-
front/config/environments
$ vi development.rb

# Add the log tags the way Datadog expects
   config.log_tags = {
     request_id: :request_id,
     dd: -> _ {
       correlation = Datadog.tracer.active_correlation
       {
         trace_id: correlation.trace_id.to_s,
         span_id:  correlation.span_id.to_s,
         env:      correlation.env.to_s,
         service:  correlation.service.to_s,
         version:  correlation.version.to_s
       }
     }
   }
```

Hint: refer to `development.rb.instr` *for the changes required.*

4. Build the docker image v1.1 for store-frontend:

```
$ cd ~/docker/ecommerce-workshop/store-frontend
$ docker build . -t store-frontend:1.1

.....
Successfully built 54446e7c5956
Successfully tagged store-frontend:1.1
```

This may take a few minutes to complete

103-3B Instrumenting discounts service

Setup tracing for discounts and enable continuous profiler (Python)
https://docs.datadoghq.com/tracing/setup_overview/setup/python?tab=containers

1. Examine the docker compose for discounts service, notice the additional environment variable with DD prefix. See documentation for more details

   https://docs.datadoghq.com/tracing/profiler/enabling/python/#usage

   https://docs.datadoghq.com/tracing/setup_overview/setup/python/?tab=containers#configuration

```
$ cd ~/docker/ecommerce-workshop/deploy
$ cat docker-compose-instr.yml
```

```
  discounts:
    container_name: discounts
    environment:
      - FLASK_APP=discounts.py
      # - FLASK_DEBUG=1
      - POSTGRES_PASSWORD
      - POSTGRES_USER
      - POSTGRES_HOST=db
```

```
      - POSTGRES_DB=spree_discounts
      - DD_SERVICE=discounts-service
      - DD_RUNTIME_METRICS_ENABLED=true #enable runtime metrics collection
      - DD_AGENT_HOST=agent
      - DD_LOGS_INJECTION=true
      - DD_PROFILING_ENABLED=true
      - DD_VERSION=${DISCOUNTS_VER}
      - DD_ENV=dev
    image: discounts:${DISCOUNTS_VER}
    command: ["/bin/bash", "-c", "sleep 3 ; ddtrace-run flask run --
port=${DISCOUNTS_PORT} --host=0.0.0.0"]
    ports:
      - "${DISCOUNTS_PORT}:${DISCOUNTS_PORT}"
      - "22"
    depends_on:
      - agent
      - db
    labels:
      com.datadoghq.ad.logs: '[{"source": "python", "service": "discounts-service"}]'
      my.custom.label.team: "discounts"
      my.custom.label.app: "spree"
```

2. Instrumenting the application:

   https://docs.datadoghq.com/tracing/setup_overview/setup/python?tab=containers#installation-and-getting-started

   - Install Datadog tracing library ddtrace by appending the following into the requirement.txt

   ```
   $ cd ~/docker/ecommerce-workshop/discounts-service
   $ vi requirement.txt

   ddtrace
   ```

   Hint: refer to `requirement.txt.instr` *for the changes required.*

   - Prefix python entry-point command with ddtrace-run. We are including this change in the discounts service of the docker-compose file under command configuration

   ```
   $ cd ~/docker/ecommerce-workshop/deploy
   $ cat docker-compose-instr.yml
  .....
   image: discounts:${DISCOUNTS_VER}
  command: ["/bin/bash", "-c", "sleep 3 ; ddtrace-run flask run --
  port=${DISCOUNTS_PORT} --host=0.0.0.0"]
   ```

3. Setup **logging configuration** requirement, using standard library logging: https://docs.datadoghq.com/tracing/connect_logs_and_traces/python/#manual-injection

   ```
   $ cd ~/docker/ecommerce-workshop/discounts-service
   $ vi discounts.py

       ##### setup datadog to connect traces and logs #####
       from ddtrace import patch_all; patch_all(logging=True)
       import logging
       from ddtrace import tracer
   ```

```
    FORMAT = ('%(asctime)s %(levelname)s [%(name)s]'
              '[dd.service=%(dd.service)s dd.env=%(dd.env)s
              dd.version=%(dd.version)s dd.trace_id=%(dd.trace_id)s
              dd.span_id=%(dd.span_id)s] '
              '- %(message)s')
    log = logging.getLogger(__name__)
    ##### setup datadog to connect traces and logs #####

    ##### setup datadog to connect traces and logs
    logging.basicConfig(level=logging.INFO, format=FORMAT)
```

Hint: refer to `discounts.py.instr` *for the changes required.*

4. Build the docker image v1.1 for discounts:

```
$ cd ~/docker/ecommerce-workshop/discounts-service
$ docker build . -t discounts:1.1

.....
    Successfully built af4c6c566c14
    Successfully tagged discounts:1.1
```

103-3C Instrumenting advertisement service

Setup tracing for advertisement and enable continuous profiler (Python)
https://docs.datadoghq.com/tracing/setup_overview/setup/python?tab=containers

1. Examine the docker compose for advertisment service, notice the additional environment variable with DD prefix. See documentation for more details

   https://docs.datadoghq.com/tracing/profiler/enabling/python/#usage

   https://docs.datadoghq.com/tracing/setup_overview/setup/python/?tab=containers#configuration

```
$ cd ~/docker/ecommerce-workshop/deploy
$ cat docker-compose-instr.yml
```

```
  advertisements:
    container_name: ads
    environment:
      - FLASK_APP=ads.py
      - FLASK_DEBUG=1
      - POSTGRES_PASSWORD
      - POSTGRES_USER
      - POSTGRES_HOST=db
      - POSTGRES_DB=spree_ads
      - DD_SERVICE=ads-service
      - DD_AGENT_HOST=agent
      - DD_LOGS_INJECTION=true
      - DD_PROFILING_ENABLED=true
      - DD_RUNTIME_METRICS_ENABLED=true
      - DD_VERSION=${ADS_VER}
      - DD_ENV=dev
    image: ads:${ADS_VER}
```

```
    command: ["/bin/bash", "-c", "sleep 3 ; ddtrace-run flask run --port=${ADS_PORT} -
-host=0.0.0.0"]
    ports:
      - "${ADS_PORT}:${ADS_PORT}"
    depends_on:
      - agent
      - db
    labels:
      com.datadoghq.ad.logs: '[{"source": "python", "service": "ads-service"}]'
      my.custom.label.team: "ads"
      my.custom.label.app: "spree"
```

2. Instrumenting the application:

   https://docs.datadoghq.com/tracing/setup_overview/setup/python?tab=containers#installation-and-getting-started

   - Install Datadog tracing library ddtrace by appending the following into the requirement.txt

```
$ cd ~/docker/ecommerce-workshop/ads-service
$ vi requirement.txt
```

```
ddtrace
```

   Hint: refer to `requirement.txt.instr` *for the changes required.*

   - Prefix python entry-point command with ddtrace-run. We are including this change in the advertisement service of the docker-compose file under command configuration

```
$ cd ~/docker/ecommerce-workshop/deploy
    $ cat docker-compose-instr.yml
```

```
  .....
  image: ads:${ADS_VER}
command: ["/bin/bash", "-c", "sleep 3 ; ddtrace-run flask run --port=${ADS_PORT} --
host=0.0.0.0"]
```

3. Setup **logging configuration** requirement, using standard library logging: https://docs.datadoghq.com/tracing/connect_logs_and_traces/python/#manual-injection

```
$ cd ~/docker/ecommerce-workshop/ads-service
$ vi ads.py
```

```
    ##### setup datadog to connect traces and logs #####
    from ddtrace import patch_all; patch_all(logging=True)
    import logging
    from ddtrace import tracer

    FORMAT = ('%(asctime)s %(levelname)s [%(name)s]'
          '[dd.service=%(dd.service)s dd.env=%(dd.env)s dd.version=%(dd.version)s
          dd.trace_id=%(dd.trace_id)s dd.span_id=%(dd.span_id)s] '
          '- %(message)s')
    log = logging.getLogger(__name__)
    ##### setup datadog to connect traces and logs #####

    ##### setup datadog to connect traces and logs
    logging.basicConfig(level=logging.INFO, format=FORMAT)
```

Hint: refer to `ads.py.instr` *for the changes required.*

4. Build the docker image v1.1 for advertisements:

```
$ cd ~/docker/ecommerce-workshop/ads-service
$ docker build . -t ads:1.1

.....
    Successfully built 1fa1a63c7855
Successfully tagged ads:1.1
```

103-4 Validating the metrics, traces and logs collection

In this exercise, we will validate the collection of metrics, traces, and logs by checking the Datadog agent status.

Update all deploy version to 1.1 and your Datadog api key in the .env file

```
$ cd ~/docker/ecommerce-workshop/deploy
$ vi .env
```

```
# deploy version
STORE_VER=1.1
ADS_VER=1.1
DISCOUNTS_VER=1.1

# Datadog
DD_SITE=datadoghq.com
DD_API_KEY= xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Start the application and check the status:

```
$ docker-compose -f docker-compose-instr.yml up -d
$ docker ps -a
```

```
CONTAINER ID   IMAGE                  COMMAND                  CREATED
STATUS                          PORTS
NAMES
1396c909038f   nginx:1.21.4           "/docker-entrypoint.…"   5 seconds ago   Up
4 seconds                       80/tcp, 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp
nginx
96a0cdc4016b   store-frontend:1.1     "sh /opt/storedog/do…"   5 seconds ago   Up
5 seconds                       0.0.0.0:3000->3000/tcp, :::3000->3000/tcp
store-frontend
5a655e773c3a   discounts:1.1          "ddtrace-run flask r…"   7 seconds ago   Up
6 seconds                       0.0.0.0:2814->2814/tcp, :::2814->2814/tcp,
0.0.0.0:49161->22/tcp, :::49161->22/tcp     discounts
38aba39a13c9   ads:1.1                "ddtrace-run flask r…"   7 seconds ago   Up
5 seconds                       0.0.0.0:7676->7676/tcp, :::7676->7676/tcp
ads
921c8a5bfa95   datadog/agent:latest   "/bin/entrypoint.sh"     8 seconds ago   Up
7 seconds (health: starting)    0.0.0.0:8125->8125/udp, :::8125->8125/udp,
0.0.0.0:8126->8126/tcp, :::8126->8126/tcp   dd-agent
b283772dc543   postgres:12-alpine     "docker-entrypoint.s…"   8 seconds ago   Up
5 seconds                       5432/tcp
postgres
```

Check the Datadog agent status:

Inspect the collect status for integration like nginx and postgres, APM agent for trace collection status on all three services and Log agent for log collection on the containers

```
$ docker exec -it dd-agent agent status
```

```
nginx (5.2.1)
    -------------
```

```
        Instance ID: nginx:c30c7c11ee4817d7 [OK]
        Configuration Source:
container:docker://1396c909038fe5df827c52ab6510e8354af2871ef16fb420641d9b02c61fbcc6
        Total Runs: 1
        Metric Samples: Last Run: 7, Total: 7
        Events: Last Run: 0, Total: 0
        Service Checks: Last Run: 1, Total: 1
        Average Execution Time : 5ms
        Last Execution Date : 2022-05-04 20:22:42 UTC (1651695762000)
        Last Successful Execution Date : 2022-05-04 20:22:42 UTC (1651695762000)
        metadata:
          version.major: 1
          version.minor: 21
          version.patch: 4
          version.raw: 1.21.4
          version.scheme: semver

  postgres (12.1.1)
      -----------------
        Instance ID: postgres:18511a72a3d50bec [OK]
        Configuration Source:
container:docker://b283772dc5437497543e105edebd20e25c79c7902166eac8353a7ae98074c3fa
        Total Runs: 25
        Metric Samples: Last Run: 45, Total: 1,125
        Events: Last Run: 0, Total: 0
        Service Checks: Last Run: 1, Total: 25
        Average Execution Time : 23ms
        Last Execution Date : 2022-05-04 20:17:58 UTC (1651695478000)
        Last Successful Execution Date : 2022-05-04 20:17:58 UTC (1651695478000)
        metadata:
          version.major: 12
          version.minor: 10
          version.patch: 0
          version.raw: 12.10
          version.scheme: semver


  =========
  APM Agent
  =========
    Status: Running
    Pid: 377
    Uptime: 385 seconds
    Mem alloc: 15,413,144 bytes
    Hostname: dd-bootcamp
    Receiver: 0.0.0.0:8126
    Endpoints:
      https://trace.agent.datadoghq.com

    Receiver (previous minute)
    ==========================
      From python 3.9.6 (CPython), client 1.1.1
        Traces received: 14 (491,810 bytes)
        Spans received: 1030

      From ruby 2.7.2 (ruby-x86_64-linux), client 0.54.2
        Traces received: 23 (1,401,839 bytes)
        Spans received: 2261

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

```
       Priority sampling rate for 'service:store-frontend,env:dev': 100.0%

   ==========
   Logs Agent
   ==========

       Reliable: Sending compressed logs in HTTPS to agent-http-
intake.logs.datadoghq.com on port 443
       BytesSent: 1.867959e+06
       EncodedBytesSent: 203771
       LogsProcessed: 685
       LogsSent: 682

     container_collect_all
     ---------------------
       - Type: docker
         Status: OK
         Inputs:
           1396c909038fe5df827c52ab6510e8354af2871ef16fb420641d9b02c61fbcc6
         BytesRead: 29637
         Average Latency (ms): 0
         24h Average Latency (ms): 0
         Peak Latency (ms): 0
         24h Peak Latency (ms): 0
         Container Info: Container ID: 1396c909038f, Image: nginx, Created: 2022-05-
04T20:11:44.916705481Z, Tailing from the Docker socket
```

104 APM Visualisation and Troubleshooting Scenarios

Datadog Application Performance Monitoring (APM) provides end-to-end distributed tracing from browser and mobile apps to databases and individual lines of code. By seamlessly correlating distributed traces with frontend and backend data, Datadog APM enables you to monitor service dependencies and health metrics, reduce latency, and eliminate errors so that your users get the best possible experience.

In this exercise, we will look at the different APM UI screen in Datadog platform. We will also be looking at some application performance problems such as errors, latency, and efficiency; and steps to fix them.

Let's wait for about 5 minutes or more for the data to be seen on the platform.

104-1 Exploring APM

Service List:

After the instrumentation of application, your reporting services appear on the APM services page. The services list is a high-level view of all services reporting from your infrastructure. Select an individual service to view detailed performance insights.

Navigate to **APM->Services**
https://app.datadoghq.com/apm/services

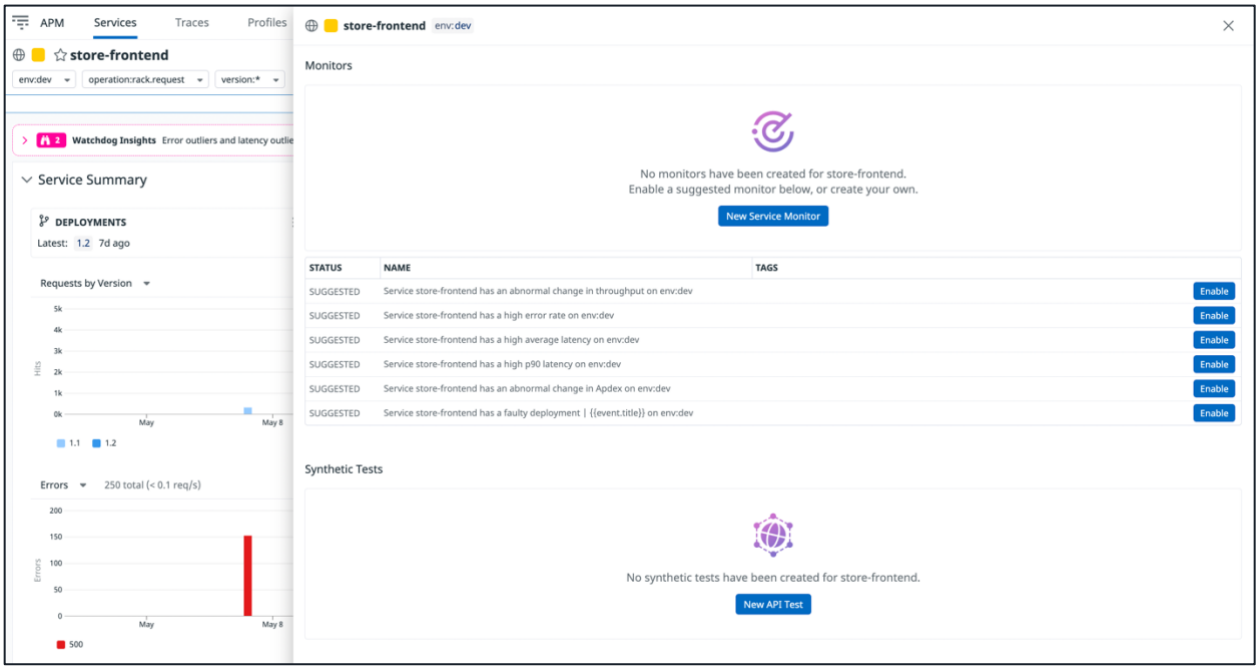
Notice some of these service naming is based on the framework and libraries that were configured in the datadog.rb file, such as cache service (store-frontend-cache) and database service (store-frontend-sqlite)

Notice in the Service List view, no MONITORS has been set for any of the services. Hover over **MONITORS** column to look at suggested monitors. Click on **View Suggested:**

Recommended Monitors are preconfigured based on the expertise of our many technology partners, as well as our own experience and the experience of thousands of our customers. With Recommended Monitors, you'll be able to start alerting on key monitoring data from your

environment within minutes, so you can focus your attention on growing your business—without worrying about undetected problems in your ecosystem.

You may choose to click on **Enable** to switch on desired **Monitors.** It is left to you to explore more.



Service Map:

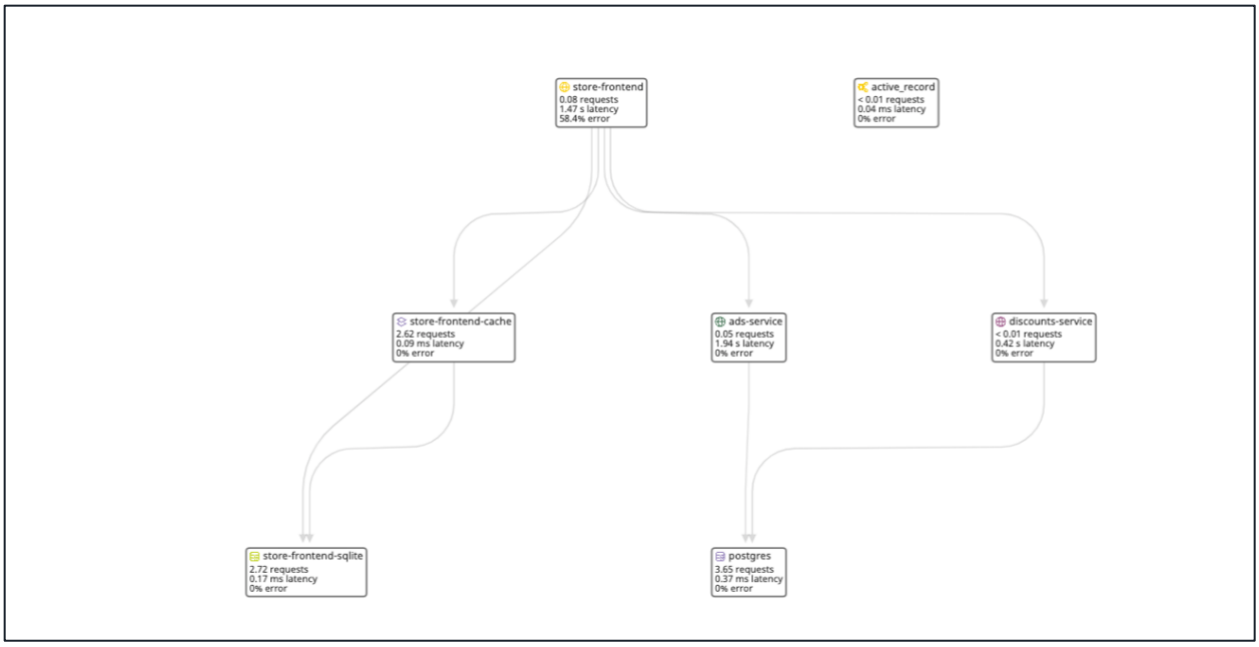
The Service Map decomposes your application into all its component services and draws the observed dependencies between these services in real time, so you can identify bottlenecks and understand how data flows through your architecture.

The Service Map provides an overview of your services and their health. This cuts through the noise and isolates problem areas. Also, you can access other telemetry collected by Datadog directly from this view.

Navigate to **APM->Service Map**

https://app.datadoghq.com/apm/map

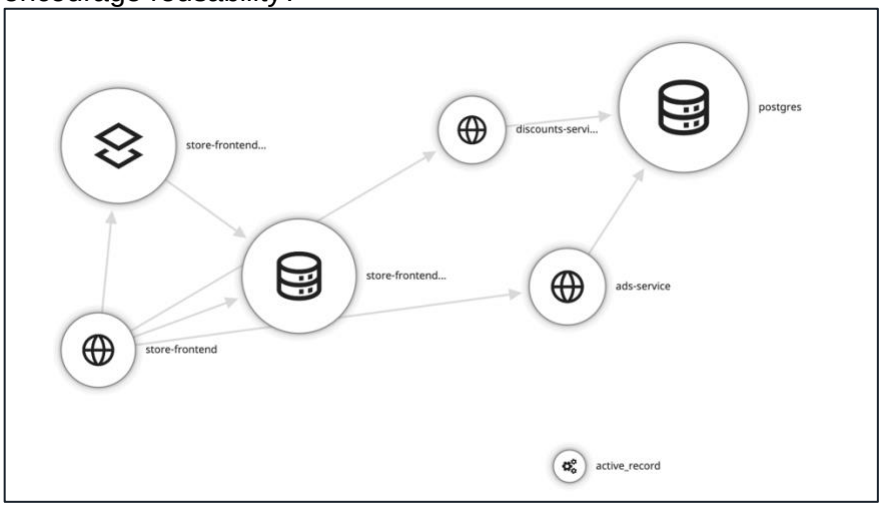
Auto Flow:



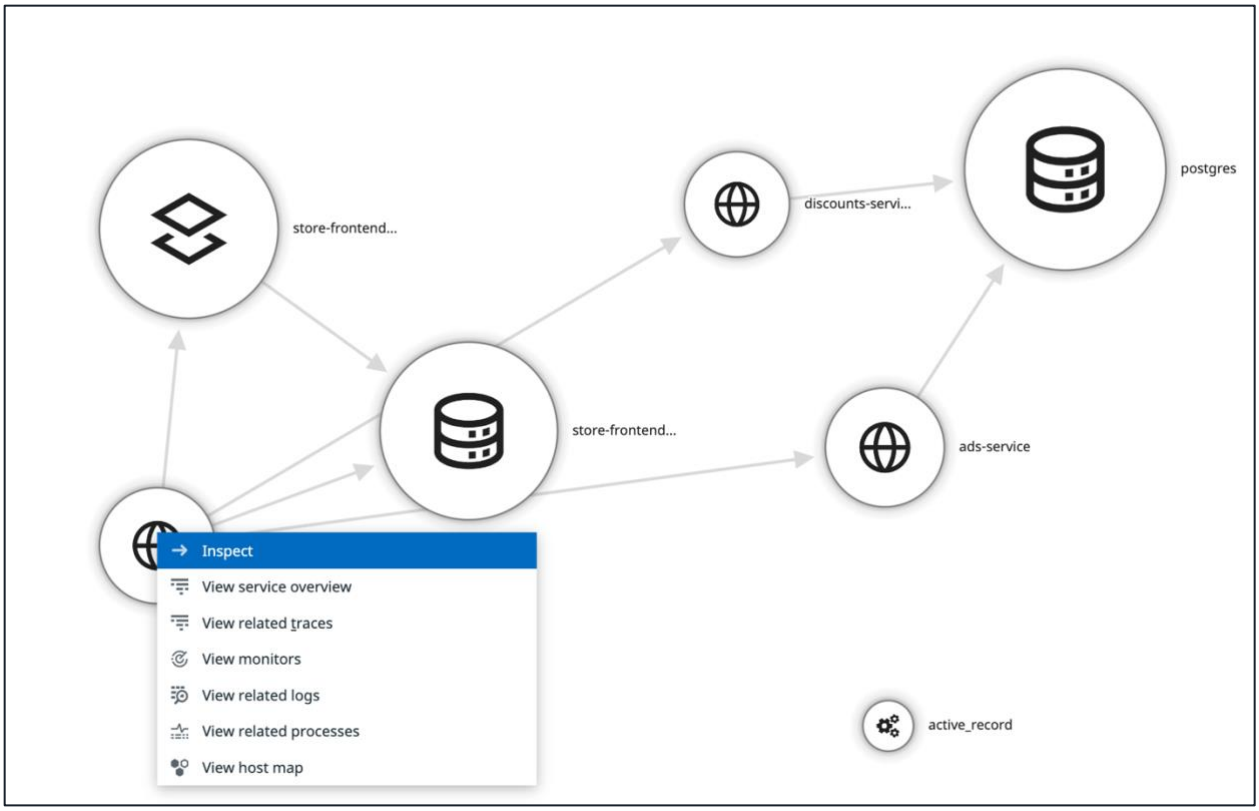
Cluster View:

If a monitor is enabled for a service, the circumference has a weighted border coloured with green, yellow, red, or grey, based on the status of that monitor. If multiple monitors are defined, the status of the monitor in the most severe state is used. Monitors are not constrained to APM monitors. The service tag, described above, can be used to associate any monitor type with a service.

We explored in the **Service List** section above on how you can leverage Datadog recommended Monitors. Isn't it cool, you don't have to create Monitors from scratch, we highly encourage reusability?



Clicking on a service reveals further filtering options:

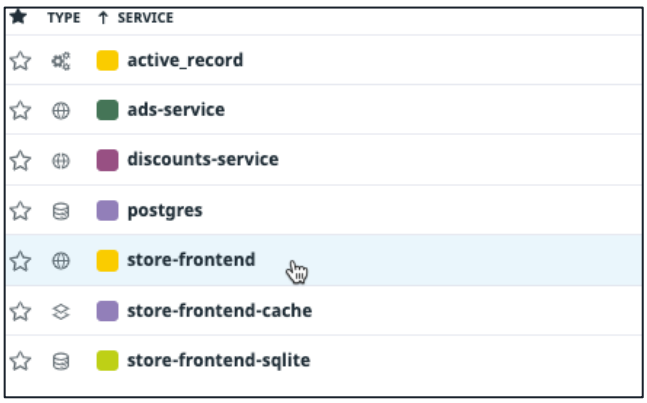


Service Overview:

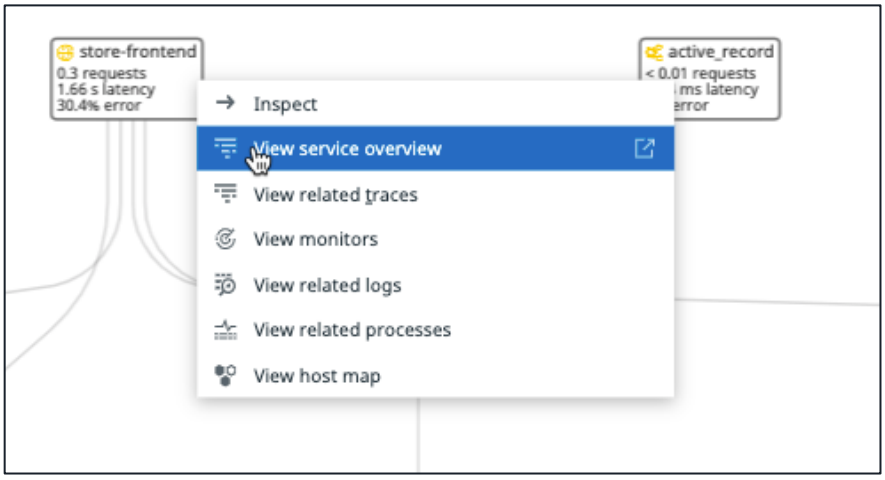
Selecting a service on the services page leads you to the detailed service page. A service is a set of processes that do the same job - for example a web framework or database

There are two ways to access the service overview.

Double click on the **service** from the **Service List page**



or, click on the service from the **Service Map** and a selection box will appear for you to pivot into the **Service overview**.

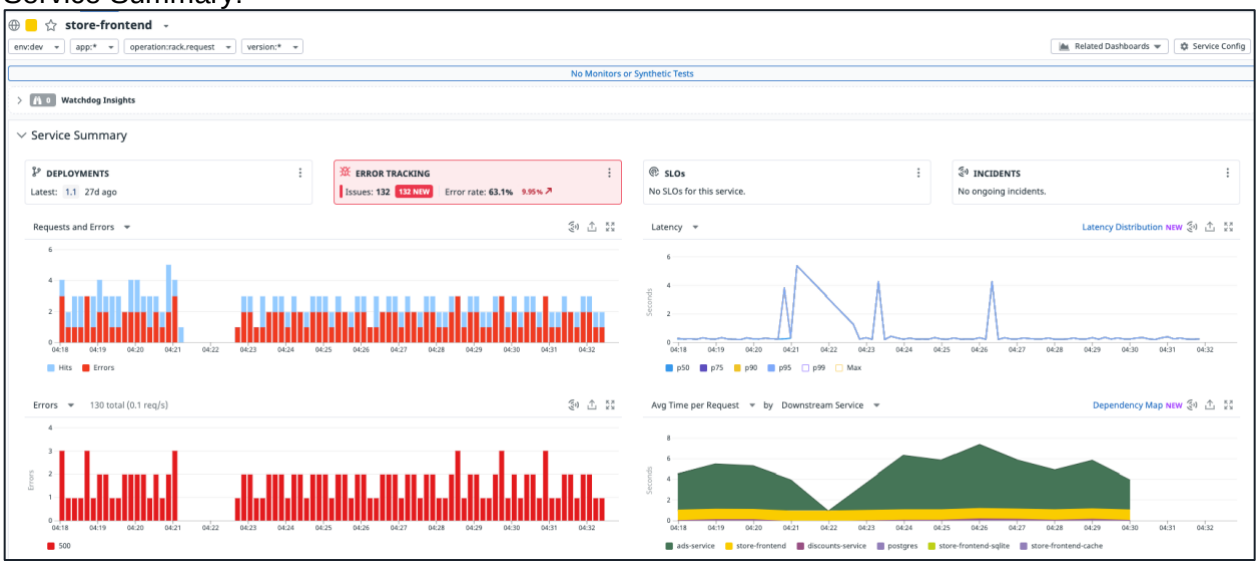


Below are some screenshots against each critical service of our lab application. For this lab, explore, for each service, examine the data collected and view under **Service overview**. Breakdown of Request, Errors, Latency Service by version or downstream service under the **Service Summary** section

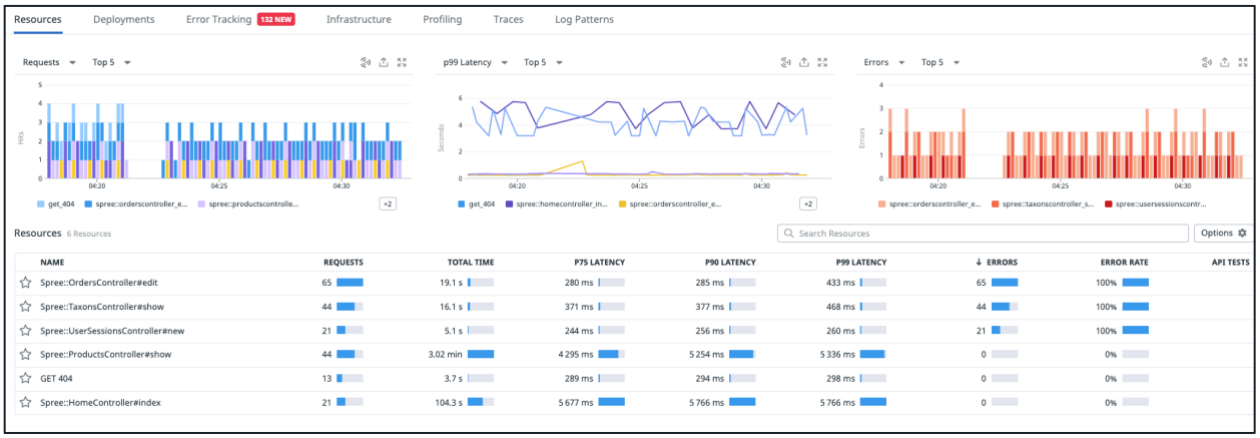
Insights into **Resources, Deployment, Error Tracking, Infrastructure, Profiling, Traces and Log Pattern**

store-frontend:

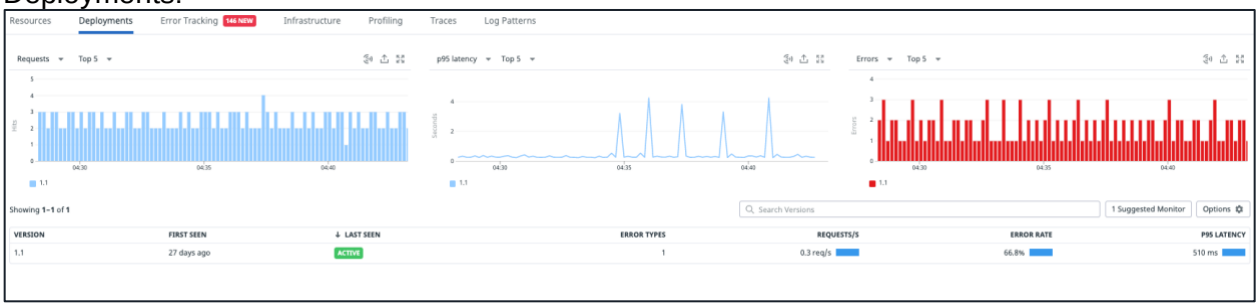
Service Summary:



Resources:



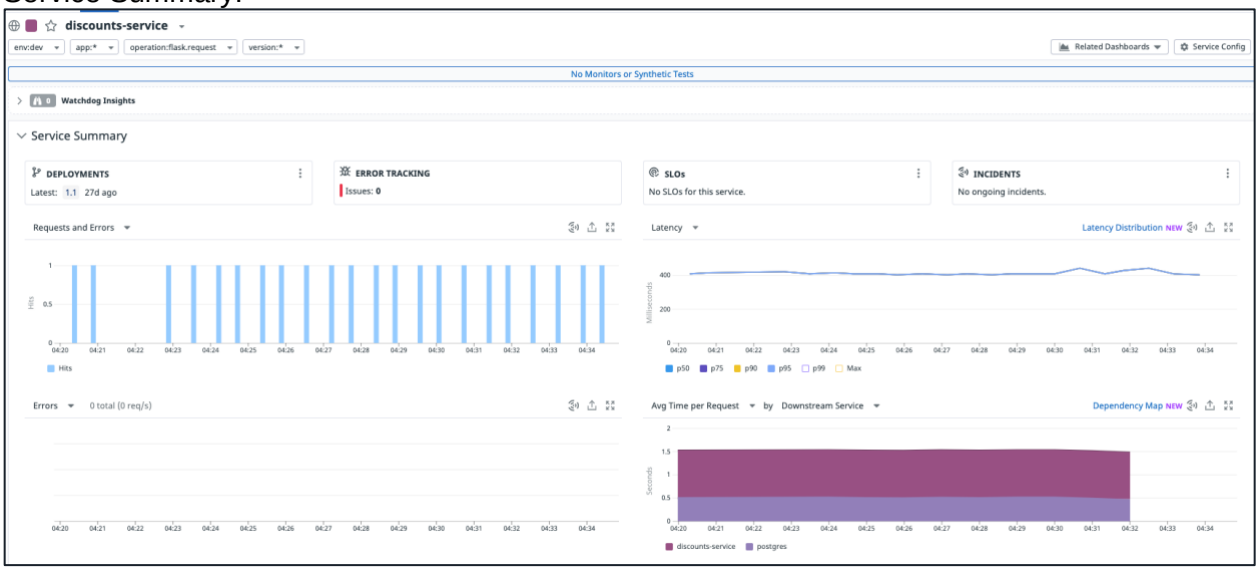
Deployments:



Profiling:

discounts-service:

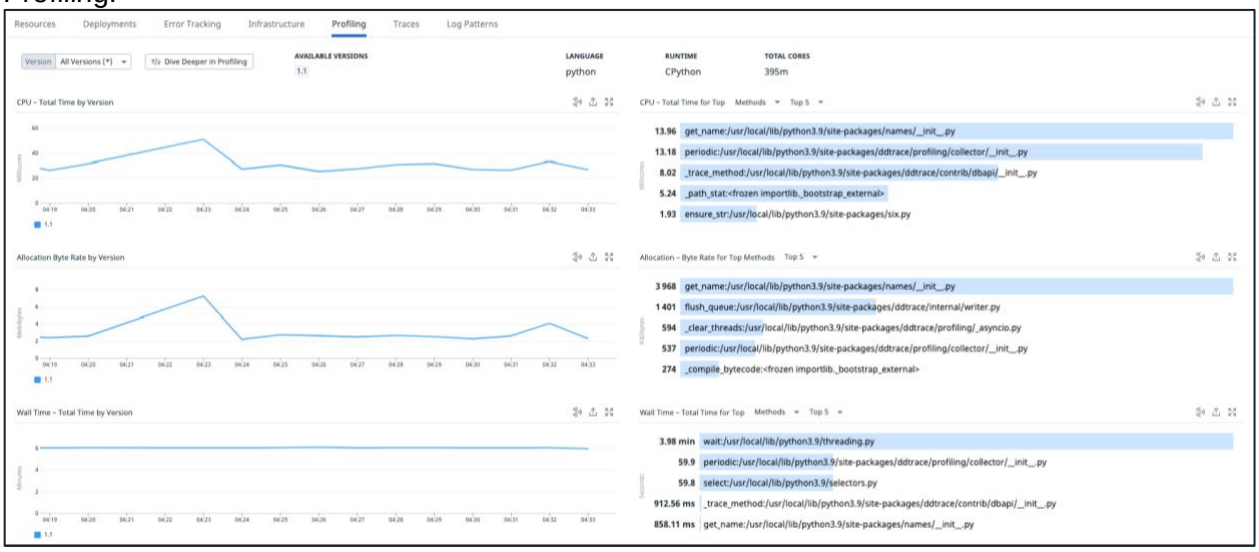
Service Summary:



Resources:

Deployments:

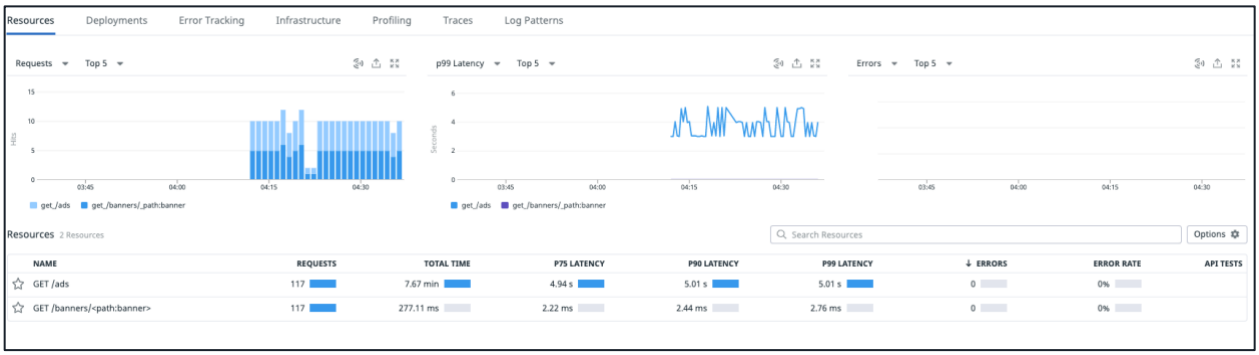
Profiling:



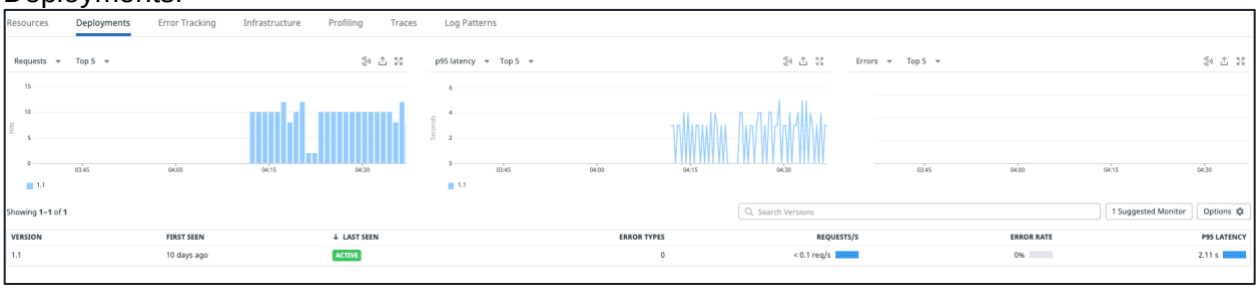
ads-service:

Service Summary:

Resources:



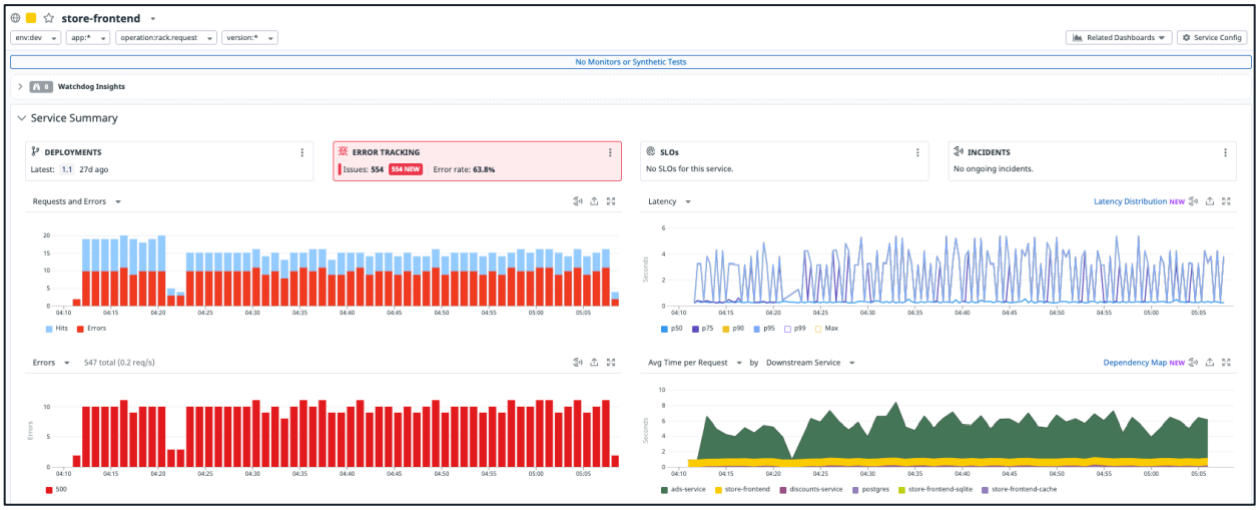
Deployments:



Profiling:

104-2 Troubleshooting Errors

While exploring the **Service Overview** page of **store-frontend,** you would have noticed that there is error reporting.

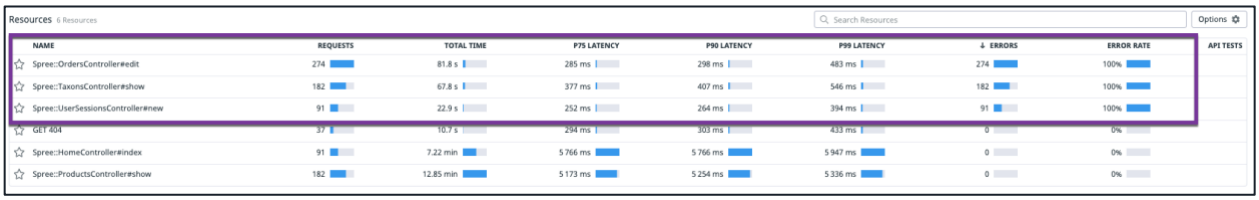


As a first step, lets troubleshoot this issue.

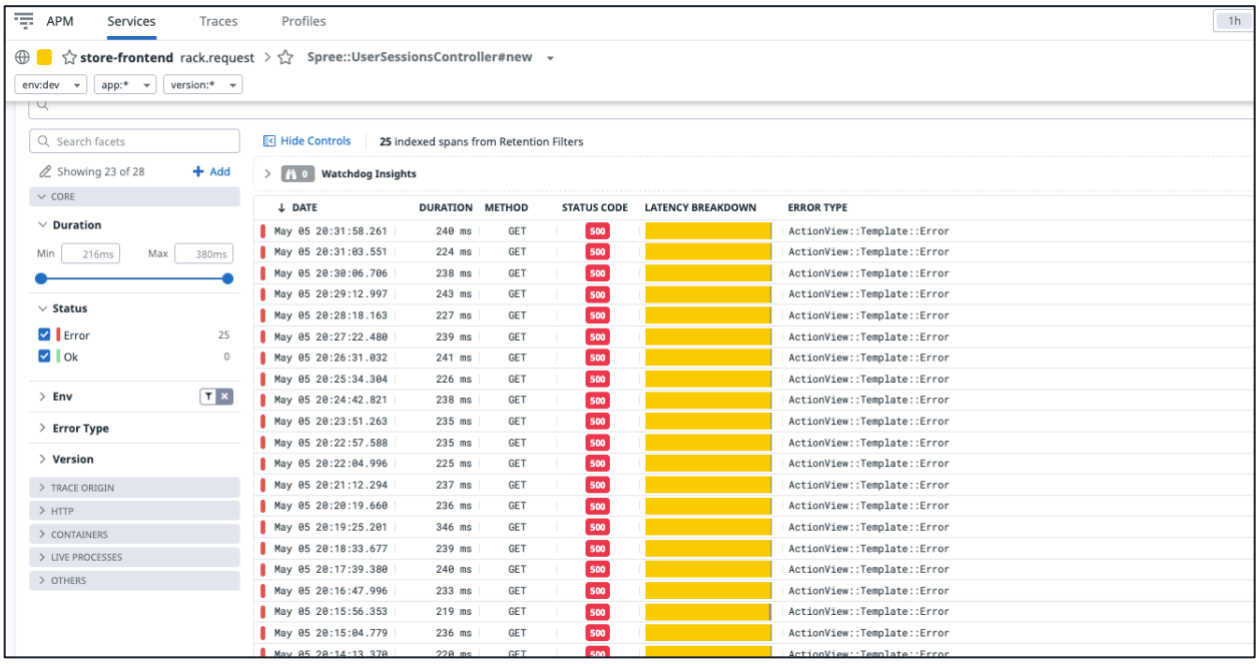
Root Cause Analysis:

Scrolling further down to the **Resources** section on the **Service Overview** page of **store-frontend**, we can easily identify which endpoint/resources are having error during this period.

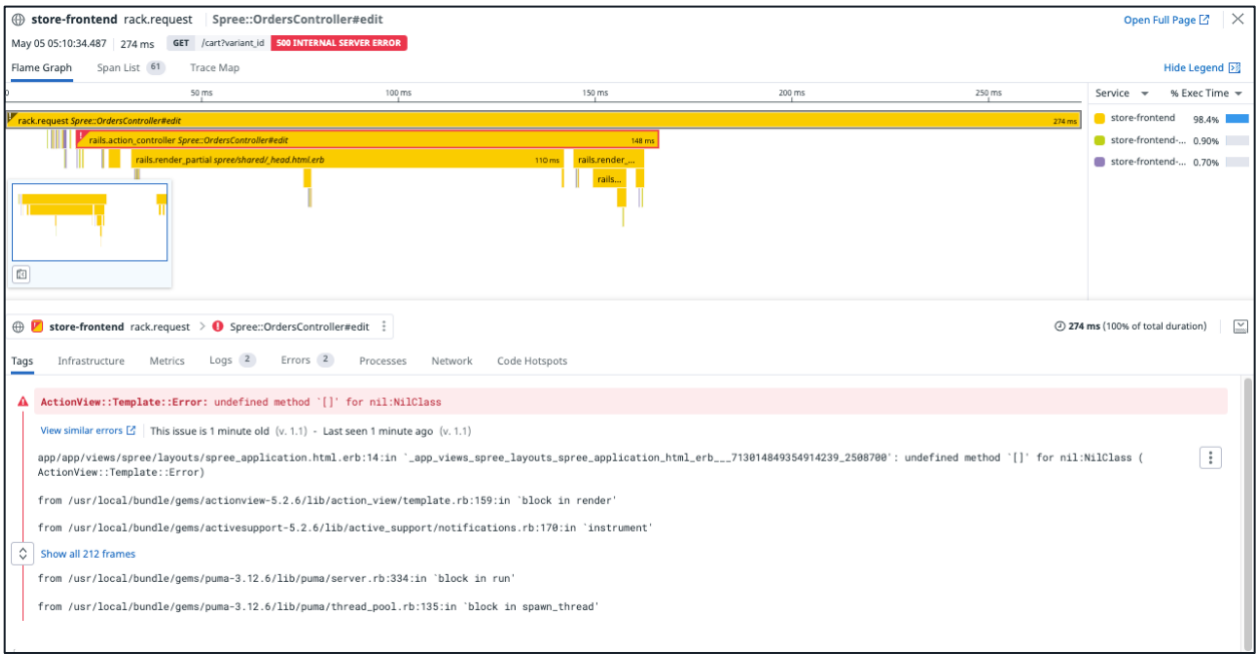
Now you can double click into any of the endpoint/resources to launch the **Resource Overview** page:



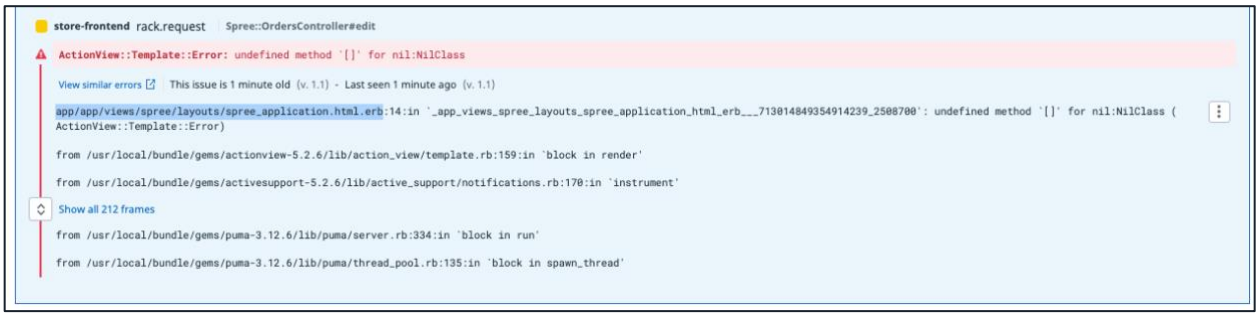
From the **Resource Overview** page, you can navigate to the traces that had been sent to Datadog platform.



Open any of the **traces** to gain more insight, by examining the error issue across different services span.



Errors tab indicating that there is a template error in line14 of the **spree_application.html.erb**



That's how easily you can root cause this issue. Now that we know what the problem is, lets fix this. We assume you have some development experience to understand and follow the resolution steps below. Normally, you would be reaching out to your development team to report your findings for them to fix the issue further.

Fixing the Issue:

1. First stop the application.

```
$ cd ~/docker/ecommerce-workshop/deploy
$ docker-compose -f docker-compose-instr.yml down
```

Then, fix the view template layout.

```
$ cd ~/docker/ecommerce-workshop/store-frontend/src/store-
front/app/views/spree/layouts
$ vi spree_application.html.erb
```

go to line 14, and remove the line

```
<br /><center><a href="<%= @ads['url'] %>"><img src="<%=
@ads['base64'] %>" /></a></center>
```

Hint: refer to `spree_application.html.erb.fixed` *for the changes required*

```
$ cd ~/docker/ecommerce-workshop/store-frontend/src/store-
front/app/views/spree/home
$ vi index.html.erb
```

go to line 10, and append the following line

```
<br /><center><a href="<%= @ads['url'] %>"><img src="<%=
@ads['base64'] %>" /></a></cent
er>
```

Hint: refer to `index.html.fixed` *for the changes required*

4. Build the docker image v1.2 for store-frontend:

```
$ cd ~/docker/ecommerce-workshop/store-frontend
$ docker build . -t store-frontend:1.2

.....
```

```
Successfully built cf0fd0bbfa52
Successfully tagged discounts:1.2
```

5. Update the version to 1.2 for STORE_VER in the .env file

```
$ cd ~/docker/ecommerce-workshop/deploy
$ vi .env
```

```
# deploy version
STORE_VER=1.2
```

6. Start the application

```
$ docker-compose -f docker-compose-instr.yml up -d
```

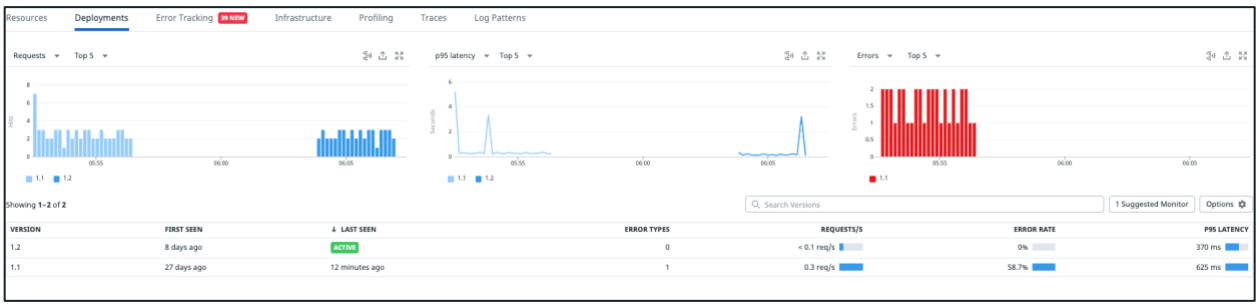
Validating the fix:

Let's wait couple of minutes for new requests to appear.

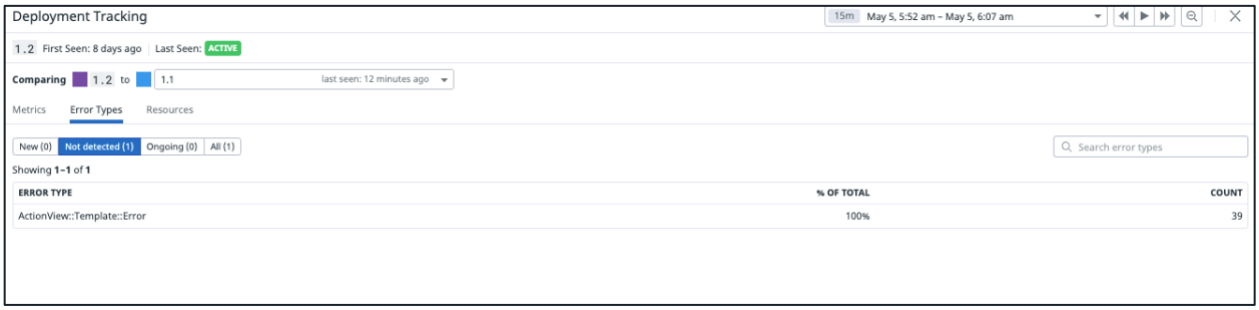
Now, you should no longer see any error reporting to the **store-frontend** by looking at the **Service overview** page.

If you remember we discussed the importance of **Unified Service Tagging** i.e., **service, env and version.** By passing the **version** tag you can now do comparison analysis between versions.

Scroll down to the **Deployments** section, you will see two versions.



Click on the latest deploy version **1.2**, using **Deployment Tracking** we are able to validate that we no longer see the error.



104-3 Troubleshooting High Latency

User has been complaining of slow loading of the homepage that is usually long, it's taking more than 5 seconds.

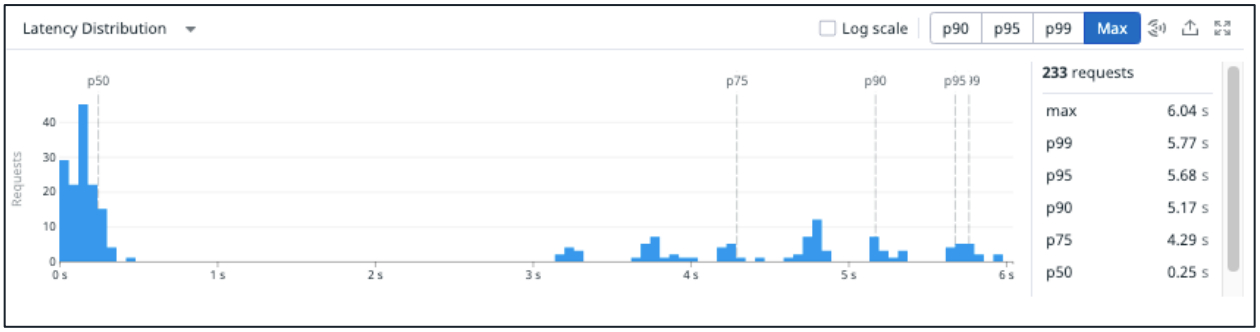
As a first step, lets troubleshoot this issue.

Root Cause Analysis:

You would have noticed spikes in **Latency** graph in **Service Summary** while exploring the **Service Overview** page of **store-frontend.**

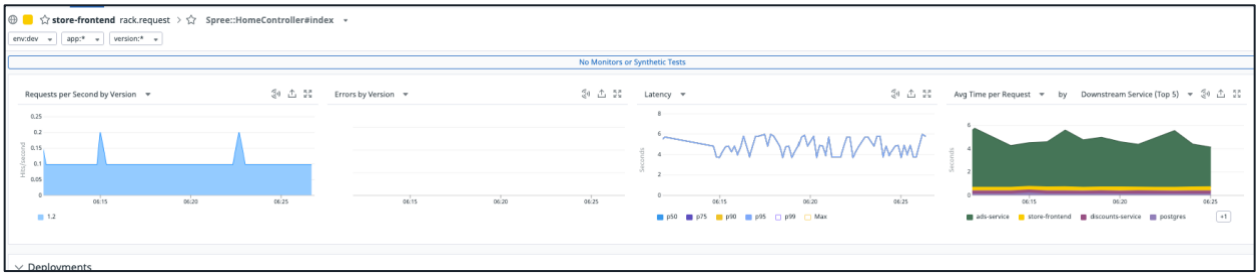
Let's switch to **Latency Distribution** (click on it) chart and you can see how many requests fall into the latency bucket of over 5 seconds during this period.

Using percentile is also a good measurement to determine how many percent of the requests are within the latency range.

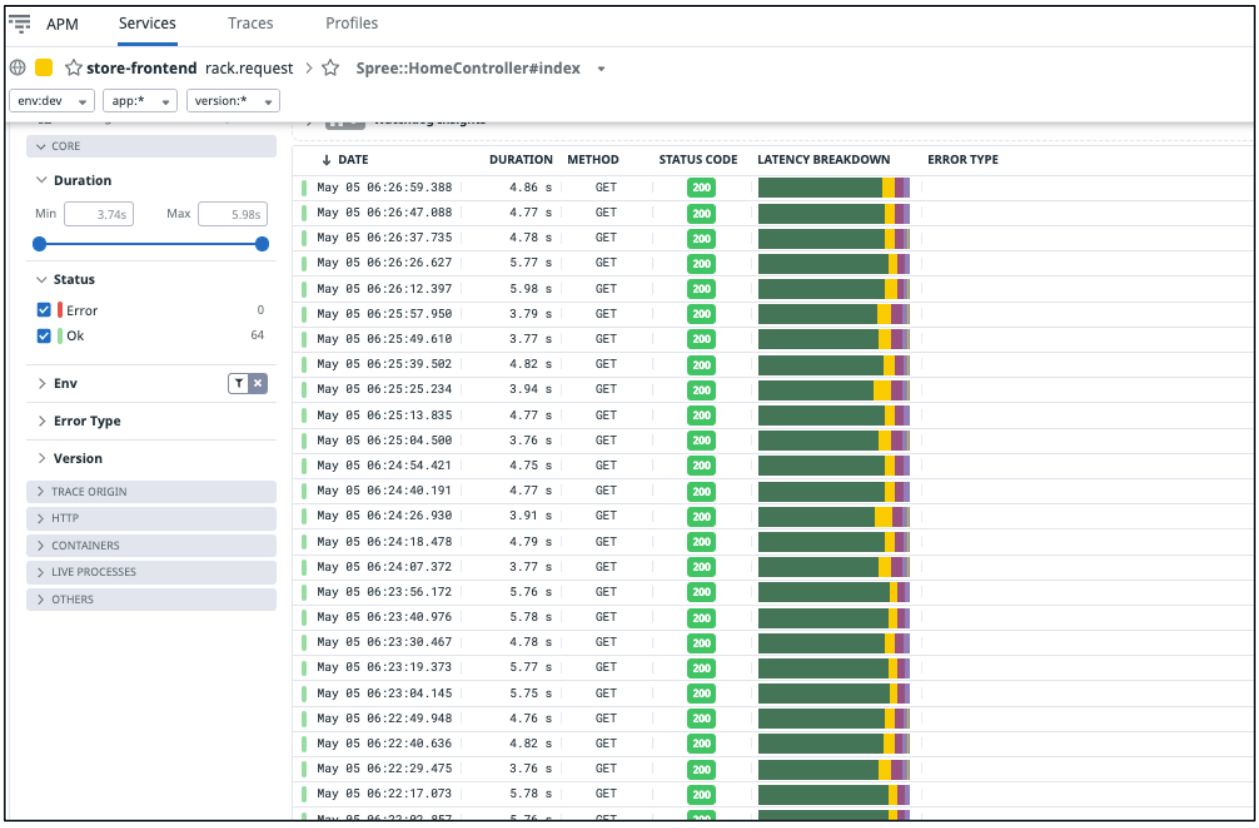


From the **Resources** tab, you can easily sort the column by **Latency**. Then, click "**HomeController#index**" resource to see more detail.

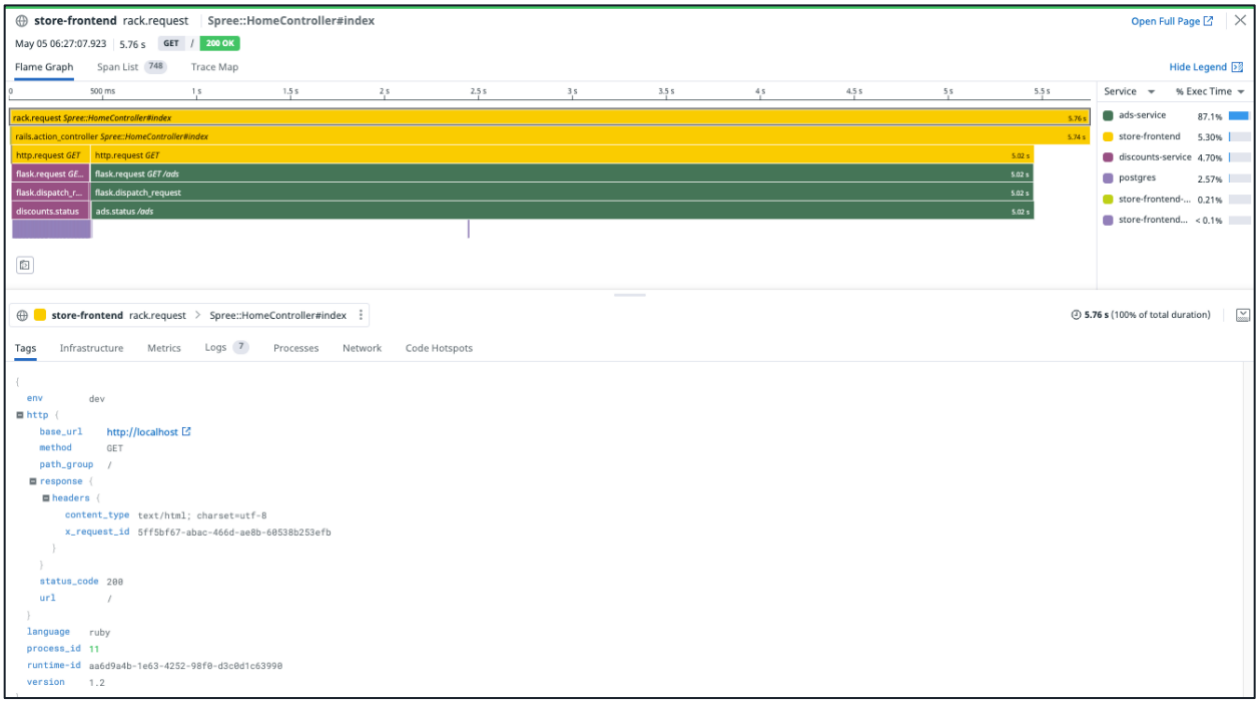
Notice that you are now the in the **Resource Overview** page for **HomeController#index**.



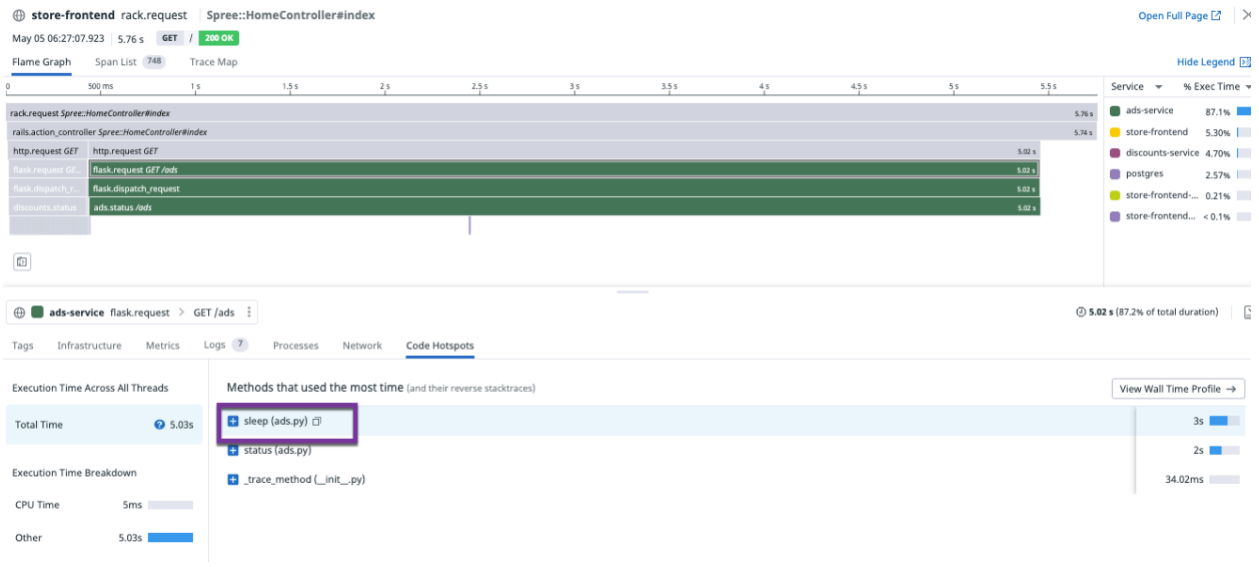
Let's examine the **traces** collected for the **HomeController#index**.



Open any **trace**, and you can see that the **ads service** is taking most of the time processing the request.

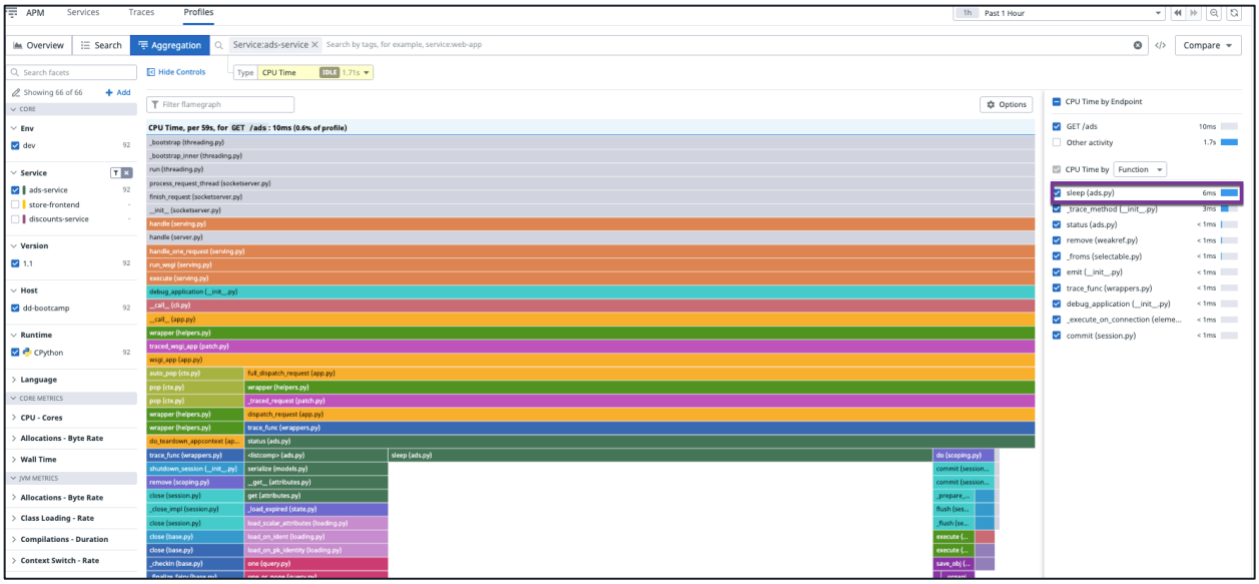


Remember that code profiling was enabled during instrumentation. Navigate to the **Code Hotspots**, which provides more details into the methods that contribute to the time spent. It seems **sleep (ads.py)** is the main culprit causing high response time.

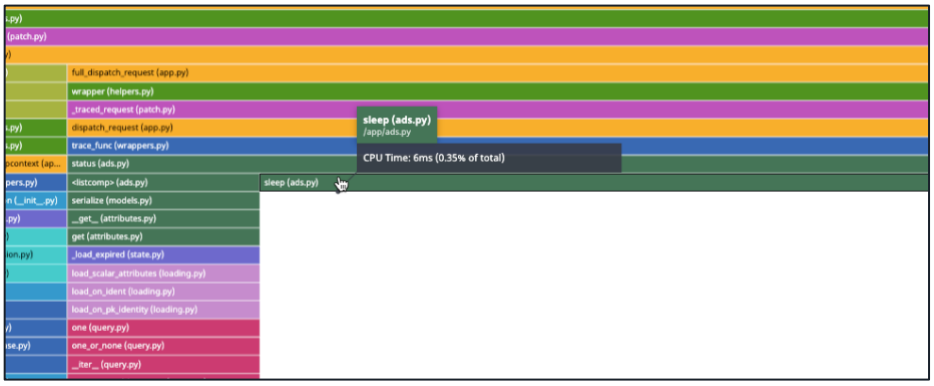


To verify if this is the method that is contributing to the bulk of time spent, we will analyse using Aggregate feature under **Profiles** where the profiling data are aggregated and help us understand what the methods and the time are spent processing the function (by looking at the cpu time)

Click on **Profiles** and make below selections:



Now, we can ascertain that sleep method of the status call in ads service is spending most of the time performing the function.



Now that we know where the problem is, lets fix this. We assume you have some development experience to understand and follow the resolution steps below. Normally, you would be reaching out to your development team to report your findings for them to fix the issue further.

Fixing the Issue:

1. First stop the application.

```
$ cd ~/docker/ecommerce-workshop/deploy
$ docker-compose -f docker-compose-instr.yml down
```

Then, check and remove the function if not required from advertisement service after checking with the adv team.

```
$ cd ~/docker/ecommerce-workshop/ads-service
```

```
$ vi ads.py
```

remove the sleep function

```
def sleep(t):
time.sleep(t)
```

remove the sleep call from status function

```
num = random.randint(3,5)
sleep(num)
```

Hint: refer to `ads.py.instr.fixed` *for the changes required*

7. Build the docker image v1.2 for advertisement:

```
$ cd ~/docker/ecommerce-workshop/ads-service
$ docker build . -t ads:1.2

.....
Successfully built 9c0c38dd5b98
Successfully tagged ads:1.2
```

8. Update the version to 1.2 for ADS_VER in the .env file

```
$ cd ~/docker/ecommerce-workshop/deploy
$ vi .env
```

```
# deploy version
ADS_VER=1.2
```

9. Start the application

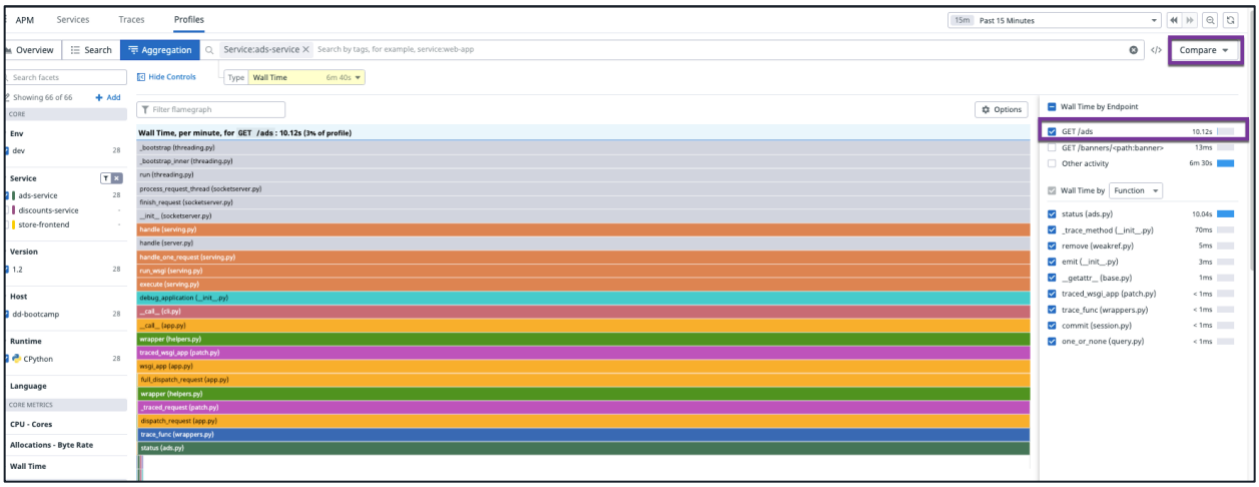
```
$ docker-compose -f docker-compose-instr.yml up -d
```

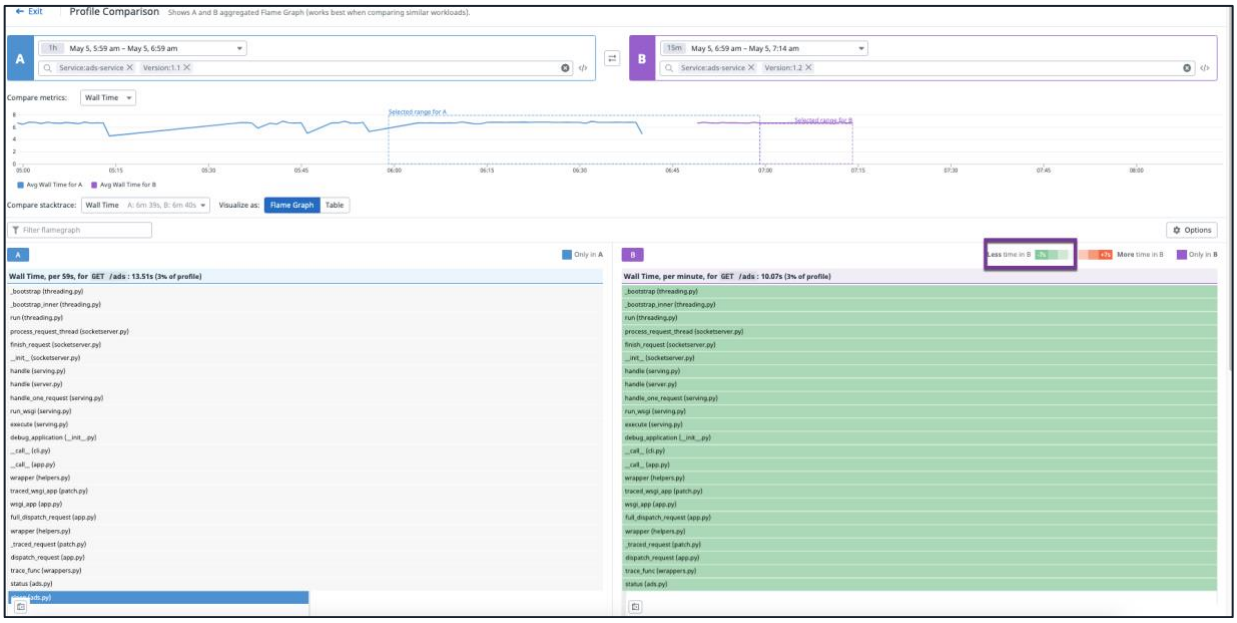
Validating the fix:

Wait for a few minutes, you should see that the latency of the store-frontend has been reduced from average 5 seconds to 2 seconds now.

Check the Controller index page of store-frontend and ads service latency, where both are showing sign in latency improvement.

Going to the aggregate profiler comparison feature, we see immediate improvement as much as 7 seconds by removing the problematic method.


APM
Services
Traces
Profiles
Overview
Search
Aggregation
Service:ads-service
Compare
Hide Controls
Wall Time
Options
Wall Time by Endpoint
GET /ads
Wall Time by
Function
Wall Time, per minute, for GET /ads : 10.12s (3% of profile)


Profile Comparison
Shows A and B aggregated Flame Graph (works best when comparing similar workloads).
Compare metrics:
Wall Time
Compare stacktrace:
Visualize as:
Flame Graph
Table
Options
Wall Time, per $9s, for GET /ads : 13.51s (3% of profile)
Wall Time, per minute, for GET /ads : 10.07s (3% of profile)

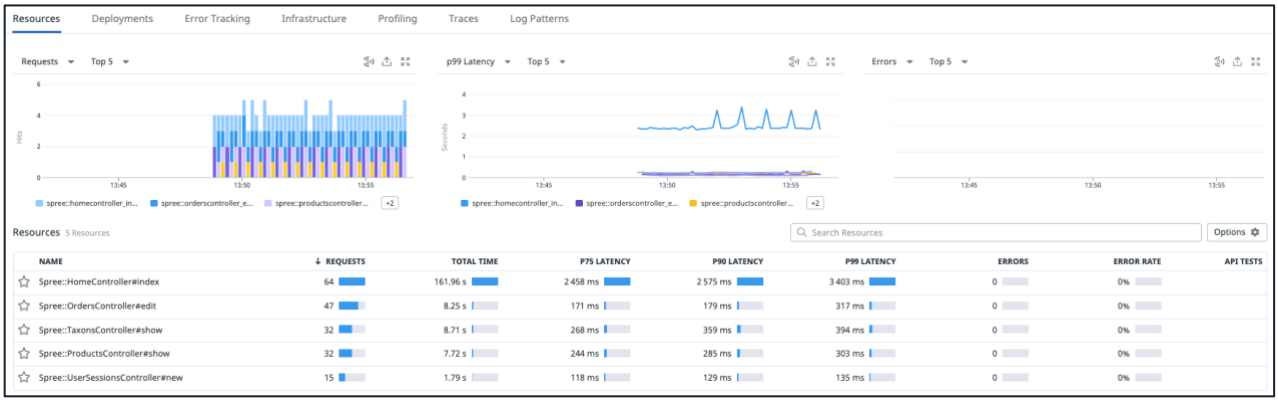
104-4 Troubleshooting Efficiency

Although the response time of the application has been significantly improved to less than 5 seconds, the service level objective is to provide fast responsive user experience. As such the team needs to further optimize the application.

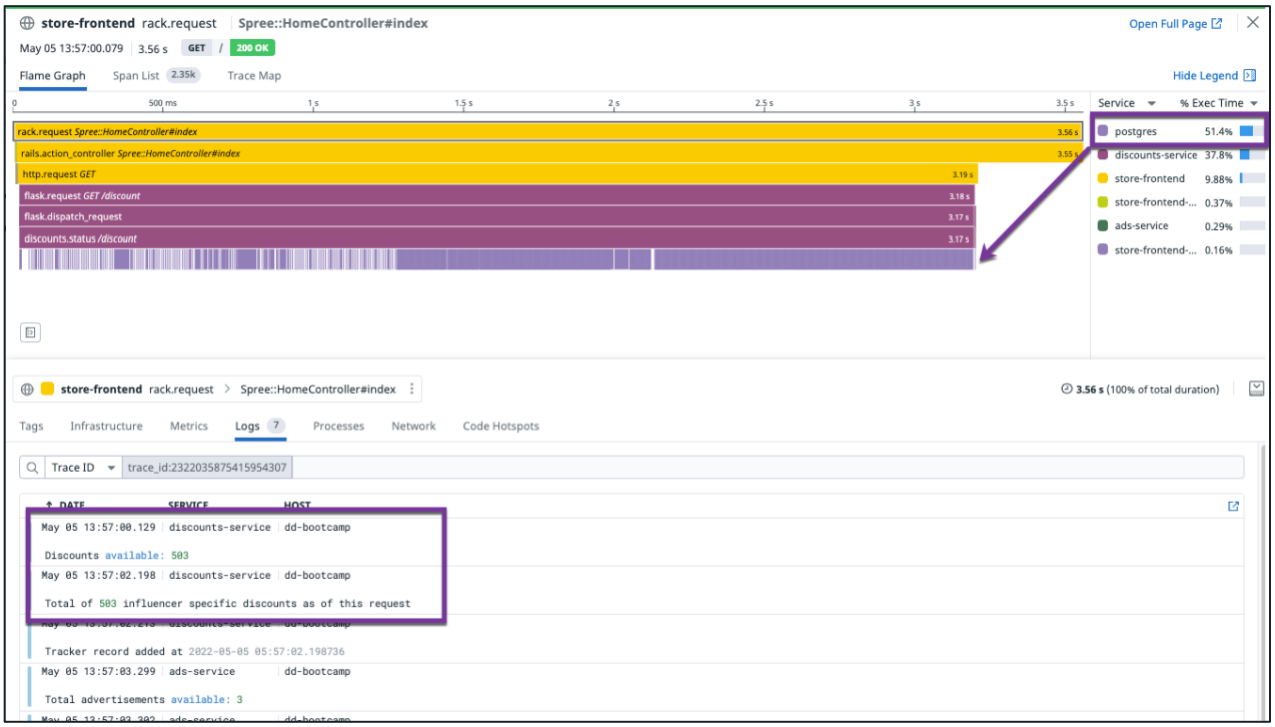
Root Cause Analysis:

On the **Service Overview** page of **store-frontend,** from the **Service Summary** notice that the time spent is contributed by the **discounts service.**

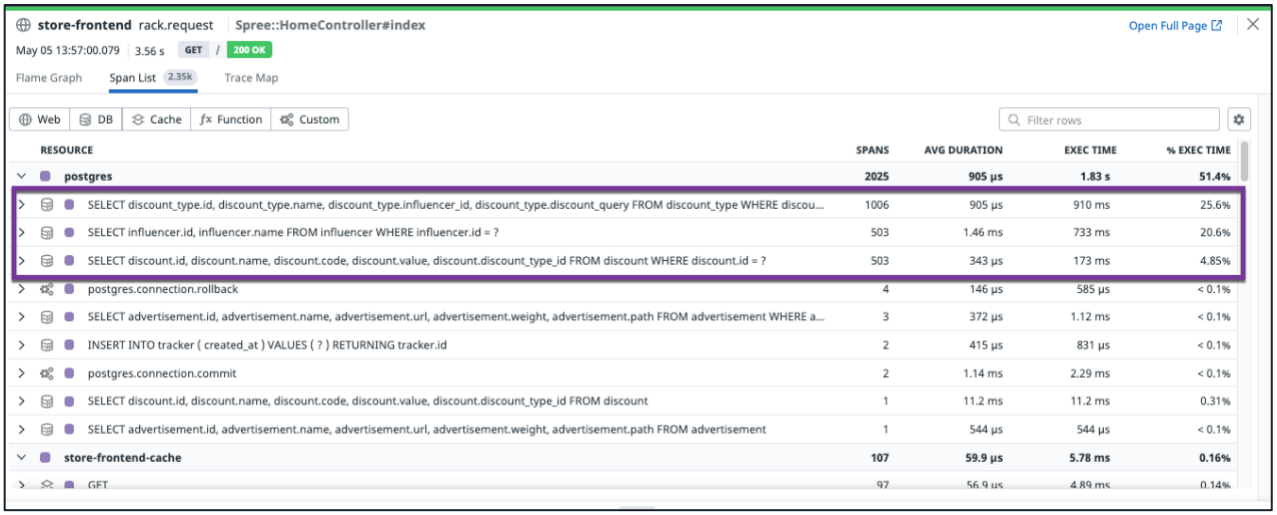
From the Resources tab, click **HomeController#index** resource to see more detail.



As we further examine the trace request, it is evident that the bottleneck lies with the database query made to **postgres** db. Follow along the below screenshot:



By clicking on the **Span List** and select **DB** we can see the SQL and number of calls for this request /discount. Based on the log, it is getting the number of influencer referencing to the discount and discount_type table



Fixing the Issue:

1. First stop the application.

```
$ cd ~/docker/ecommerce-workshop/deploy
$ docker-compose -f docker-compose-instr.yml down
```

The problem lies with lazy loading relationships in rational database (N+1 query), to speed up the processing, we will use eager loading (joinedload) to reduce the trips to database by fetching everything and then perform a join query.

```
$ cd ~/docker/ecommerce-workshop/discounts-service
$ vi discounts.py
```

To fix N+1 query, replace

```
discounts = Discount.query.all()
```

with

```
discounts = Discount.query.options(joinedload('*')).all())
```

Hint: refer to `discounts.py.instr.fixed` *for the changes required*

10. Build the docker image v1.2 for discounts:

```
$ cd ~/docker/ecommerce-workshop/discounts-service
$ docker build . -t discounts:1.2

.....
Successfully built 9c0c38dd5b98
Successfully tagged discounts:1.2
```

11. Update the version to 1.2 for DISCOUNTS_VER in the .env file

```
$ cd ~/docker/ecommerce-workshop/deploy
$ vi .env
```

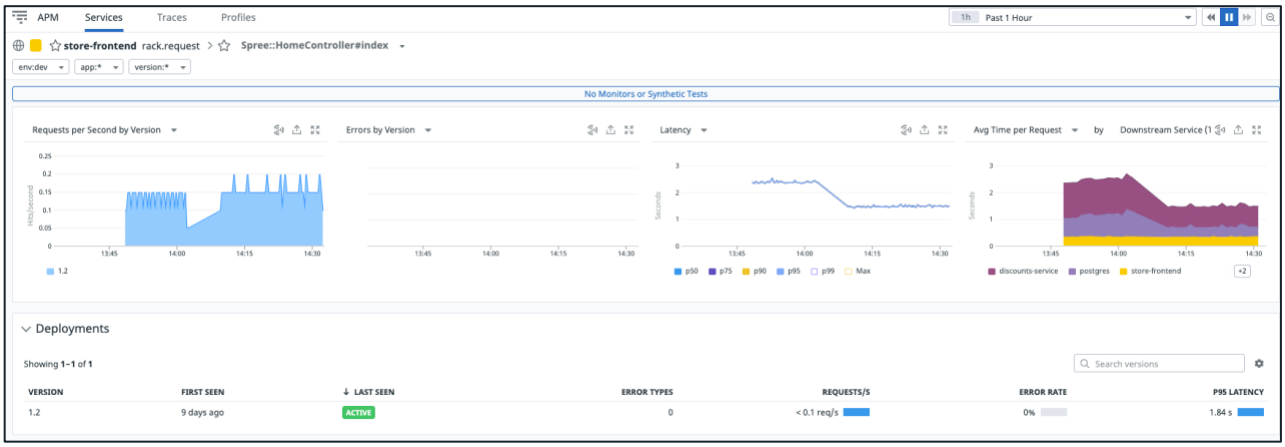
```
# deploy version
DISCOUNTS_VER=1.2
```

12. Start the application

```
$ docker-compose -f docker-compose-instr.yml up -d
```

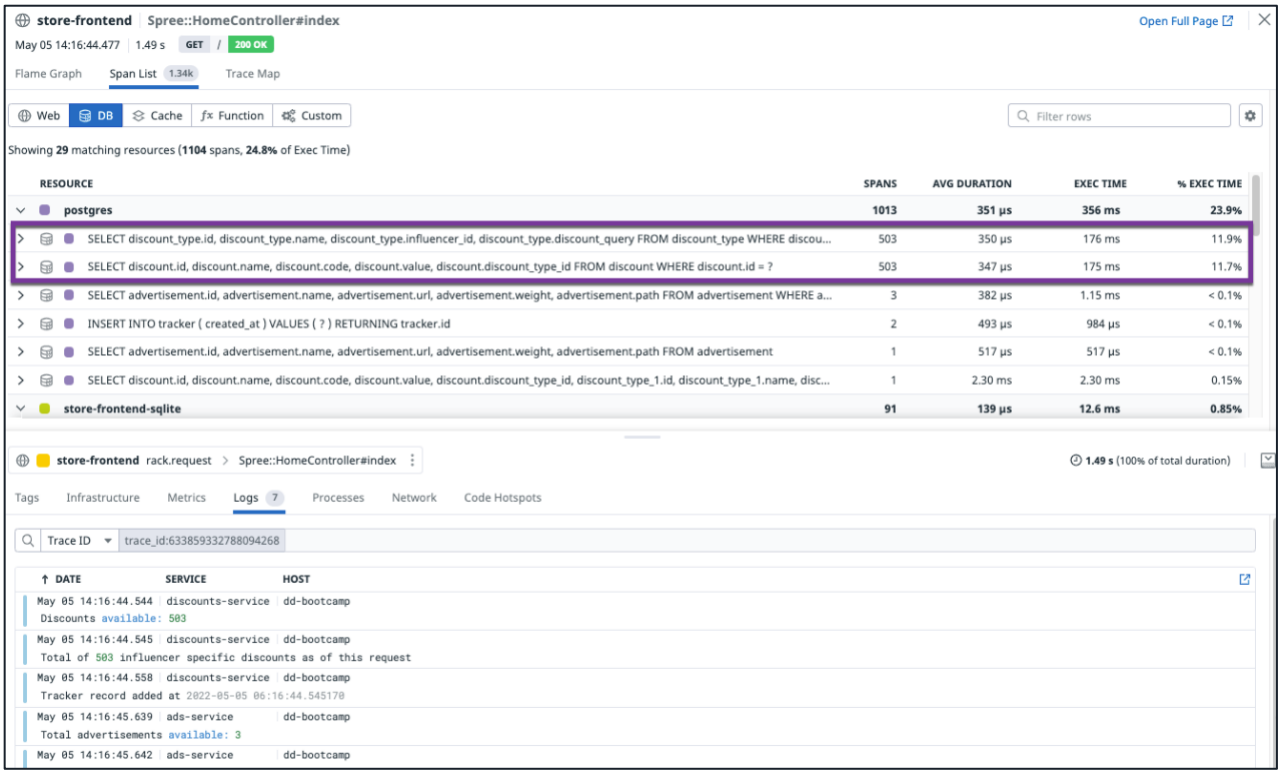
Validating the fix:

Wait for a few minutes for new request to appear.



For validation, follow the similar steps as performed in the **Root Cause Analysis** section.

Validate that the updated query has reduce the number of sql call, hence further improve the latency.



105 Exploring Logs

In Datadog, collected logs are processed using Pipelines. A pipeline takes a filtered subset of incoming logs and applies a list of sequential Processors. Each processor completes a data-structuring action such that a properly built pipeline results in a list of json-formatted, standardized attributes.

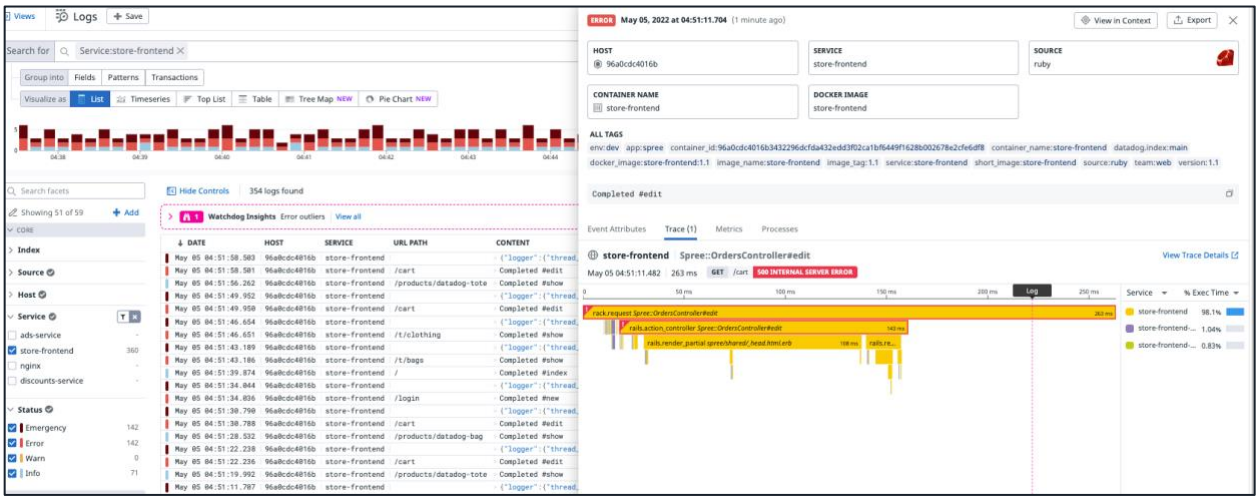
Defining a set of pipelines can help you normalize log data across all the logs you collect in Datadog from different sources throughout your infrastructure and applications. You may prefer to organize pipelines by source or you may prefer to organize pipelines by team and then nest pipelines by sources, or you may prefer to use a combination of both approaches.

105-1 Pipelines and Processors

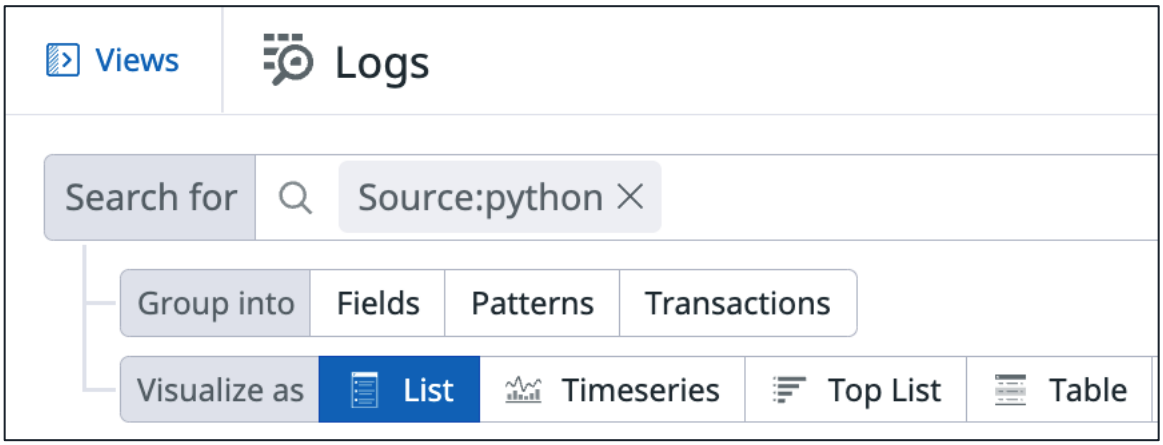
In this exercise, we will take you through the additional steps for correlating logs and trace for python logs

Identifying the problem:

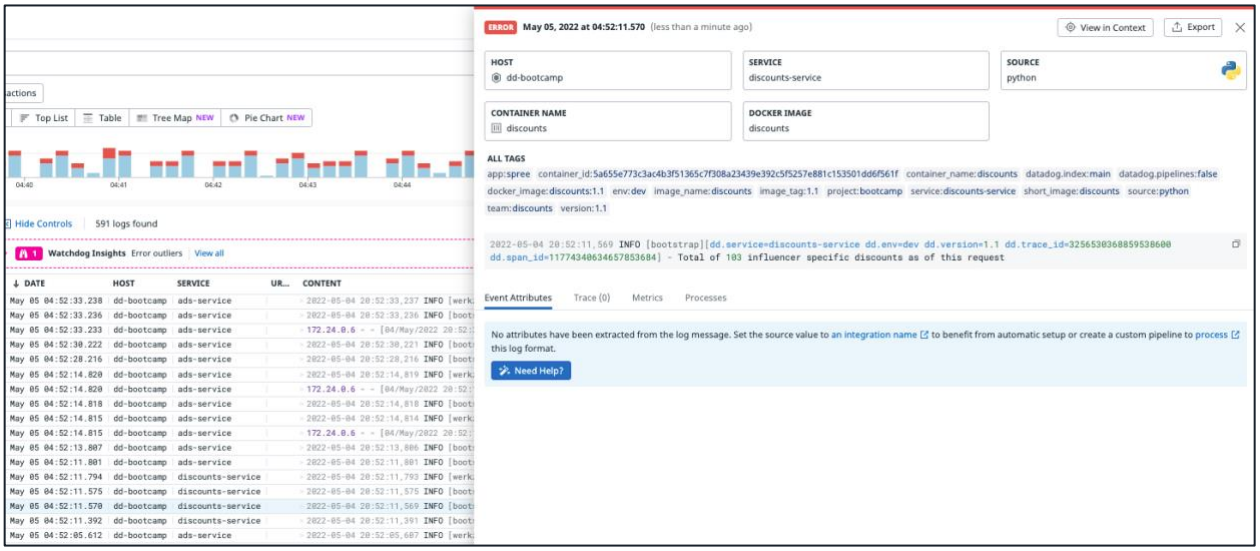
1. Navigate to **Logs->Search** https://app.datadoghq.com/logs , filter by **Service:store-frontend.** Open any log record and you can see that the **Trace** are properly correlated as depict in the screen below

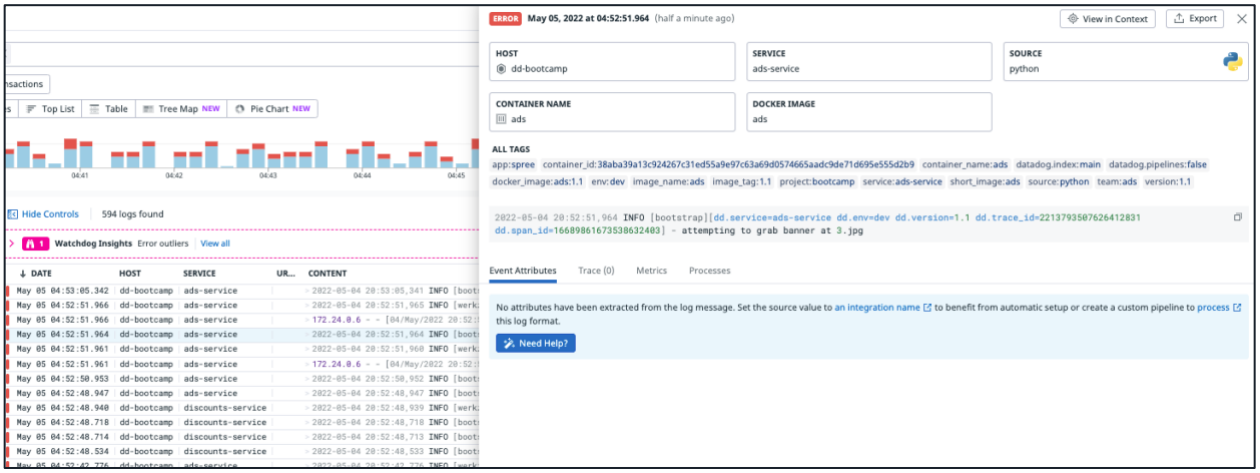


2. Now, **filter by python source**.



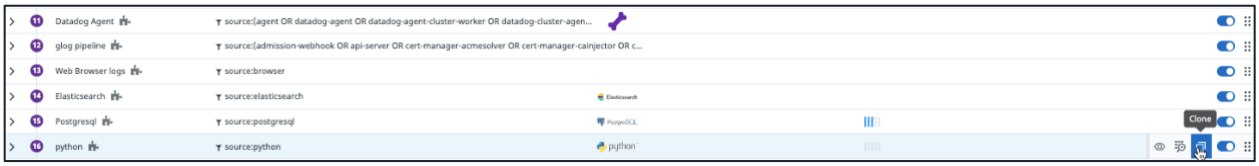
Open any log record and you can see that the **Trace** is not properly correlated despite the trace id and span id are captured for both discounts and advertisement service

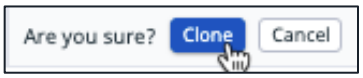




Setting up Log Pipelines:

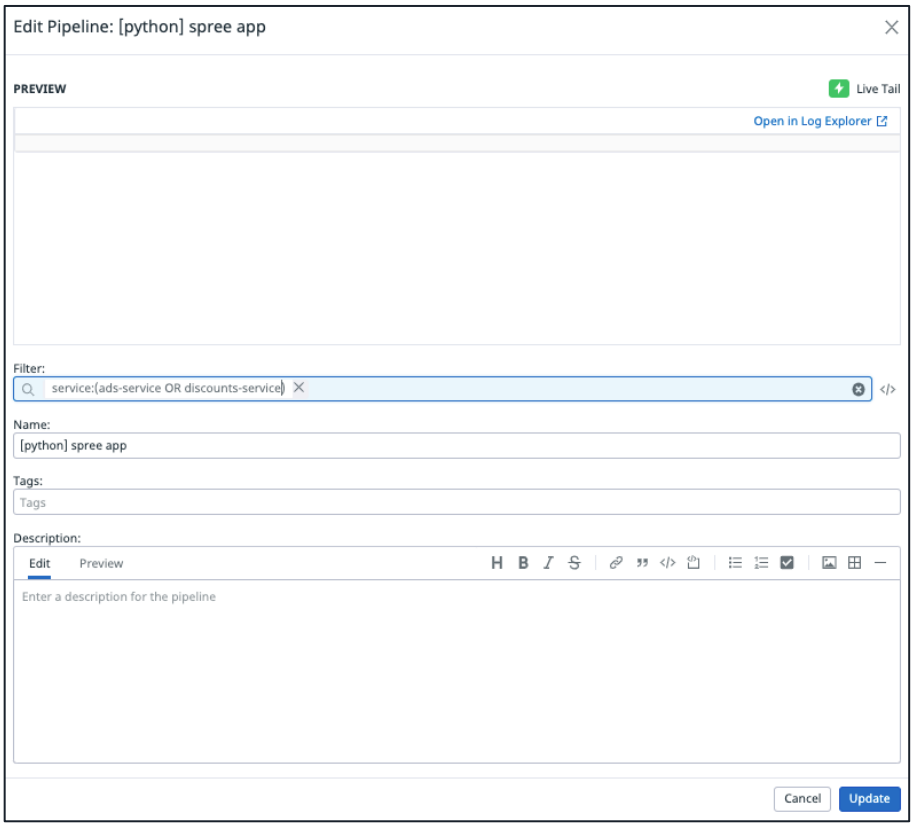
1. Navigate to **Logs->Configurations** https://app.datadoghq.com/logs/pipelines By modifying the processor in the python source, we will be adding additional grok parser rule to extract the trace and span id. Click **clone** on the default python pipeline





Default python will be disabled.

2. Edit the clone ones, and provide a different pipeline name to identify with the spree app. Then, provide filter by using service tag to apply the pipeline process for these logs. Follow along the screenshots below:



3. Finally click on **Update.**

Editing Grok Parser:

Edit existing grok parser, copy a few examples of log record from discounts or advertisement services for testing as depict below. Enter the following additional rule, to extract the **dd.trace_id** and **dd.span_id.** Follow along the screenshot below:

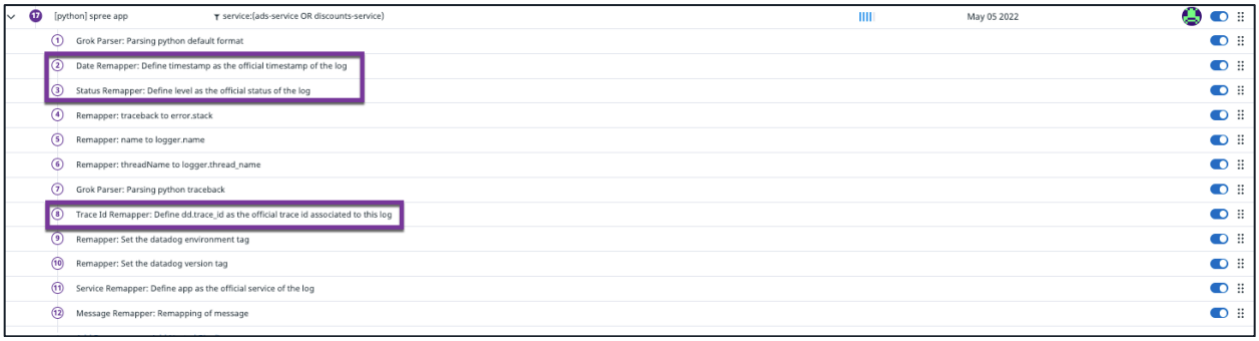
```
other_format %{date("yyyy-MM-dd HH:mm:ss,SSS"):timestamp}
%{word:levelname}\s+\[%{notSpace:process.name}\]\[%{data::keyvalue}dd.trace_id=%{word:
dd.trace_id} dd.span_id=%{word:dd.span_id}\] - %{data:context}
```

Check that the rule matches the log record copied from discounts or advertisement service.

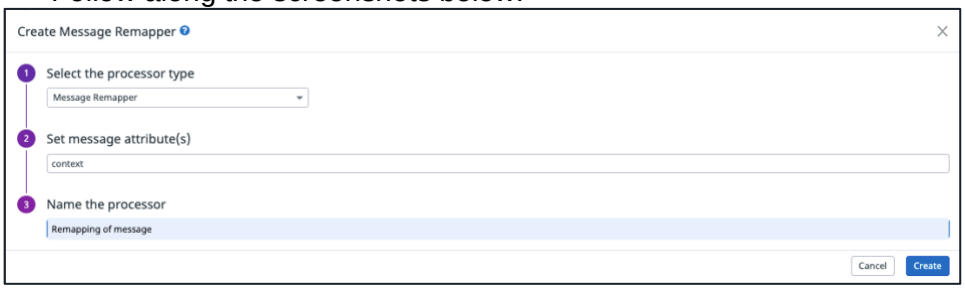
Remappers:

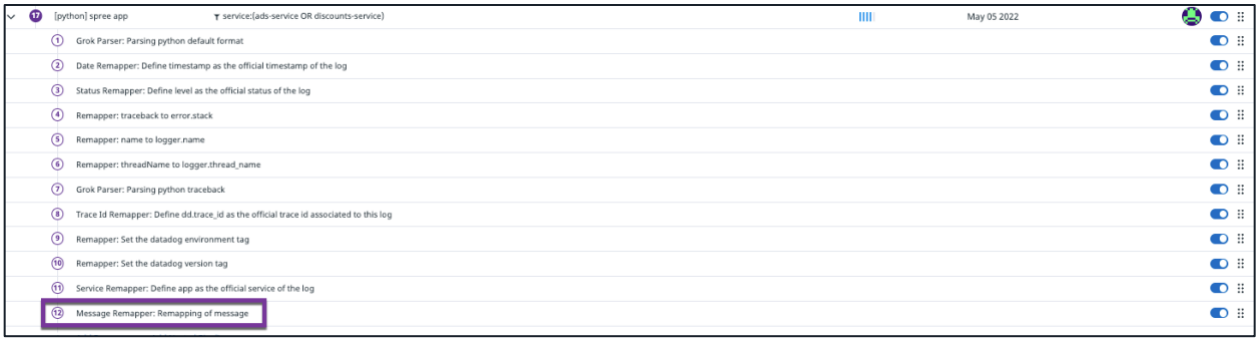
1. The next step is to remap the **trace id** since the processor already exist. Check to make sure that the attribute used matches the ones in parsing rule.
   https://docs.datadoghq.com/logs/log_configuration/processors/?tab=ui#trace-remapper

2. Check if other mapper is required like **date** and **log status** to ensure that they are extracted and mapped accurately.
   https://docs.datadoghq.com/logs/log_configuration/processors/?tab=ui#log-date-remapper
   https://docs.datadoghq.com/logs/log_configuration/processors/?tab=ui#log-status-remapper
   Follow along the screenshot below:



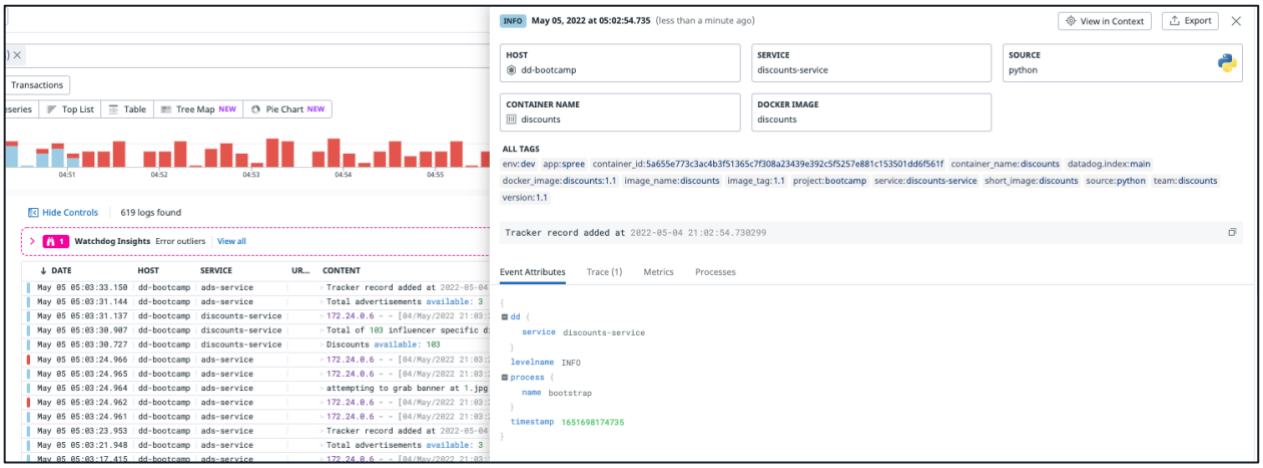
3. Create **message remapper** so that it provides proper context and also you will be able to search by the log message.
   https://docs.datadoghq.com/logs/log_configuration/processors/?tab=ui#log-message-remapper
   Follow along the screenshots below:

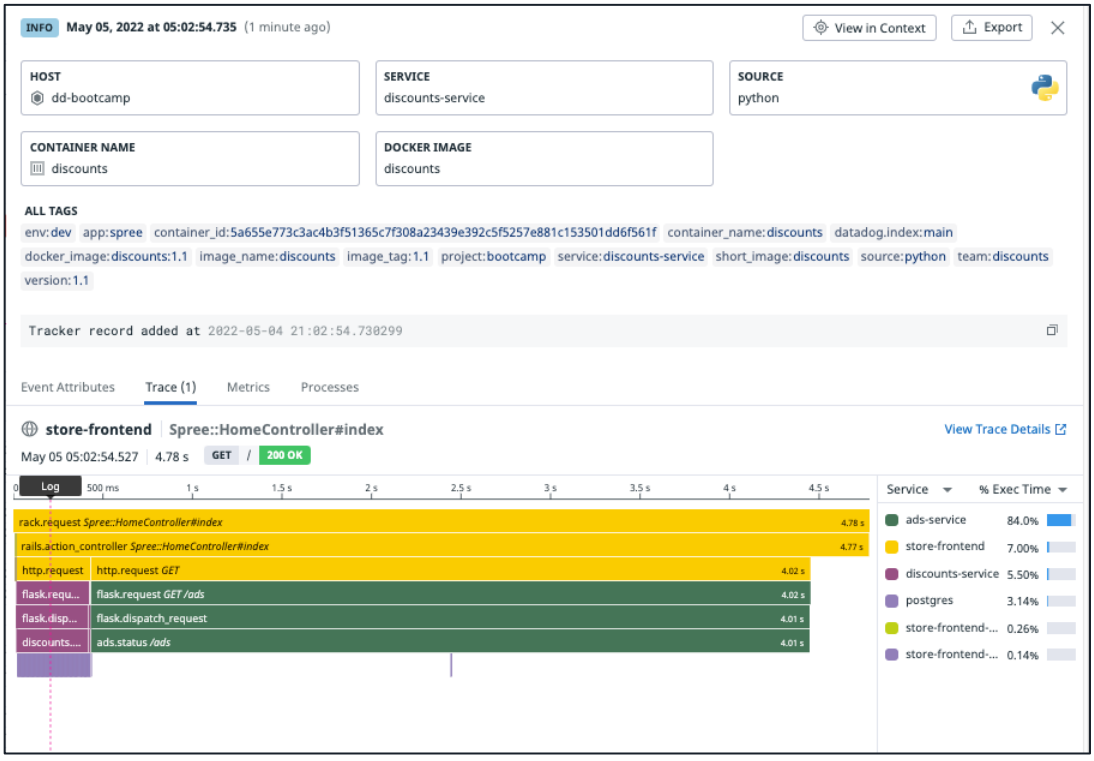


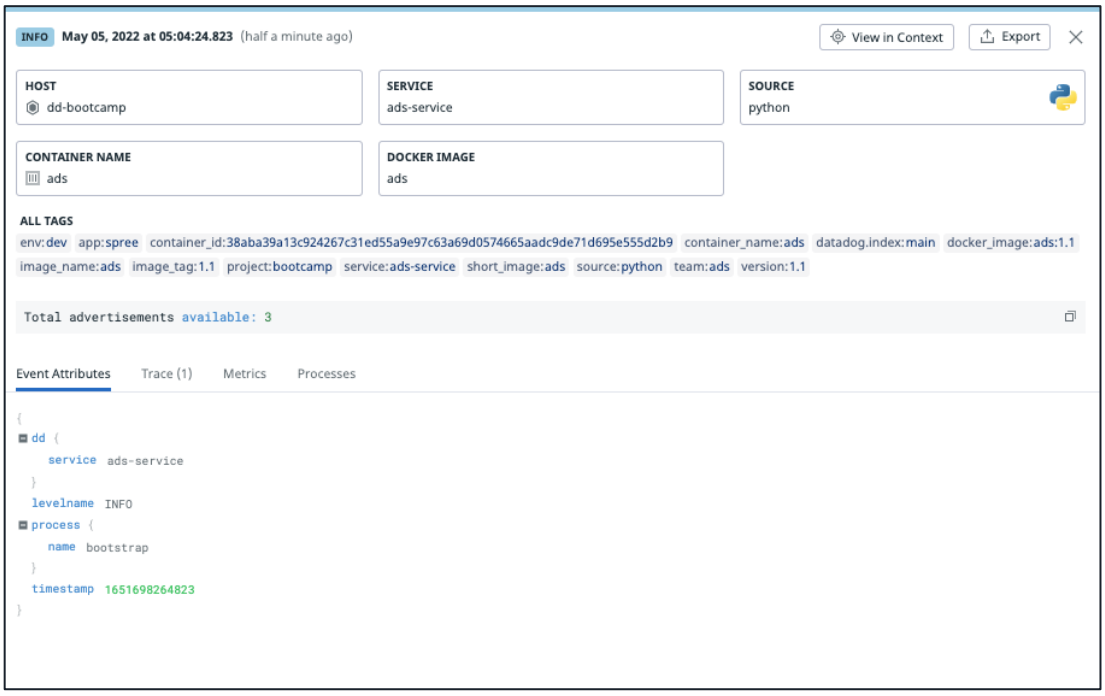


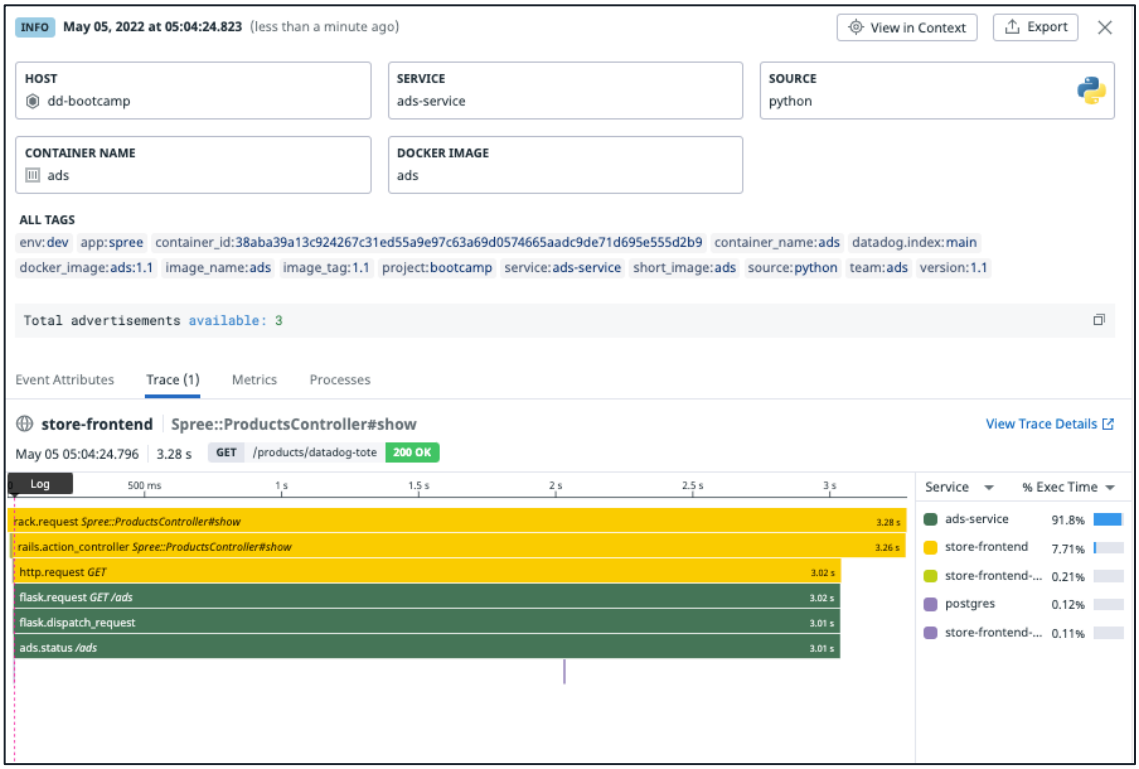
Validating the Logs and Traces Co-relation:

Navigate to **Logs->Search** https://app.datadoghq.com/logs Validate the new incoming log records that the changes are applied to the logs from discounts service and ads service. Follow along the screenshots:






INFO May 05, 2022 at 05:04:24.823 (half a minute ago)
View in Context
Export
HOST
dd-bootcamp
SERVICE
ads-service
SOURCE
python
CONTAINER NAME
ads
DOCKER IMAGE
ads
ALL TAGS
env:dev app:spree container_id:38aba39a13c924267c31ed55a9e97c63a69d0574665aadc9de71d695e555d2b9 container_name:ads datadog.index:main docker_image:ads:1.1 image_name:ads image_tag:1.1 project:bootcamp service:ads-service short_image:ads source:python team:ads version:1.1
Total advertisements available: 3
Event Attributes
Trace (1)
Metrics
Processes
{
dd {
service ads-service
}
levelname INFO
process {
name bootstrap
}
timestamp 1651698264823
}


INFO May 05, 2022 at 05:04:24.823 (less than a minute ago)
View in Context
Export
HOST
dd-bootcamp
SERVICE
ads-service
SOURCE
python
CONTAINER NAME
ads
DOCKER IMAGE
ads
ALL TAGS
env:dev app:spree container_id:38aba39a13c924267c31ed55a9e97c63a69d0574665aadc9de71d695e555d2b9 container_name:ads datadog.index:main docker_image:ads:1.1 image_name:ads image_tag:1.1 project:bootcamp service:ads-service short_image:ads source:python team:ads version:1.1
Total advertisements available: 3
Event Attributes
Trace (1)
Metrics
Processes
store-frontend
Spree::ProductsController#show
View Trace Details
May 05 05:04:24.796
3.28 s
GET
/products/datadog-tote
200 OK
Log
500 ms
1 s
1.5 s
2 s
2.5 s
3 s
rack.request Spree::ProductsController#show 3.28 s
rails.action_controller Spree::ProductsController#show 3.26 s
http.request GET 3.02 s
flask.request GET /ads 3.02 s
flask.dispatch_request 3.01 s
ads.status /ads 3.01 s
Service
% Exec Time
ads-service 91.8%
store-frontend 7.71%
store-frontend-... 0.21%
postgres 0.12%
store-frontend-... 0.11%

105-2 Logs Search

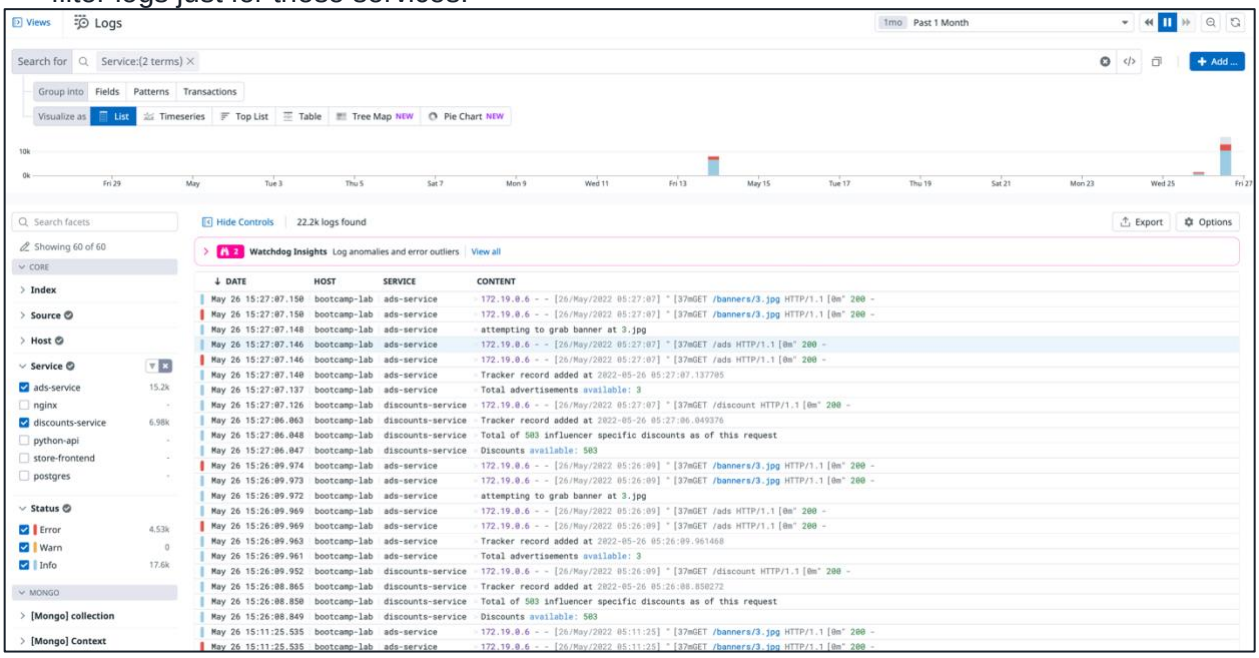
The Search view in Logs is a log list that displays indexed logs matching a search context (i.e., a search query for a selected time range).

1. Navigate to **Logs->Search**

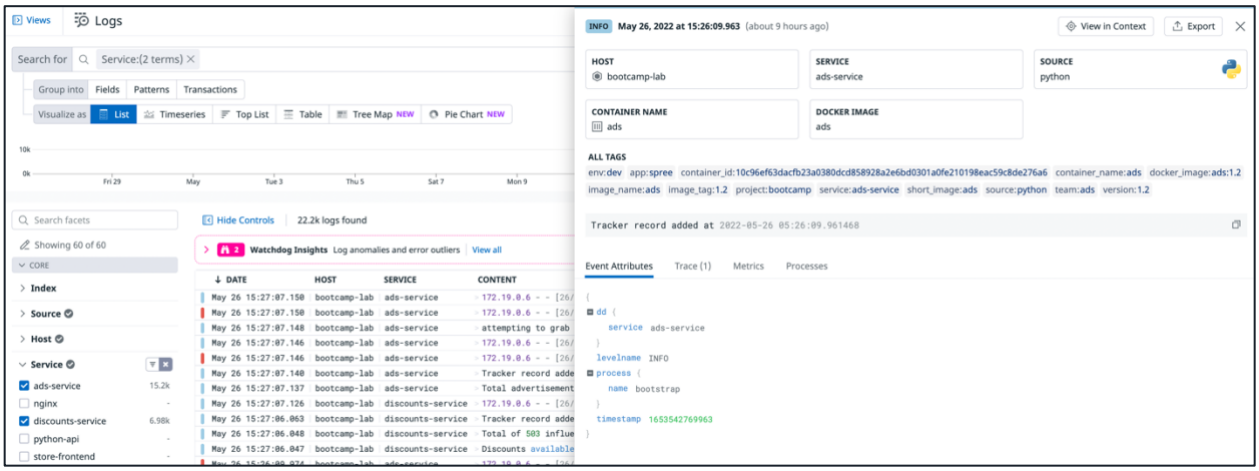
2. You can use the search field and the Facets list, which is located to the left of the logs list, to create a search query to filter the log list.

   A search query can include assigned tags like **env** and **service**, attributes extracted from the logs like @http.status_code, and text strings from log messages. Search queries built in the search field require proper search syntax.

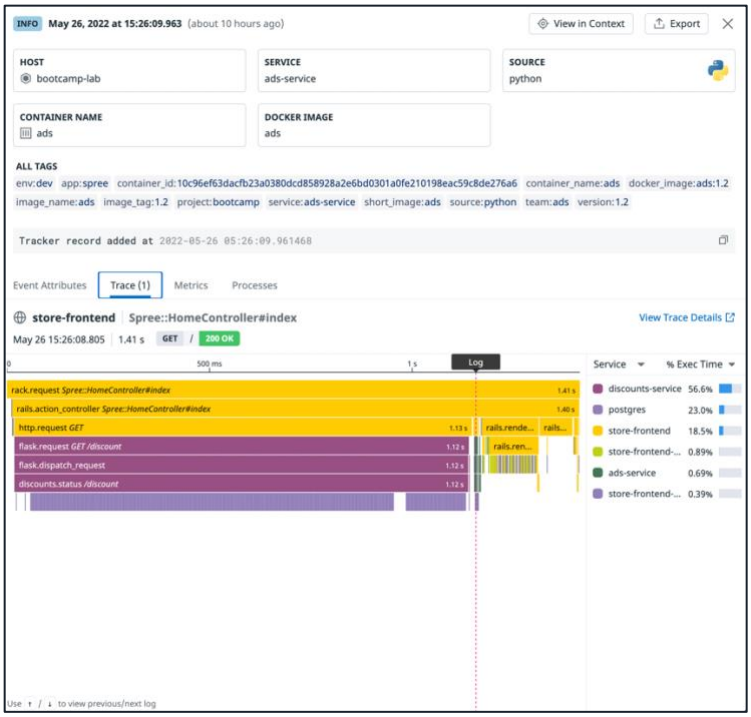
   In the Facets list on the left, select **ads-service and discounts-service** under **Service** to filter logs just for these services.



3. Find a log which says as “Tracker record…” and click it. The Log Side Panel opens with the log details including assigned tags, the log message, any extracted attributes, related traces, and related infrastructure metrics.



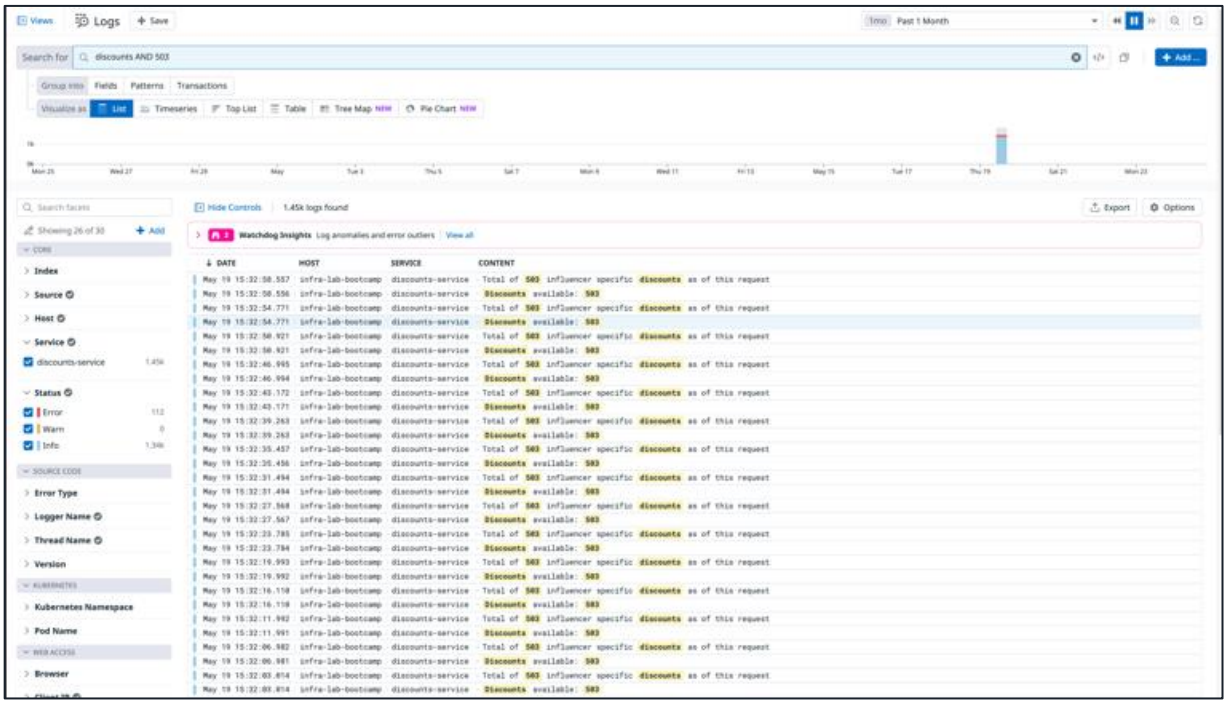
4. In this detailed log panel, click the **Traces** tab to view the associated trace.



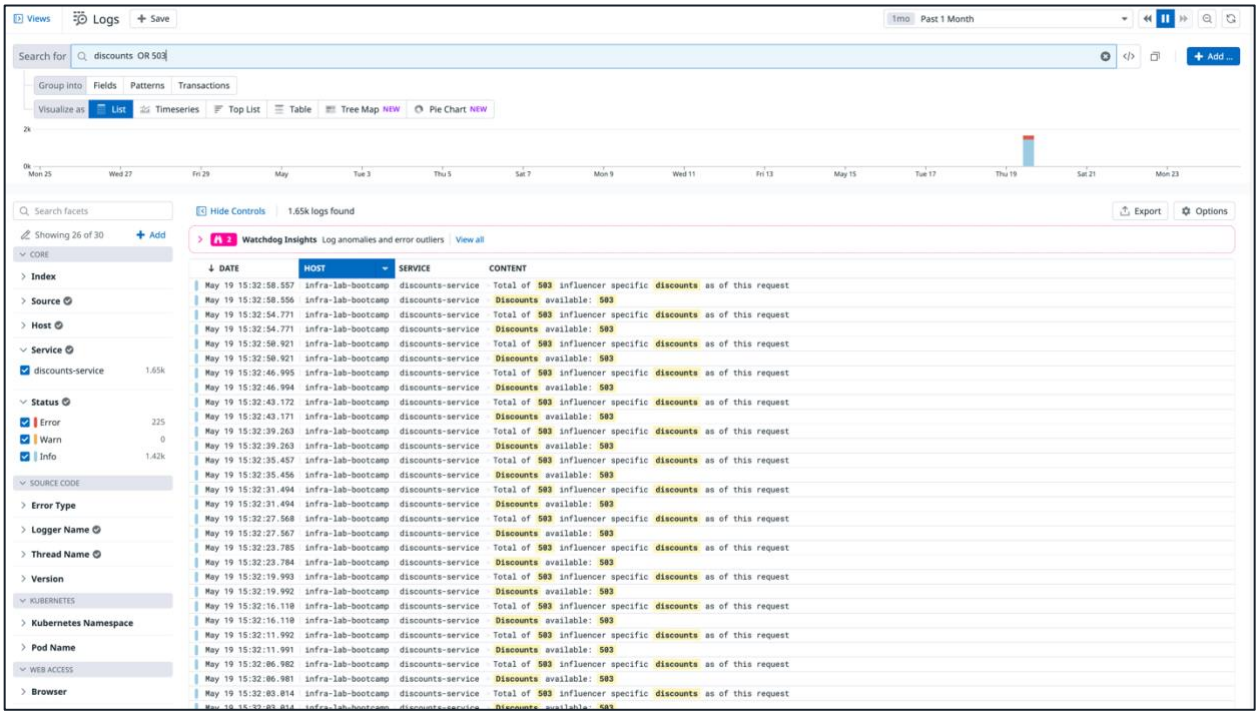
5. Click the **Metrics** tab to view the associated infrastructure metric.

To combine multiple terms into a complex query, you can use any of the following Boolean operators, examples below:

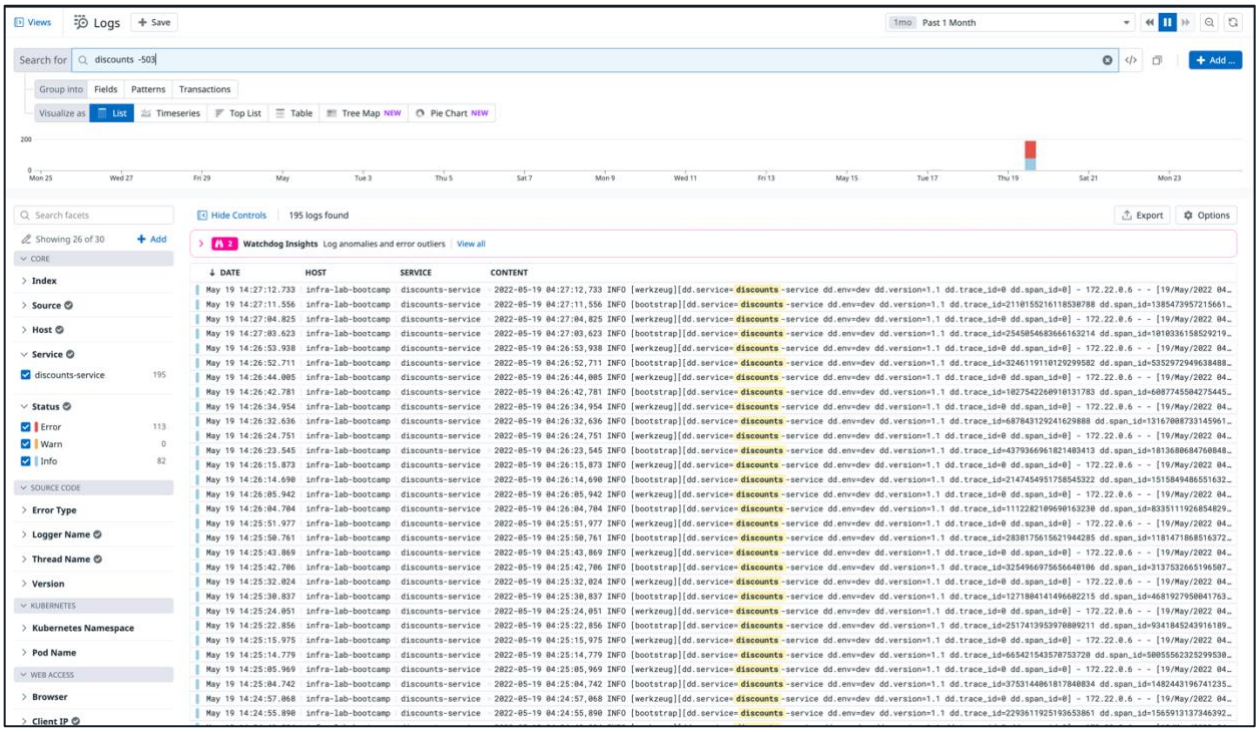
AND



OR


Views
Logs
Save
1mo Past 1 Month
Search for discounts OR 503
Add ...
Group into Fields Patterns Transactions
Visualize as List Timeseries Top List Table Tree Map NEW Pie Chart NEW
Search facets
Hide Controls
1.65k logs found
Export
Options
Showing 26 of 30
Add
Watchdog Insights Log anomalies and error outliers View all
DATE HOST SERVICE CONTENT
Index
Source
Host
Service
discounts-service 1.65k
Status
Error 225
Warn 0
Info 1.42k
Error Type
Logger Name
Thread Name
Version
Kubernetes Namespace
Pod Name
Browser

NOT


Views
Logs
Save
1mo Past 1 Month
Search for discounts -503
Add ...
Group into Fields Patterns Transactions
Visualize as List Timeseries Top List Table Tree Map NEW Pie Chart NEW
Search facets
Hide Controls
195 logs found
Export
Options
Showing 26 of 30
Add
Watchdog Insights Log anomalies and error outliers View all
DATE HOST SERVICE CONTENT
Index
Source
Host
Service
discounts-service 195
Status
Error 113
Warn 0
Info 82
Error Type
Logger Name
Thread Name
Version
Kubernetes Namespace
Pod Name
Browser
Client IP

105-3 Logs Patterns

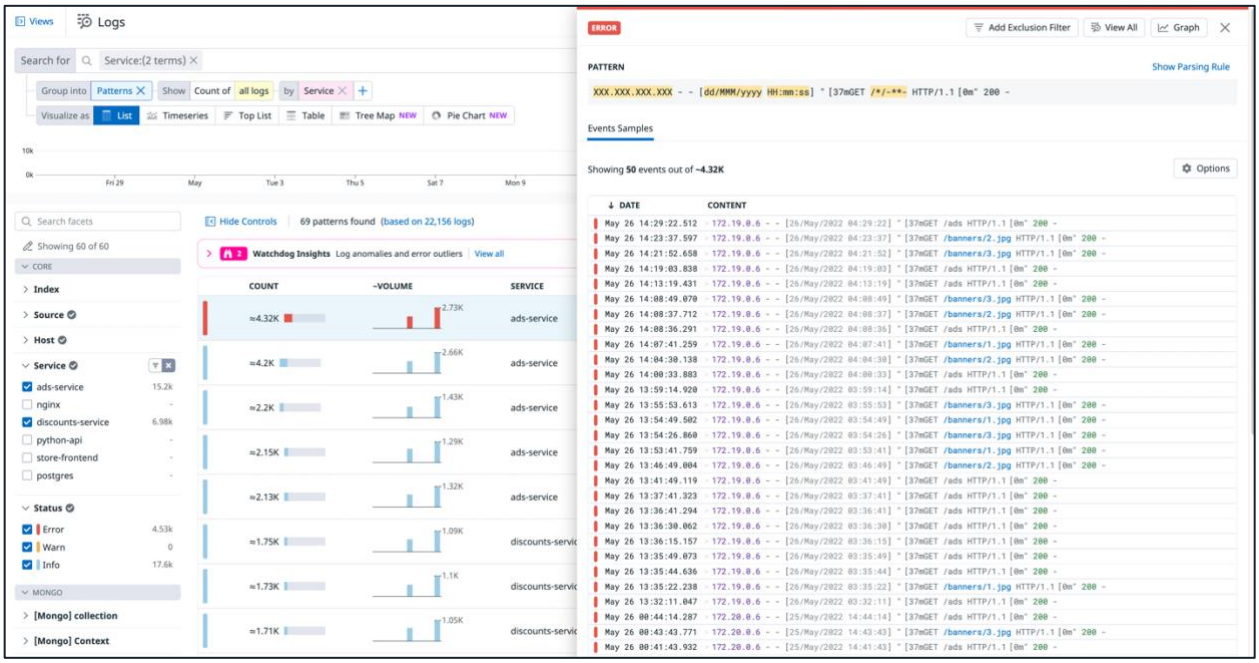
Organizing large volumes of logs from different sources can be cumbersome. However, logs from a source usually have specific patterns. The Patterns aggregation can be surfaced automatically in the Log Explorer and are listed by number of logs with a service name, log status, and log message that matches a certain pattern.

You can filter the list using search syntax and facets to focus on patterns you're interested in. You can also click a pattern from the list to view log samples matching the pattern and even the pattern's parsing rule.

With pattern aggregation, logs that have a message with similar structures, belong to the same service, and have the same status are grouped together. The patterns view is helpful for detecting and filtering noisy error patterns that could cause you to miss other issues.

1. In **Logs**, filter the list to the service:advertisements-service and service:discounts-service.
2. Select Patterns for **Group in** below the search field. Observe the differences in the patterns that are listed.

3. Click one of the patterns to open the details side panel. The pattern and a list of "Events Samples" are displayed.



In the upper-right above the Pattern, click **Show Parsing Rule**. This allows you to see how the logs were parsed and if there are any optimizations you'd like to make by building a custom pipeline for this service/source.

Close the side panel.

Note: To learn more about creating custom pipelines, check out the Going Deeper with Logs: Processing course in the Learning Center.

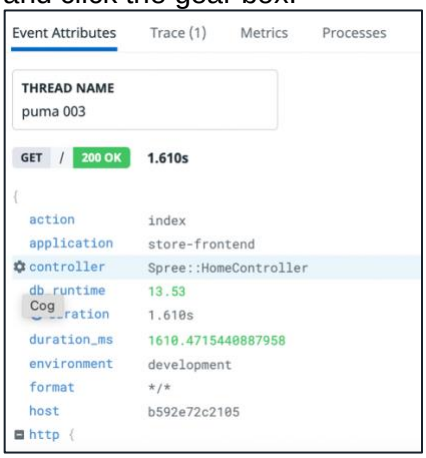
4. Click the "X" next to **Patterns** to return to the log list

105-4 Facets and Measures

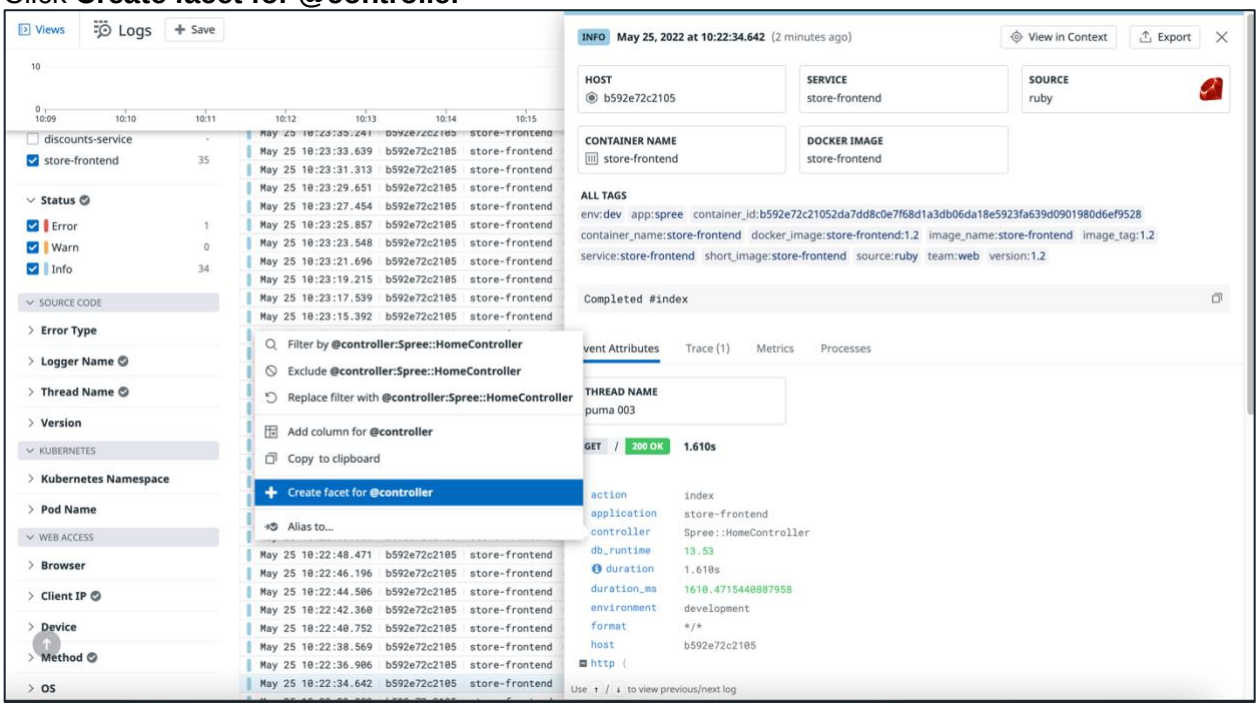
Creating a Facet:

Navigate to **Logs->Search** for **service:store-frontend**

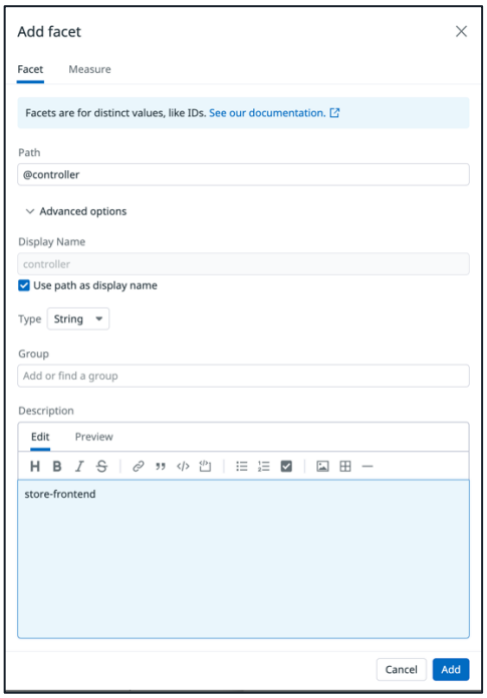
Click any log line, in the pop up from the Event Attributes sub-tab, hover beside the **controller** and click the gear box.



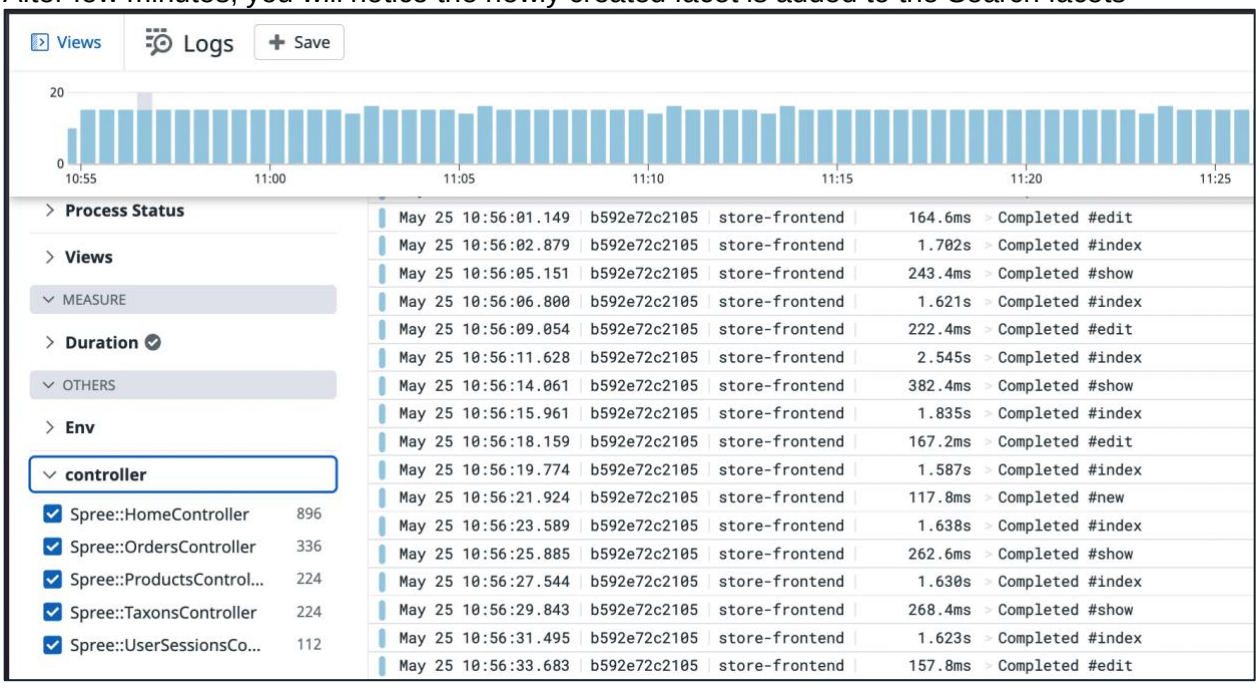
Click **Create facet for @controller**



Make below selections and inputs. Click **Add**



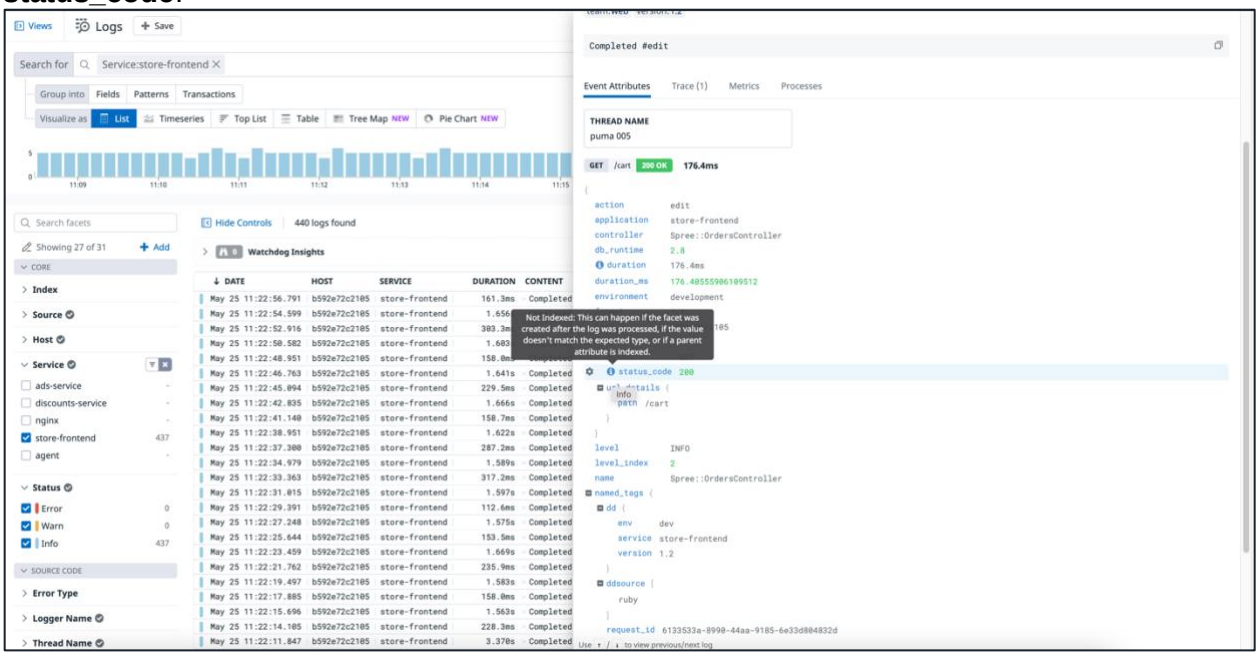
Validation:
After few minutes, you will notice the newly created facet is added to the Search facets



Fixing problem with facet:

Navigate to **Logs->Search** for **service:store-frontend**

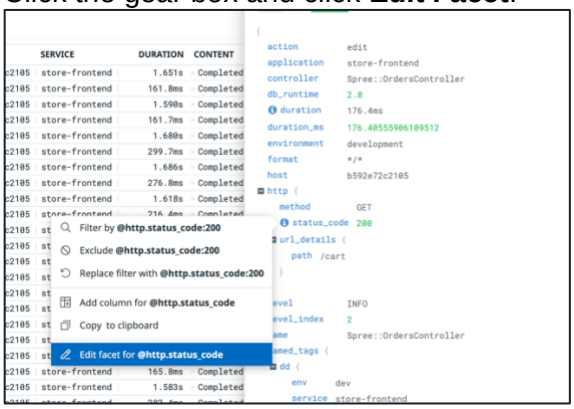
Click any log line, in the pop up from the Event Attributes sub-tab, notice the message against **status_code**:



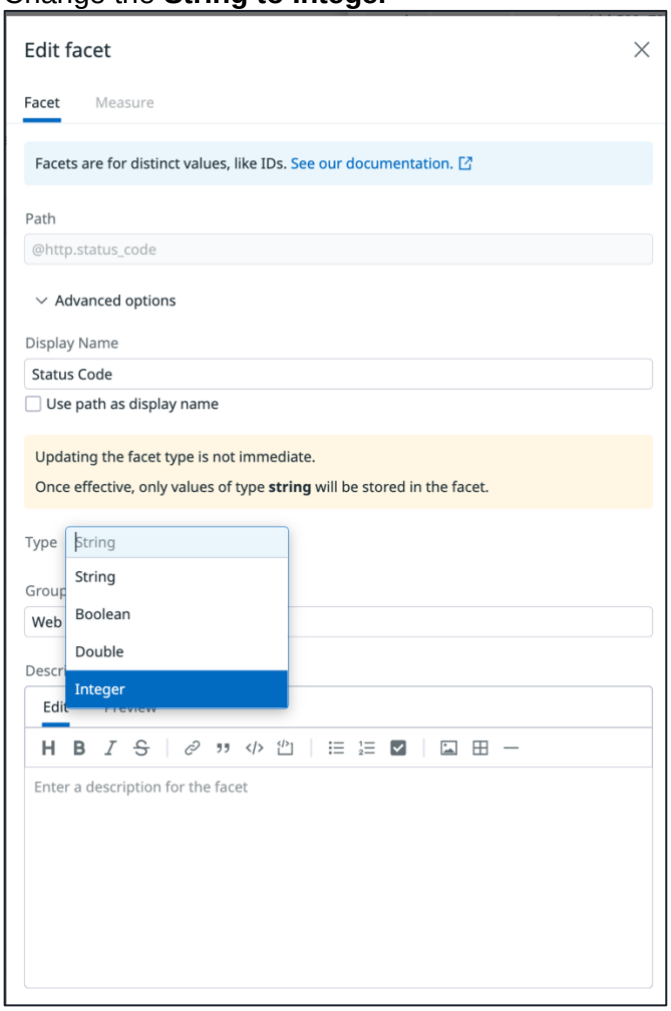
The issue is that the facet is of type string, while the status code is actually an integer(the green colour of the value indicates this).

Let’s edit the facet to fix the problem. Follow along the screenshots

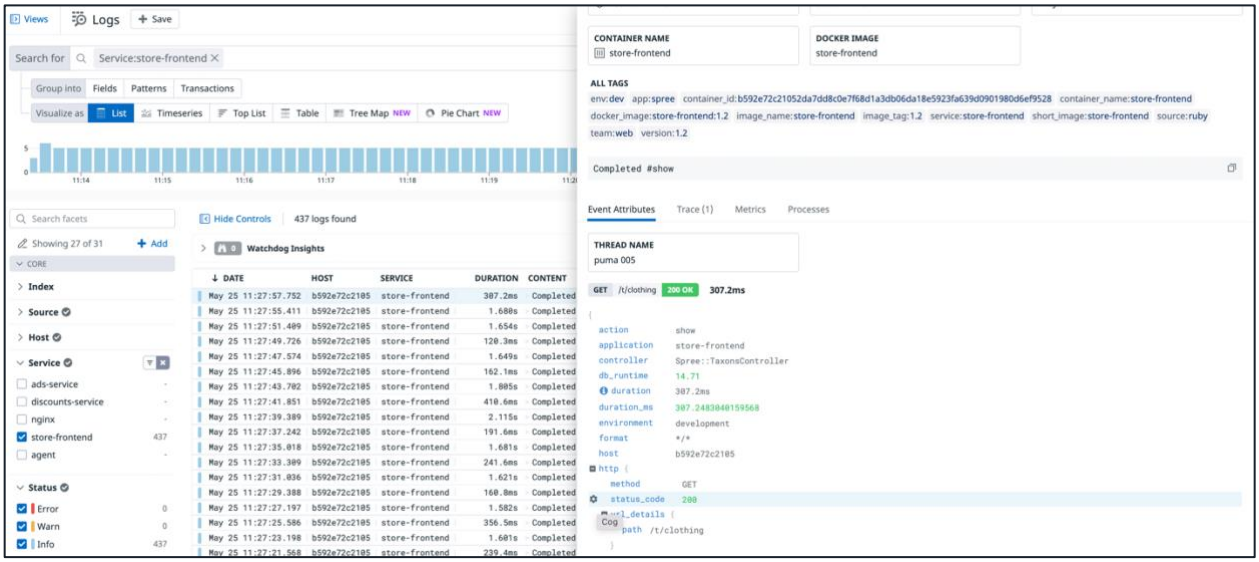
Click the gear box and click **Edit Facet**:



Change the **String to Integer**



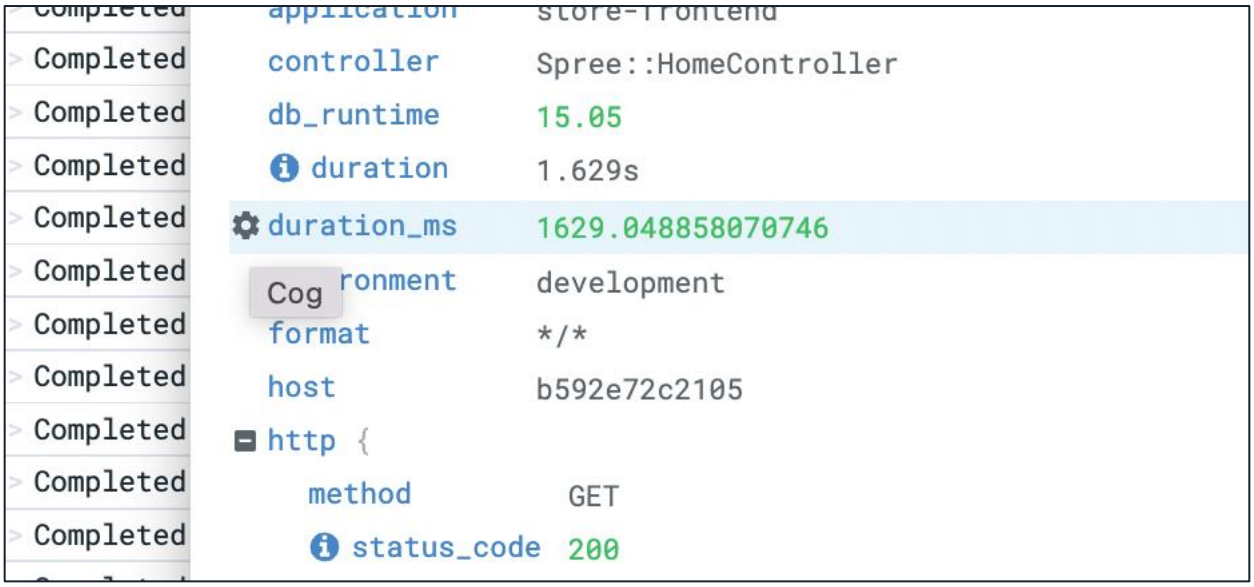
Wait for some time and in the new logs you will see the issue is fixed:



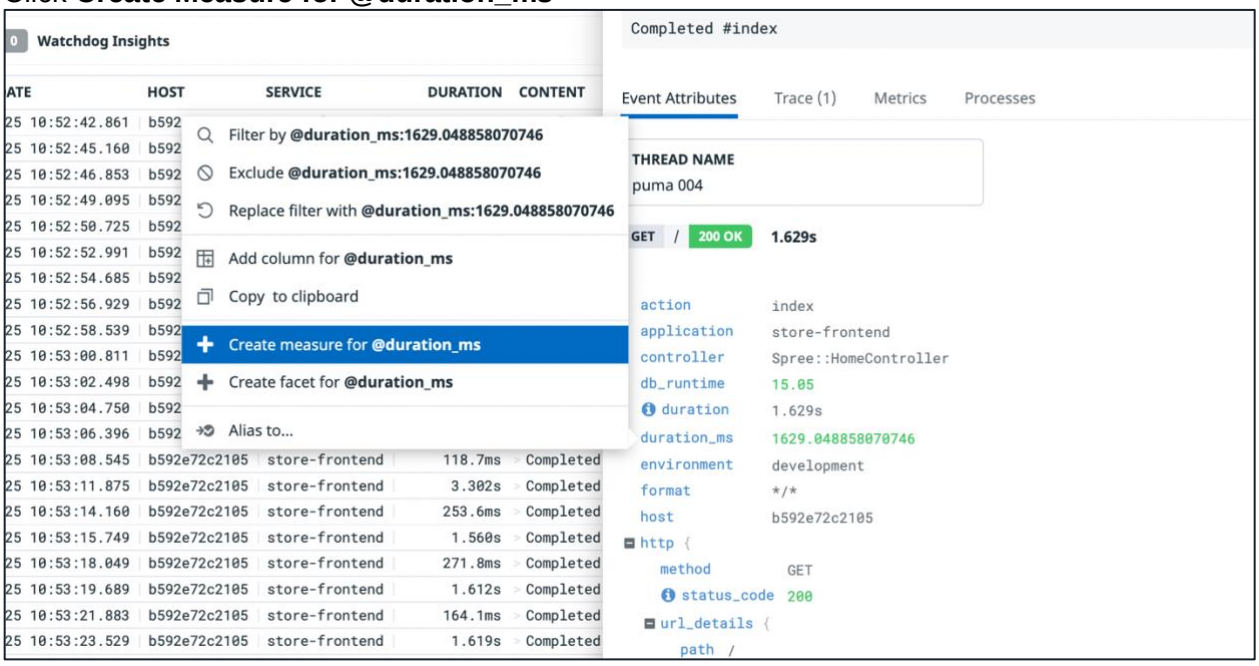
Creating a Measure:

Navigate to **Logs->Search for service:store-frontend**

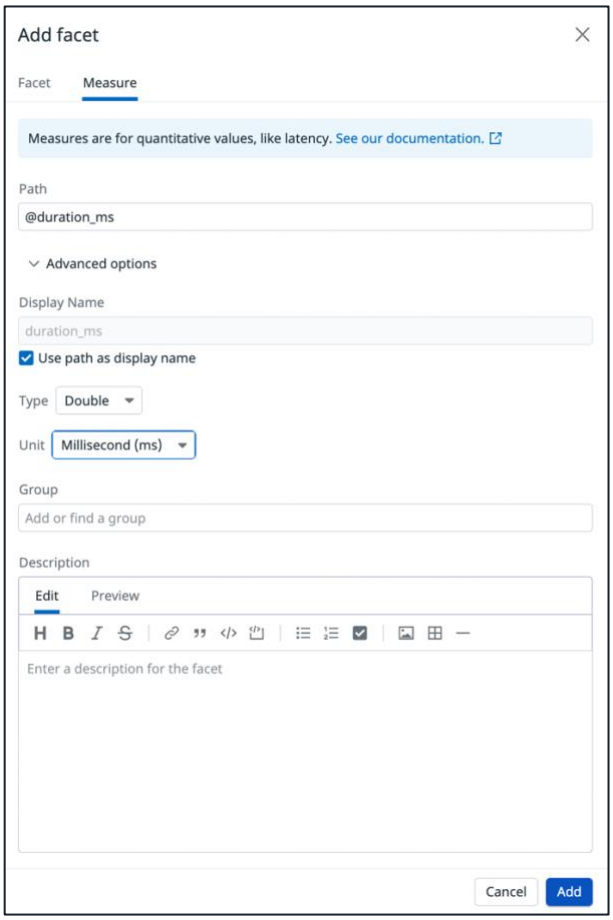
Click any log line, In the pop up from the Event Attributes sub-tab, hover beside the **duration_ms** and click the gear box.



Click **Create Measure for @duration_ms**

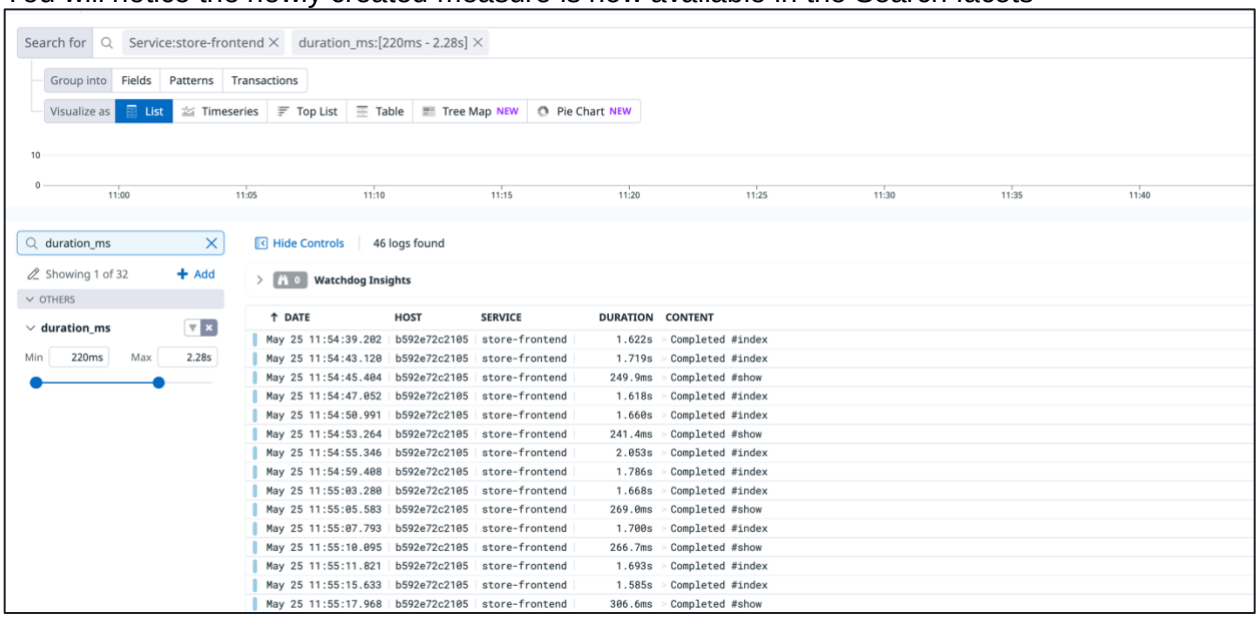


Make below selections and inputs. Click **Add**



Validation:

You will notice the newly created measure is now available in the Search facets



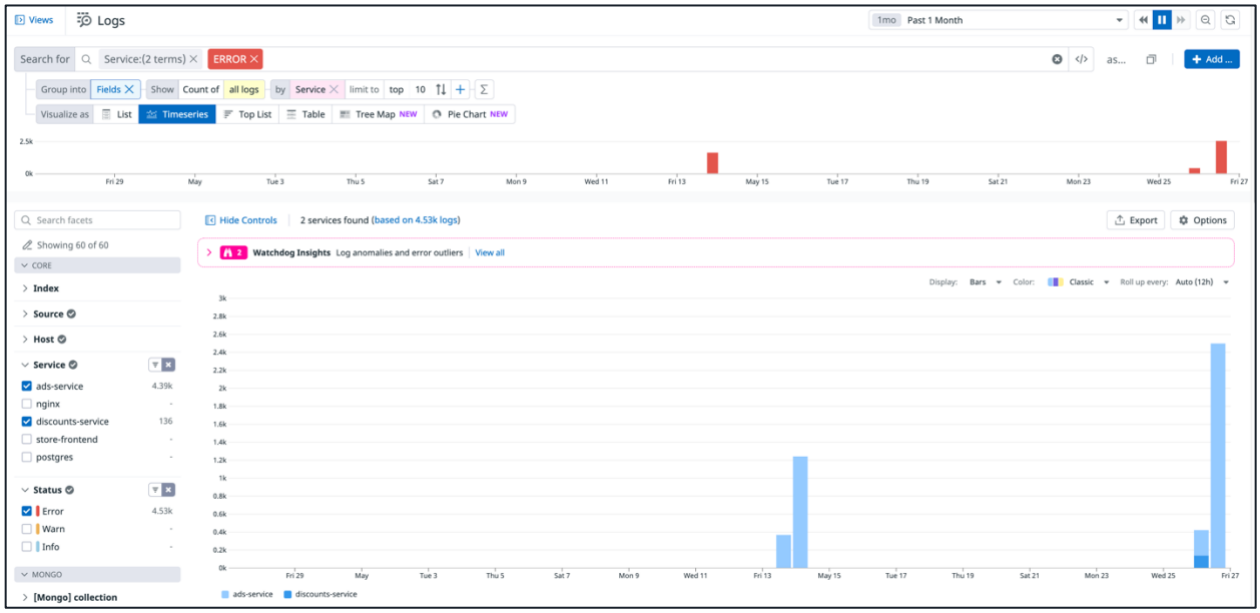
105-5 Log Analytics

With the Fields aggregation option, which is located right underneath the log query filter at the top of the page, all logs matching the query filter are aggregated into groups based on the value of a log facet. For these groups, you can extract counts of logs per group, unique counts of coded values for a facet per group, and statistical operations on numerical values of a facet per group.

1. In **Logs**, filter the list to the **service:ads-service, service:discounts-service, and status:error.**
2. Select Fields for **Group into** below the search field. A graph visualization of the filtered logs will replace the log list. Group the fields by service so that it reads **Group into** Fields and **Show Count of all logs by Service.**

   Above the graph, you'll see that **Timeseries** is selected, which gives you a timeline of total errors broken out by service. Select **Toplist** next to **Timeseries**, then select **Table** to view the different visualizations. In this case, you'll see the total number of errors per service in a set period of time, whereas the **Timeseries** view gives you a more granular breakdown of when the errors occurred during that time period.

   Click **Timeseries** again.



3. Click **Export** above the graph. You can export the visualization areas of the product, such as a Logs Monitor, to a dashboard, and to generate a log-based metric (Generate Metrics). Click **Export** again to close the menu.

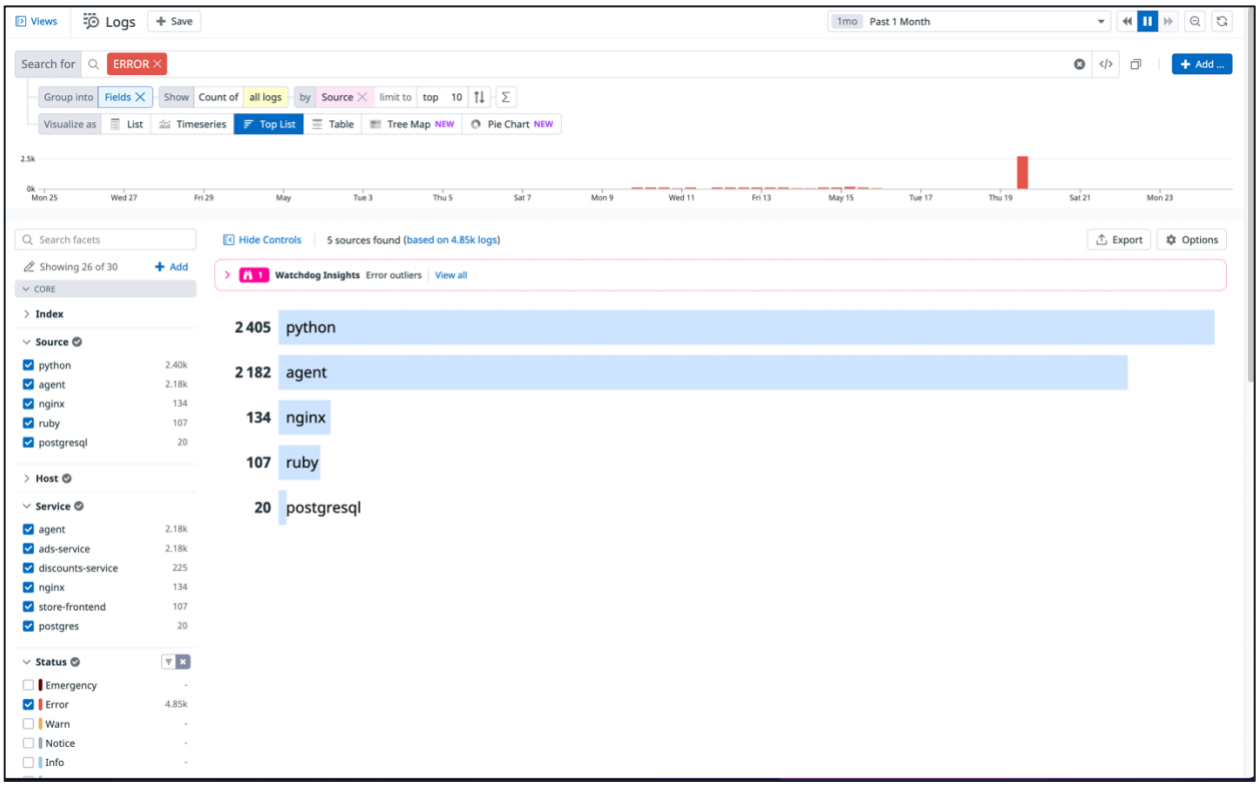
4. Click **Save** above the search field to save this view.

   Enter **Ads_Discounts_Errors_Timeseries** as the name of the **New View** and click **Save**. The new view will appear in the list.

Spend a moment or two exploring the options you just learned, then return to this view by selecting it from the list. When you do, you'll notice the URL parameters and search criteria change to reflect the new view, meaning you can share it with others.

Below are some more examples for you to explore Logs Analytics.

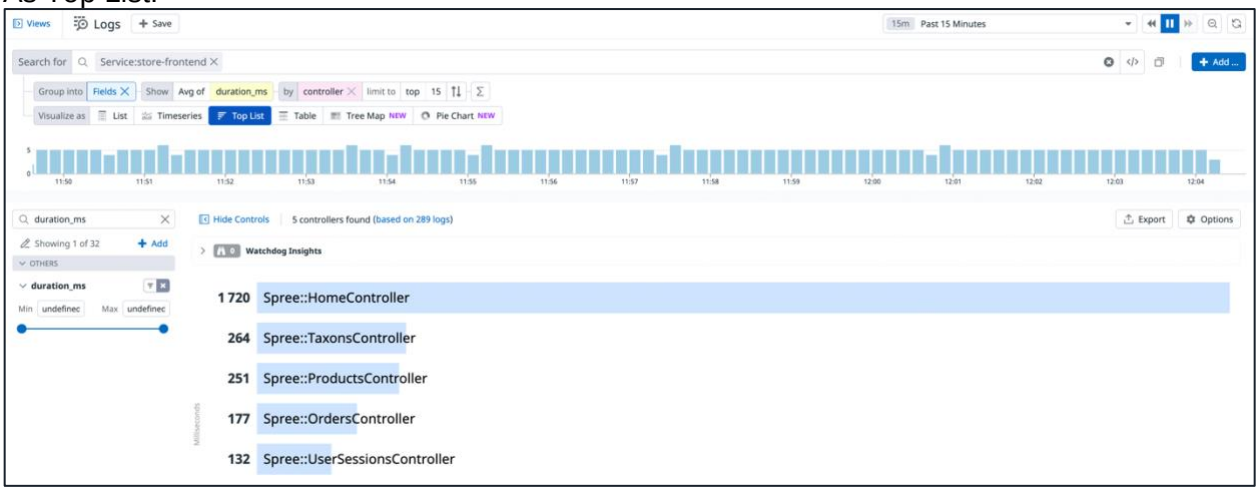
Explore Logs Source with Errors:



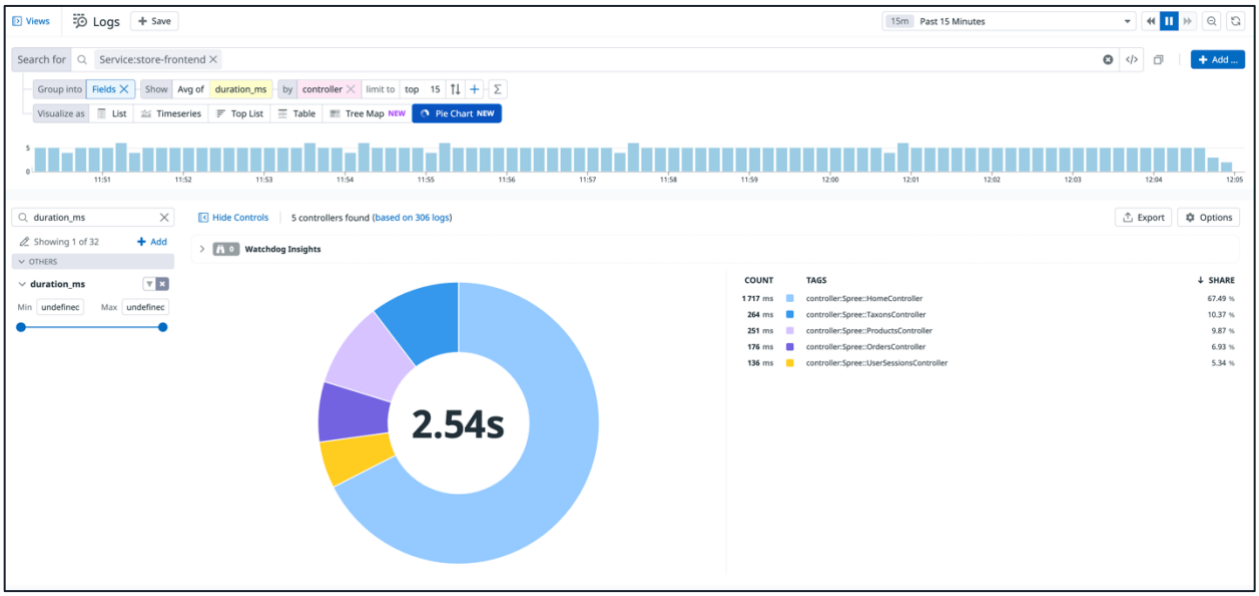
Using the Facets and Measure created in Lab 105-3:

As Timeseries:

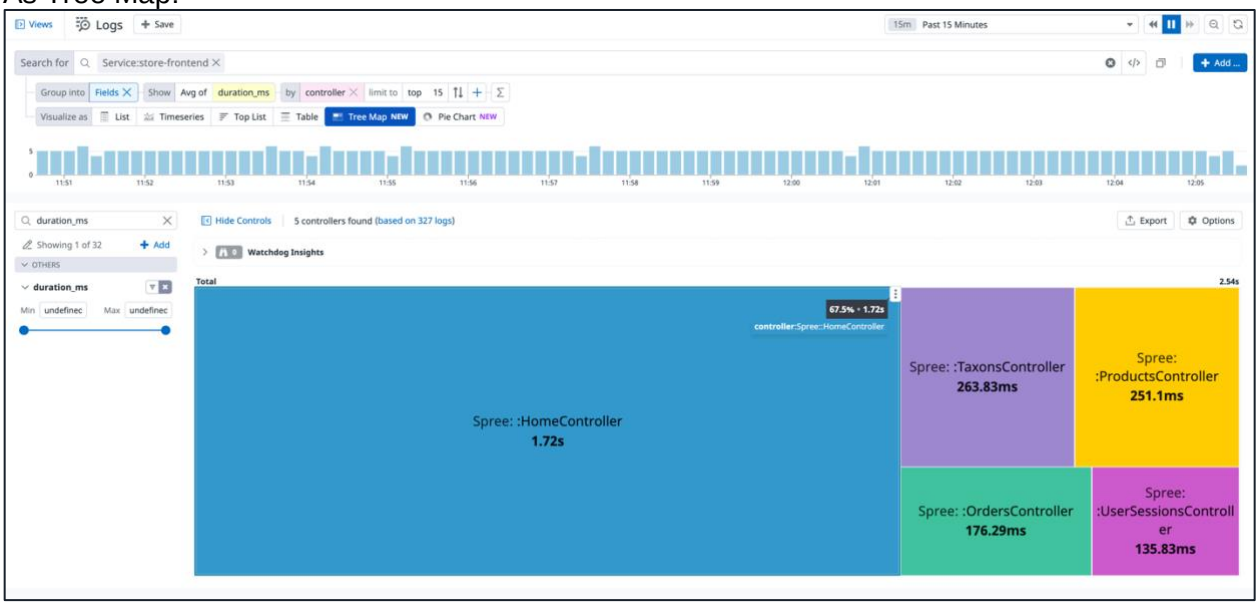
As Top List:



As Pie Chart:



As Tree Map:



106 Browser RUM Lab

Datadog Real User Monitoring (RUM) enables you to visualize and analyze the real-time performance and user journeys of your application's individual users.

In this lab exercise, we will setup Datadog Real User Monitoring (RUM) to enable the team to visualize and analyse the real-time performance and user journey of the spree ecommerce application.

106-1 Prepare the lab

Instructions:

First stop the application.

```
$ cd ~/docker/ecommerce-workshop/deploy
$ docker-compose -f docker-compose-instr.yml down
```

106-2 Setup Browser RUM monitoring

Setting up RUM Browser monitoring, comprising the following:

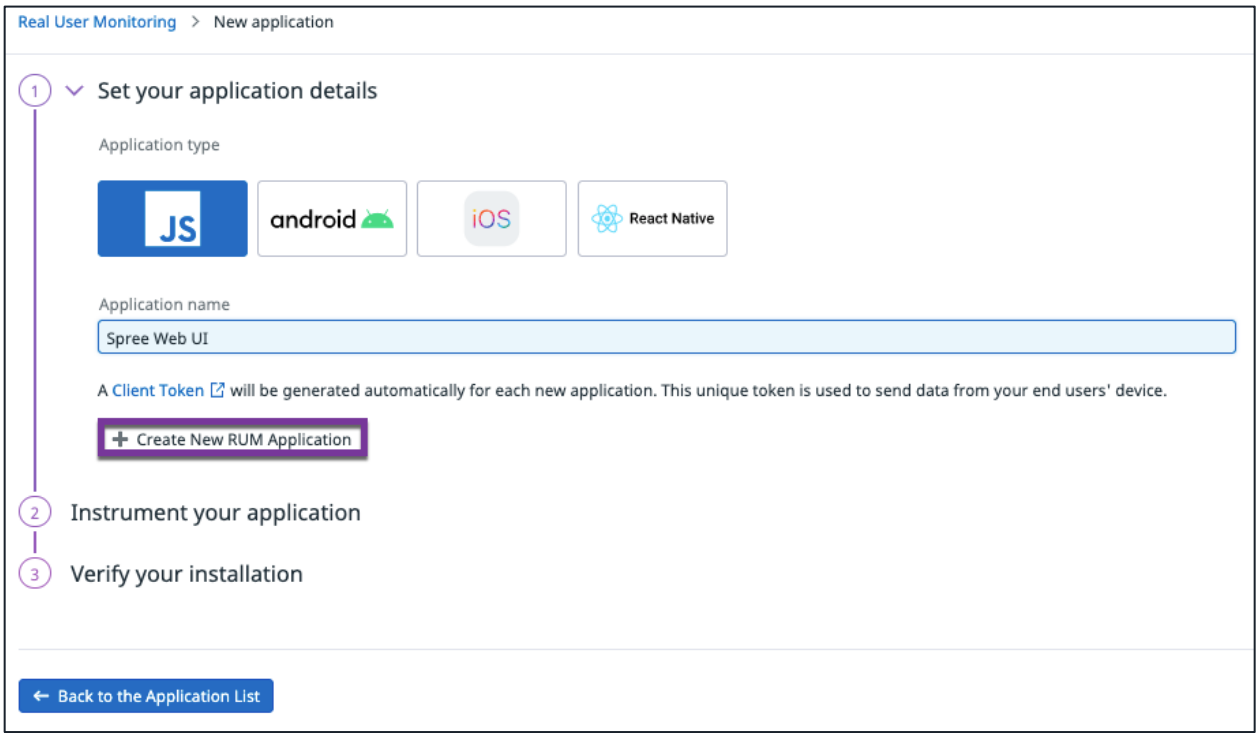
- Create new Application via Datadog Real User Monitoring page
- Setup the Datadog RUM SDK
- Connect RUM and Traces
- Deploy the changes to your application

Refer to the link for more details: https://docs.datadoghq.com/real_user_monitoring/browser/

Create New Application via Datadog Real User Monitoring page:

The detailed steps are outlined here: https://app.datadoghq.com/rum/application/create

1. First, navigate to **UX Monitoring->RUM Applications,** hit the **+New Application,** select JS (browser monitoring), input Application name as **Spree Web UI** and click Create New RUM Application.

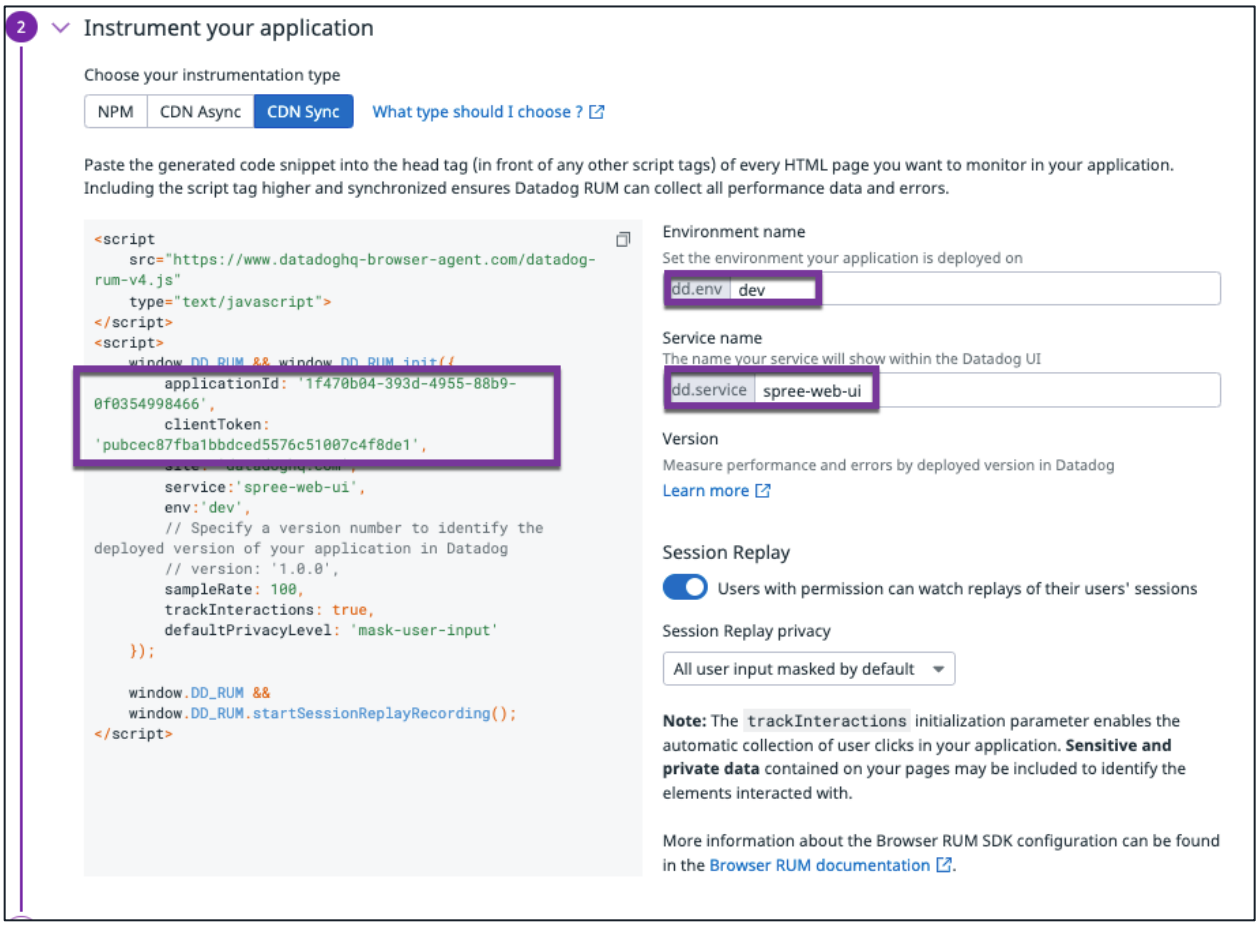


2. As a next steps, select installation method to generate code snippet – choose **CDN Sync**.

Refer to the following link to find out more on other options available: https://docs.datadoghq.com/real_user_monitoring/browser/#choose-the-right-installation-method

Enter **environment name** and **service name** (optional) this can be change in actual deployment

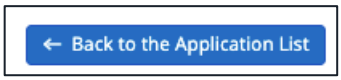
Take note of the **application Id** and **clientToken** that is uniquely generated for your account in Datadog platform.



The next step is to deploy the JS snippet into the html page of the spree application.

Copy the snippet by clicking on the **copy icon**

For now, click **Back to the Application List** on the right hand bottom corner of the page.



Connect RUM and Traces:

In our lab exercise, we will be enabling rum and traces correlation. Do take note of the prerequisites, and the correlation apply to XHR requests here: https://en.wikipedia.org/wiki/XMLHttpRequest

Prerequisites:

- You have set up APM tracing on the services targeted by your RUM applications.
- Your services use an HTTP server.

- Your HTTP servers are using a library that supports distributed tracing.
- You have XMLHttpRequest (XHR) or Fetch resources on the RUM Explorer to your allowedTracingOrigins.
- You have a corresponding trace for requests to allowedTracingOrigins.

Allow these HTTP headers, prefix with x-Datadog mentioned in:
https://docs.datadoghq.com/real_user_monitoring/connect_rum_and_traces/?tab=browserrum#how-are-rum-resources-linked-to-traces

Note: These HTTP headers are not CORS-safelisted, so you need to configure **Access-Control-Allow-Headers** on your server handling requests that the SDK is set up to monitor. The server must also accept preflight requests (OPTIONS requests), which are made by the SDK prior to every request.

For more details, refer to the following link:
https://docs.datadoghq.com/real_user_monitoring/connect_rum_and_traces?tab=browserrum

Below is an example of the copied JS snippet.

```
<script
    src="https://www.datadoghq-browser-agent.com/datadog-rum-v4.js"
    type="text/javascript">
</script>
<script>
    window.DD_RUM && window.DD_RUM.init({
        applicationId: 'xxxxxx',
        clientToken: 'xxxxx',
        site: 'datadoghq.com',
        service:'spree-web-ui',
        env:'dev',
        // Specify a version number to identify the deployed version of your
application in Datadog
        // version: '1.0.0',
        sampleRate: 100,
        trackInteractions: true,
        defaultPrivacyLevel: 'mask-user-input'
    });

    window.DD_RUM &&
    window.DD_RUM.startSessionReplayRecording();
</script>
```

Modifying the JS snippet:

1. In these steps, we will be modifying the JS script to include the steps to connect RUM and traces and environment variable for deployment purpose (in bold)

Configure environment variables for mandatory configuration as such

- **applicationId (DD_APPLICATION_ID)**
- **clientToken (DD_CLIENT_TOKEN)**
- **site (DD_SITE)**
- **service (DD_BRUM_SERVICE)**
- **env (DD_ENV)**
- **version (DD_VERSION)**

Configure **allowedTracingOrigins**, specifying the originating request called by your browser application.

In our example, we will define **regex** to specify the public ec2 host and port to be used.

```
<script
    src="https://www.datadoghq-browser-agent.com/datadog-rum-v4.js"
    type="text/javascript">
</script>
<script>
    window.DD_RUM && window.DD_RUM.init({
        applicationId: '<%= ENV['DD_APPLICATION_ID'] %>',
        clientToken: '<%= ENV['DD_CLIENT_TOKEN'] %>',
        site: '<%= ENV['DD_SITE'] %>',
        service: '<%= ENV['DD_BRUM_SERVICE'] %>',
        env: '<%= ENV['DD_ENV'] %>',
        version: '<%= ENV['DD_VERSION] %>',
        sampleRate: 100,
        trackInteractions: true,
        defaultPrivacyLevel: 'mask-user-input'
       allowedTracingOrigins: [/http:\/\/ec2.*\.amazonaws\.com:8080/]
    });

    window.DD_RUM &&
    window.DD_RUM.startSessionReplayRecording();
</script>
```

Refer to the following link for more details regarding each configuration:

- sampleRate:
https://docs.datadoghq.com/real_user_monitoring/browser/#browser-and-session-replay-sampling-configuration

- trackInteractions:
https://docs.datadoghq.com/real_user_monitoring/session_replay/privacy_options/#configuration

- defaultPrivacyLevel
https://docs.datadoghq.com/real_user_monitoring/browser/tracking_user_actions/?tab=npm#what-interactions-are-being-tracked

- allowTracingOrigins
https://docs.datadoghq.com/real_user_monitoring/connect_rum_and_traces/?tab=browserrum#setup-rum

- sessionReplay
https://docs.datadoghq.com/real_user_monitoring/session_replay/

2. Configure CORS:

In order to allow these **HTTP headers**, we will be leveraging **rack cors** to configure **Access-Control-Allow-Headers** on the spree application.

https://www.rubydoc.info/gems/rack-cors/1.0.3#installation

First, add the following to the Gemfile

```
$ cd ~/docker/ecommerce-workshop/store-frontend/src/store-front

$ vi Gemfile
```

```
# Use rack-cors to add CORS for datadog rum and traces
gem 'rack-cors', '~> 1.0.3'
```

Hint: refer to `Gemfile.instr.brum` *for the changes required.*

Then, add the CORS configuration in the application.rb, do the following:

```
$ cd ~/docker/ecommerce-workshop/store-frontend/src/store-front/config

$ vi application.rb
```

```
# configure CORS to allow datadog rum headers
    config.middleware.insert_before 0, Rack::Cors do
      allow do
        origins 'ec2*.amazonaws.com:8080'
        resource '*',
          headers: :any,
          methods: [:get, :post, :delete, :put, :options]
      end
    end
```

Hint: refer to `application.rb.brum` *for the changes required.*

3. Update the changes:

Append the JS snippet (see those in bold) into the head tag of the following html files.

```
<head data-hook="inside_head">
  <script
    src="https://www.datadoghq-browser-agent.com/datadog-rum-v4.js"
    type="text/javascript">
  </script>
  <script>
    window.DD_RUM && window.DD_RUM.init({
        applicationId: '<%= ENV['DD_APPLICATION_ID'] %>',
        clientToken: '<%= ENV['DD_CLIENT_TOKEN'] %>',
        site: '<%= ENV['DD_SITE'] %>',
        service:'<%= ENV['DD_BRUM_SERVICE'] %>',
        env:'<%= ENV['DD_ENV'] %>',
        version: '<%= ENV['DD_VERSION'] %>',
        sampleRate: 100,
        trackInteractions: true,
        defaultPrivacyLevel: 'mask-user-input',
        allowedTracingOrigins: [/http:\/\/ec2.*\.amazonaws\.com:8080/]
    });

    window.DD_RUM &&
    window.DD_RUM.startSessionReplayRecording();
```

```
  </script>

    <%= render partial: 'spree/shared/head' %>
  </head>
```

```
$ cd ~/docker/ecommerce-workshop/store-frontend/src/store-front/app/views/layouts
$ vi application.html.erb

$ cd ~/docker/ecommerce-workshop/store-frontend/src/store-
front/app/views/spree/layouts
$ vi spree_application.html.erb
```

Hint: *refer to* `application.html.erb.brum` *and* `spree_application.html.erb.brum` *for the changes required.*

Build the docker image v1.3 for store-frontend:

```
$ cd ~/docker/ecommerce-workshop/store-frontend
$ docker build . -t store-frontend:1.3

.....
Successfully built 54446e7c5956
Successfully tagged store-frontend:1.1
```

Deploy the changes:

1. Update the environment variable accordingly based on your generated Application Id, Client Token and Service name for Browser RUM, ensure version for store is set to v1.3 in .env

```
$ cd ~/docker/ecommerce-workshop/deploy
$ vi .env
```

```
STORE_VERSION=1.3

DD_APPLICATION_ID=xxxxxx
DD_CLIENT_TOKEN=xxxxxx
DD_BRUM_SERVICE=spree-web-ui
```

2. Examine the docker-compose for store-frontend, notice the additional environment variable added for Browser RUM in bold:

```
$ cat docker-compose-instr-brum.yml
```

```
store-frontend:
    container_name: store-frontend
    environment:
      - DD_AGENT_HOST=agent
      - DD_LOGS_INJECTION=true
      - DD_PROFILING_ENABLED=true
      - DD_RUNTIME_METRICS_ENABLED=true #enable runtime metrics collection
      - DD_ENV=dev
      - DD_SERVICE=store-frontend
      - DD_VERSION=${STORE_VER}
      - DD_CLIENT_TOKEN
      - DD_APPLICATION_ID
      - DD_BRUM_SERVICE
      - DD_SITE
```

```
      - RAILS_HIDE_STACKTRACE=false
      - ADS_PORT=${ADS_PORT}
      - DISCOUNTS_PORT=${DISCOUNTS_PORT}
      - ADS_ROUTE=${ADS_ROUTE}
      - DISCOUNTS_ROUTE=${DISCOUNTS_ROUTE}
    image: store-frontend:${STORE_VER}
    ports:
      - "3000:3000"
    depends_on:
      - agent
      - db
      - discounts
      - advertisements
```

3. Start the application:

```
$ cd ~/docker/ecommerce-workshop/deploy
$ docker-compose -f docker-compose-instr-brum.yml up -d
```

106-3 Verifying the collection

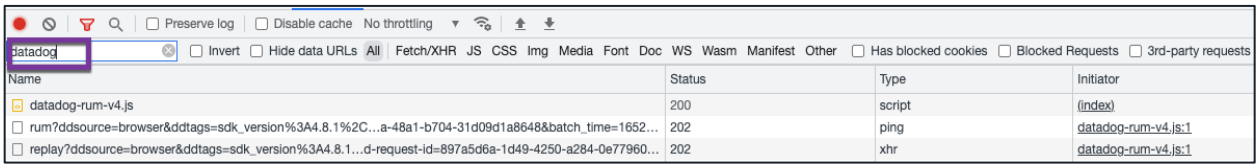
1. Browse the ecommerce website. Go to **http:<your-public-hostname>:8080**

To find out your public hostname, do the following

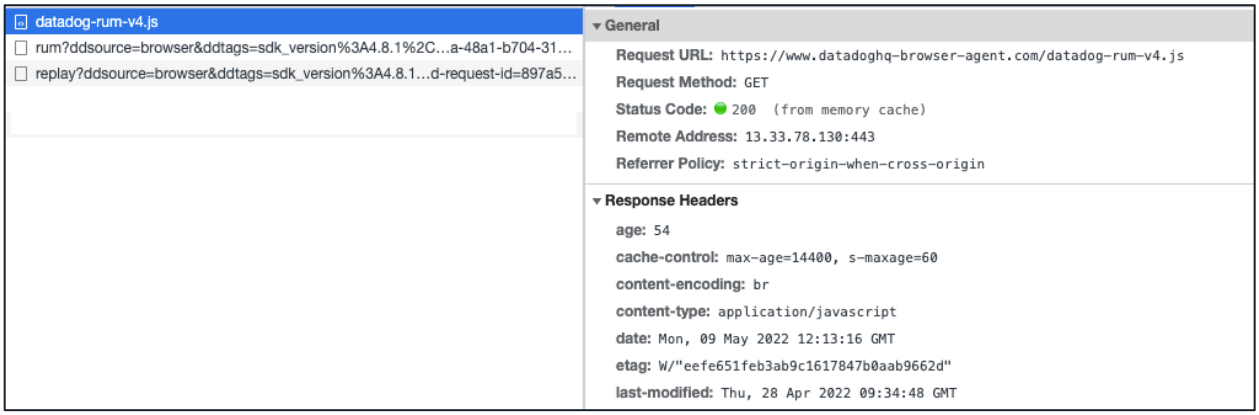
```
$ curl http://169.254.169.254/latest/meta-data/public-hostname; echo
ec2-13-236-6-50.ap-southeast-2.compute.amazonaws.com
```

2. Validate data collection via web browser **developer tool**

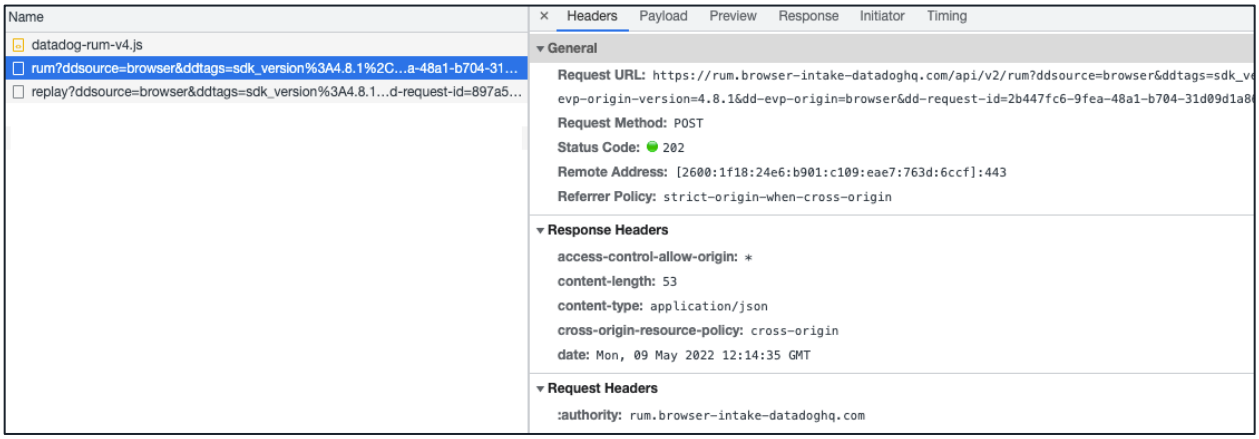
Go to Network tab, filter "**datadog**". Notice there's three requests.



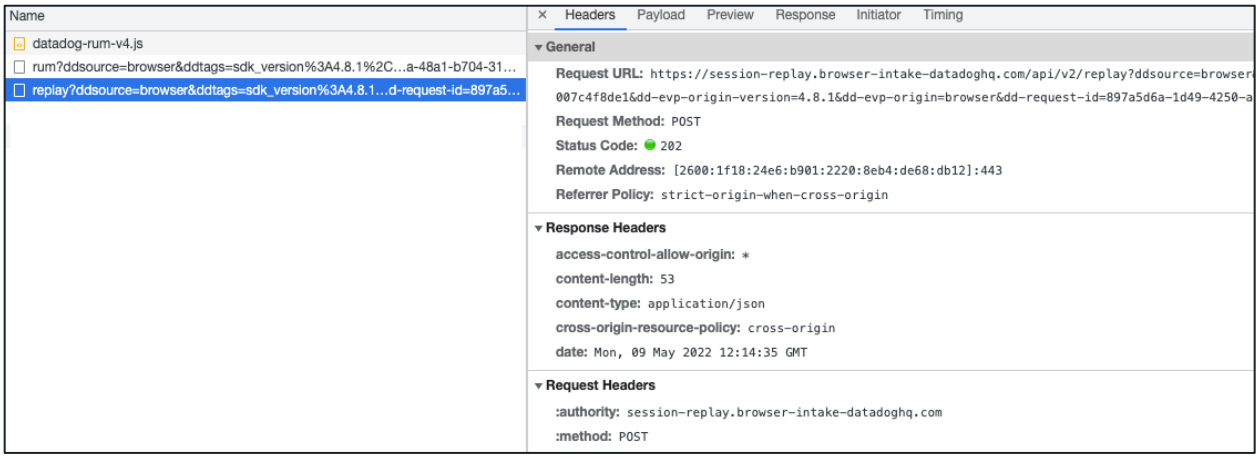
The highlight request show the download of js from CDN



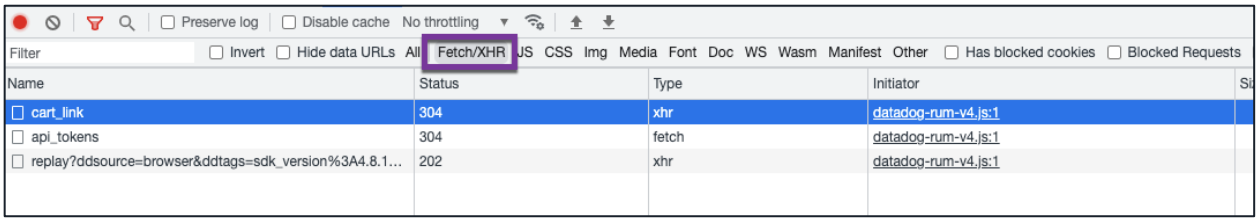
The highlighted request shows the sending of browser rum data to Datadog platform



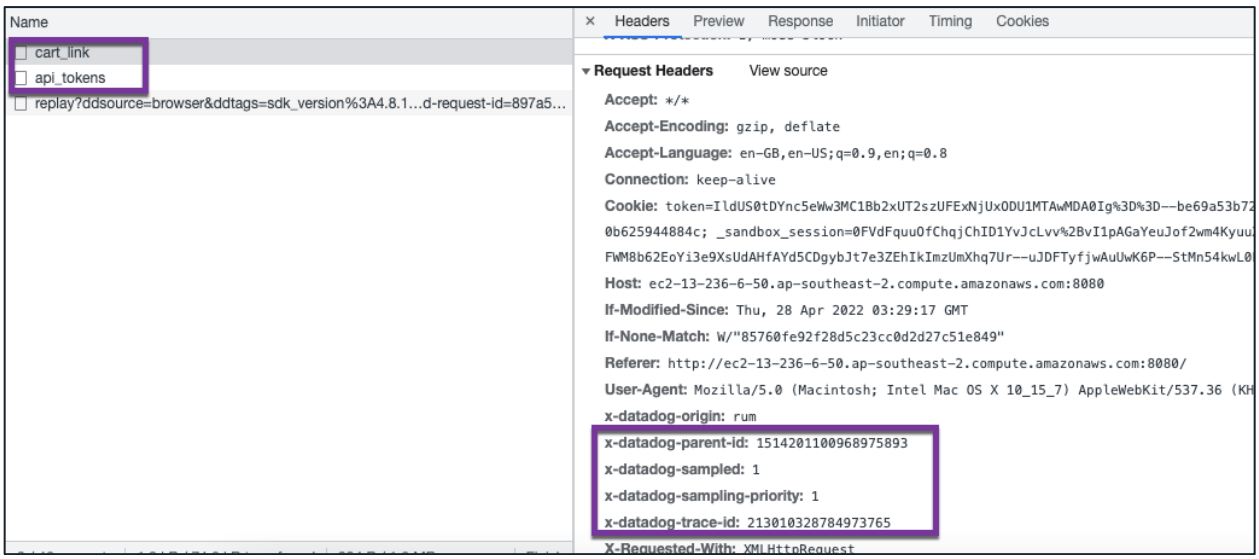
The highlighted request shows the sending of browser replay session data to Datadog platform



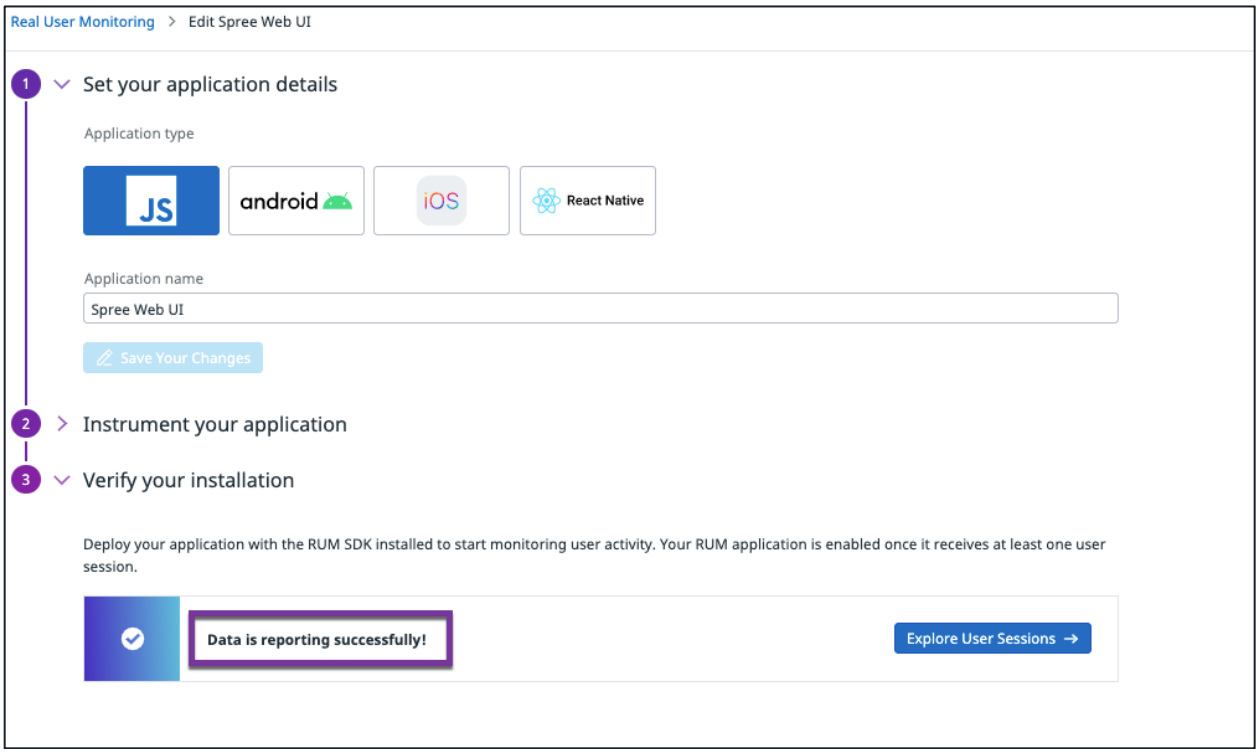
Now, remove the filter and click to filter by Fetch/XHR



The highlighted request show that the Datadog rum header has been successfully sent as shown in the request header.



Go to the edit application page for Spree Web UI, you should see that the **Data is reporting successfully!**



106-4 Visualising the Browser RUM Metrics

Generate some traffic by navigating the web pages, using different web browser like Chrome, Firefox, and Safari

RUM List:

The **Real User Monitoring->Applications (**navigate to **UX Monitoring->RUM Applications)** page outlines all the RUM applications reporting to your Datadog platform. In nutshell, shows **User Sessions, Loading Time and Error** details for every RUM application.

From the **UX Monitoring->RUM Applications** and by using the search box click on **Application Overview** of our newly created RUM application **Spree Web UI.**

https://app.datadoghq.com/rum/list

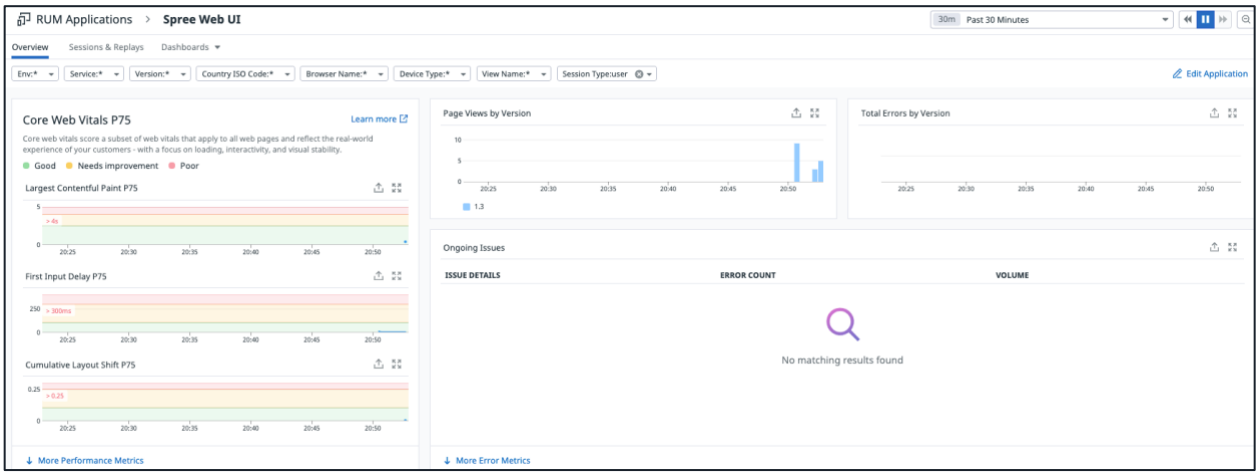
Application Overview:

RUM view events collect extensive performance metrics for every single page view. Datadog recommends analyzing the performance metrics in the following ways:

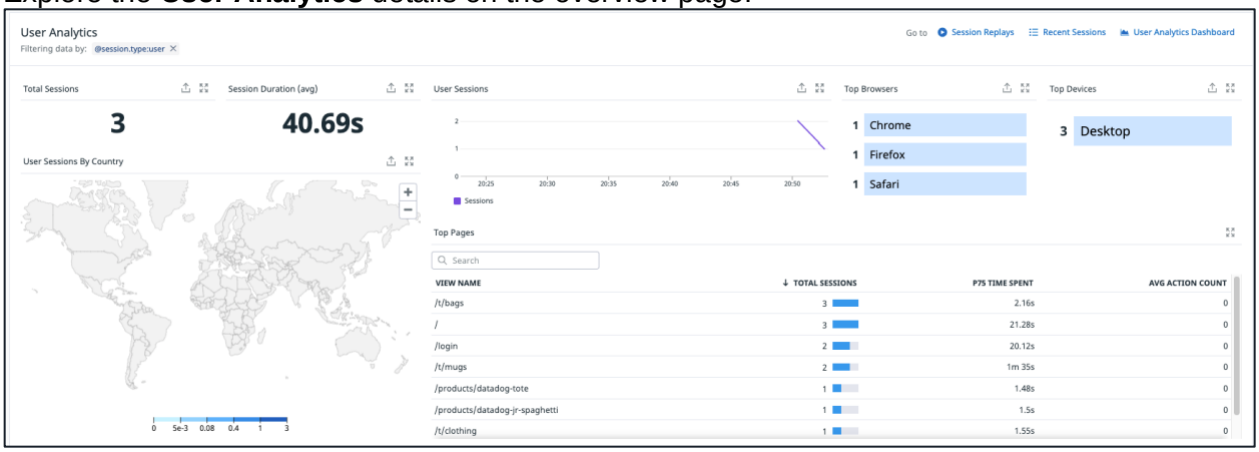
- **Dashboards:** provides you with a high-level view of your application's performance. For example, the out-of-the-box Performance Overview dashboard can be filtered on default attributes collected by RUM to surface issues impacting a subset of users. This dashboard can be cloned and customized to your specific needs. All RUM performance metrics can be used in dashboard queries.
- **RUM waterfall:** accessible for every single RUM view event in the RUM Explorer, it lets you troubleshoot the performance of a specific page view. It shows how your website assets and resources, long tasks, and frontend errors affect performance for your end users, at the page level.

Explore the **overview page**:

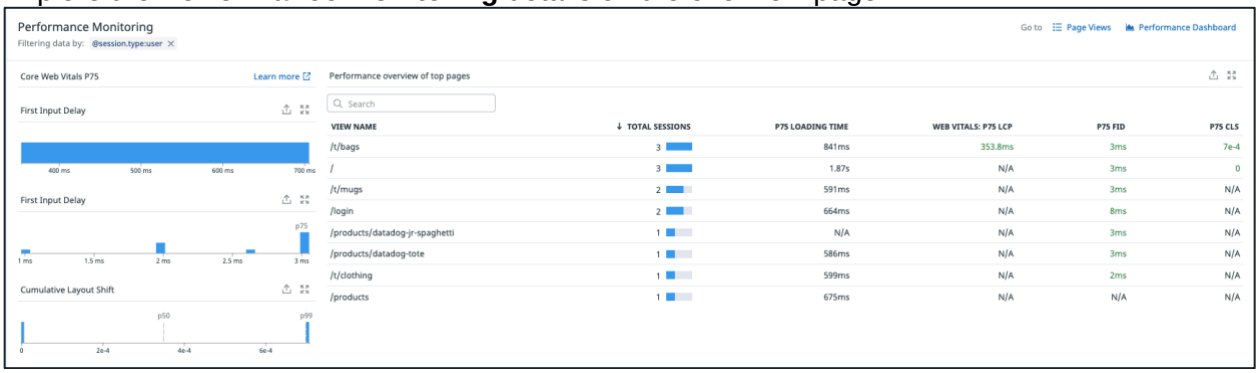
The overview page provides Google's Core Web Vitals which are a set of three metrics designed to monitor a site's user experience. These metrics focus on giving you a view of **load performance, interactivity, and visual stability**. Each metric comes with guidance on the range of values that translate to good user experience. Datadog recommends monitoring the 75th percentile for these metrics. More details can be found here: https://docs.datadoghq.com/real_user_monitoring/browser/monitoring_page_performance/#core-web-vitals



Explore the **User Analytics** details on the overview page:



Explore the **Performance Monitoring** details on the overview page:



Explore the **Errors details** on the overview page:

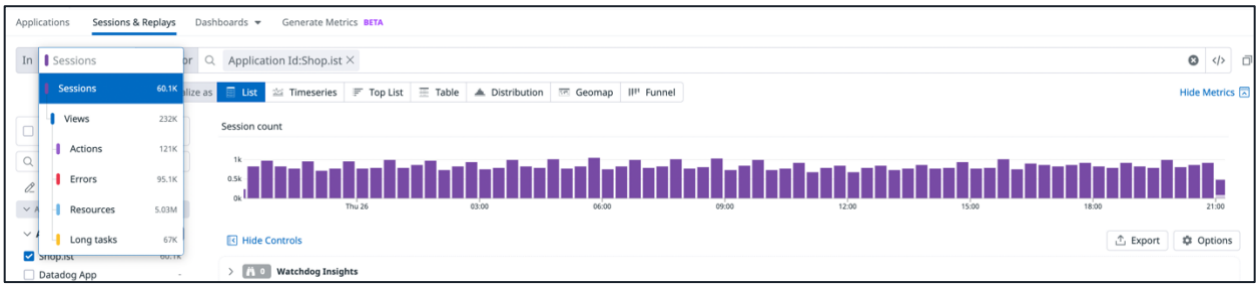
Session & Replays:

Datadog's Session Replay allows you to capture and visually replay the web browsing experience of your users. Combined with RUM performance data, Session Replay is beneficial for error identification, reproduction, and resolution, and provides insights into your web application's usage patterns and design pitfalls.

Let's start looking at some user session data.

1. Navigate to **UX Monitoring > Sessions & Replays**: https://app.datadoghq.com/rum/explorer

2. Along the top of the **Session count** graph, you can click to explore the types of data that RUM makes accessible to you:

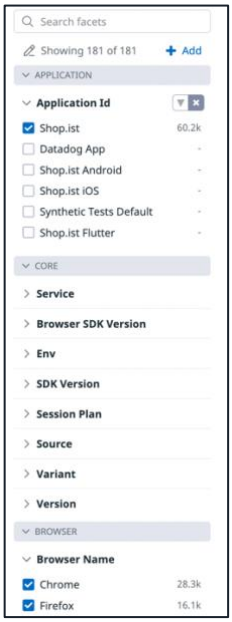


- **Sessions** groups interactions by user session and includes information about session duration, pages visited, interactions, resources loaded, errors, and more. By clicking on a row, you can also look at the **Attributes** tab to determine the user's browser and other information.
- **Views** shows you what pages users are visiting. Clicking on a row provides more detail on the view, including page load speeds and resources used.
- **Actions** lists actions taken by a user such as clicks and any custom actions you've defined with the addUserAction API.

- **Errors** provides a list of errors experienced on the user side.
- **Resources** provides a list of all resources served to users.
- **Long Tasks** provides a listing of any task in a browser that blocks the main thread for more than 50ms.

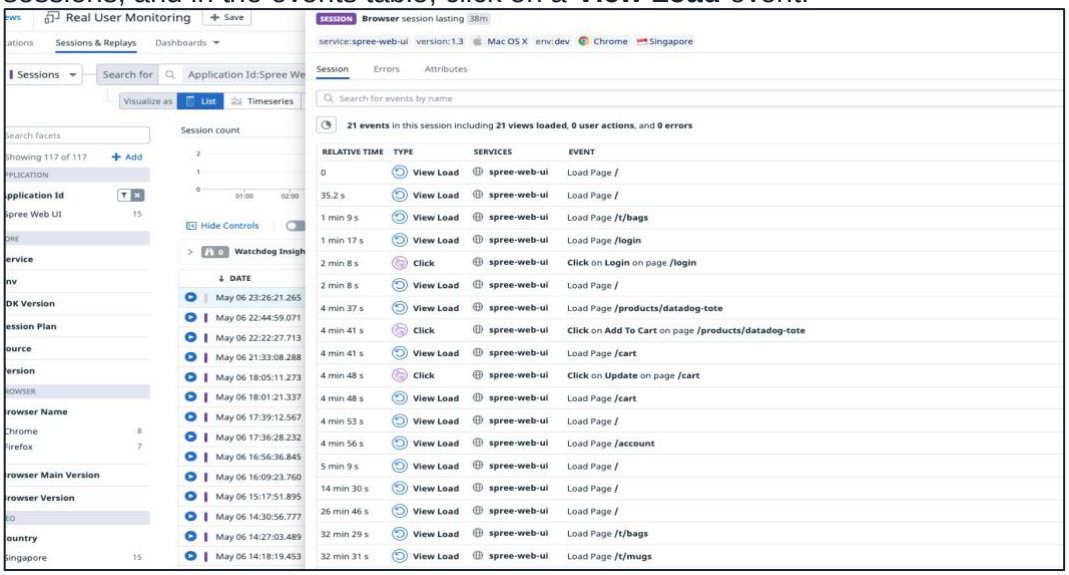
Further information on the data that RUM collects is available here.

3. Along the left-hand side, you'll see a list of **Facets**. These allow you to filter your data by a number of different criteria.

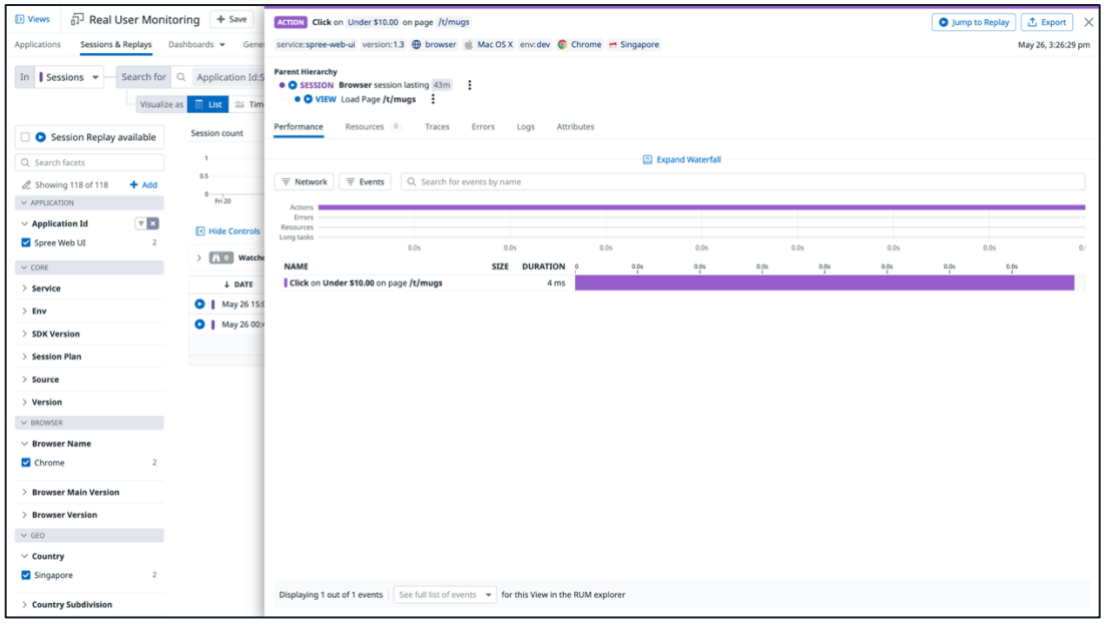


One of the major benefits that Datadog RUM offers is the ability to dive into granular data.

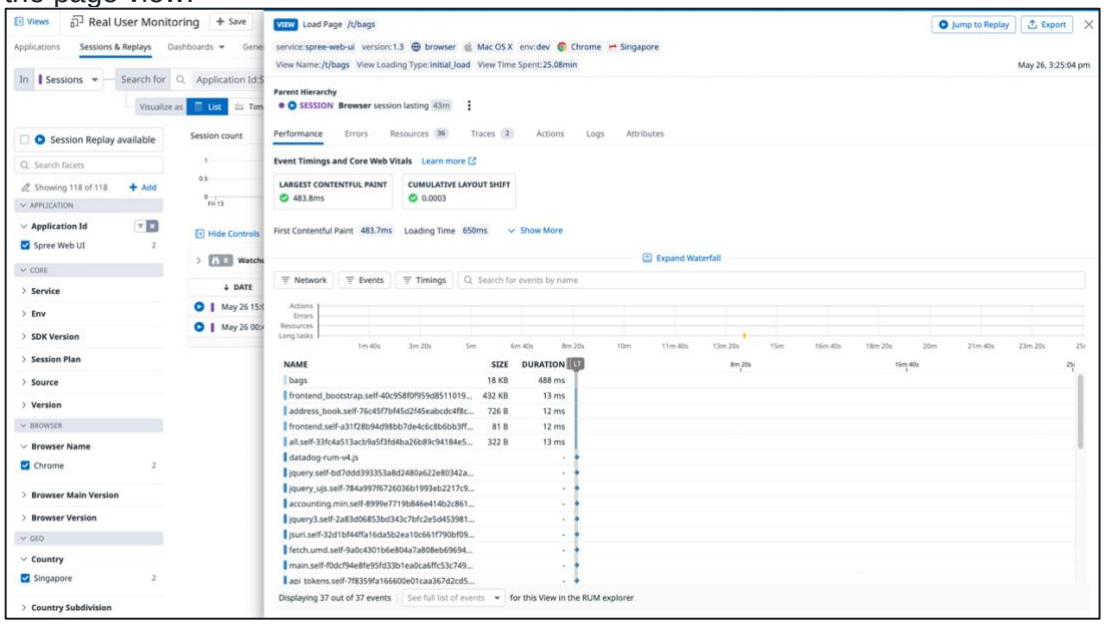
1. First, navigate to the main **Sessions & Replays** tab. Next, click into one of the user sessions, and in the events table, click on a **View Load** event:



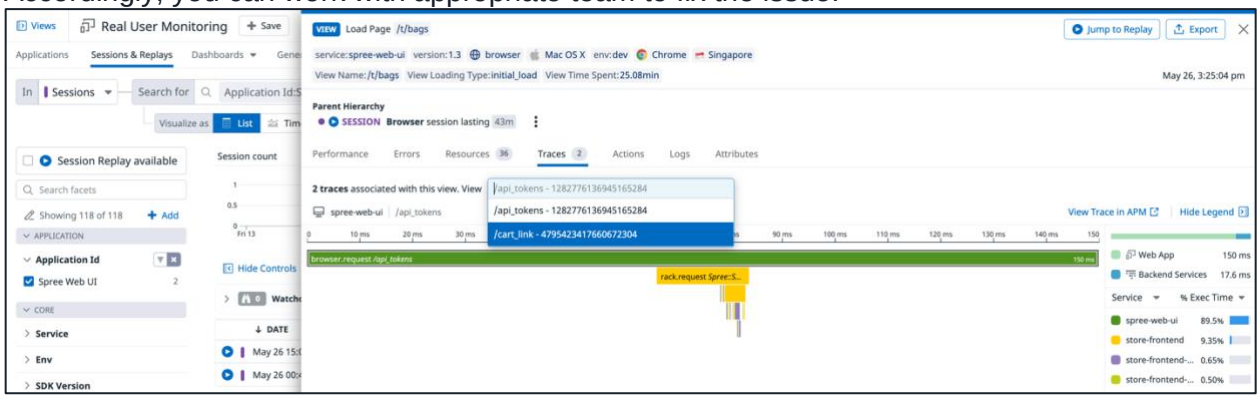
2. By looking into this single view, you can see performance, resources, errors, and user actions associated with this page load. This data can be provided to the responsible engineers to help debug and rectify issues for your users. By sharing the URL from the browser on your view, an engineer will be brought to the same panel you are looking at so there is no discrepancy in information.
3. Click on one of the **user actions** from the list. You'll find a detailed view of how that action performed, any logs associated with it, what resources it used, and any errors that occurred because of the action.



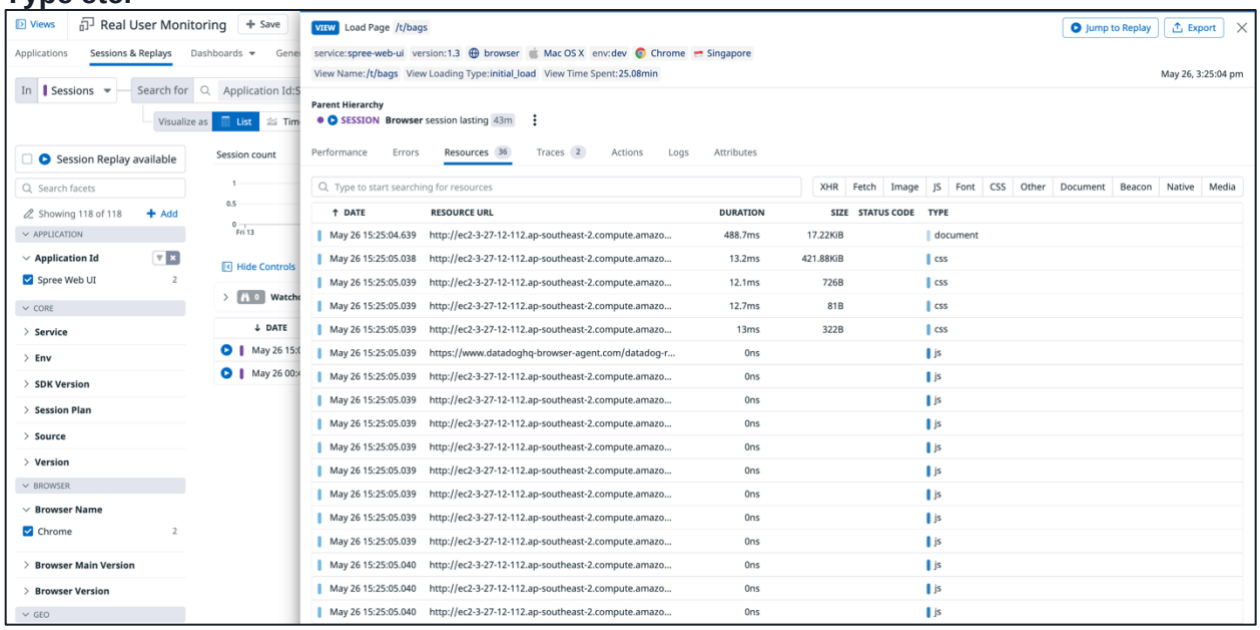
4. Close the action by clicking on **X** located on the right corner. Click on one of the user **Page View** from the list. You'll find a detailed about the any Performance issue, related Traces, any logs associated with it, what resources it used, and any errors that occurred because of the page view.



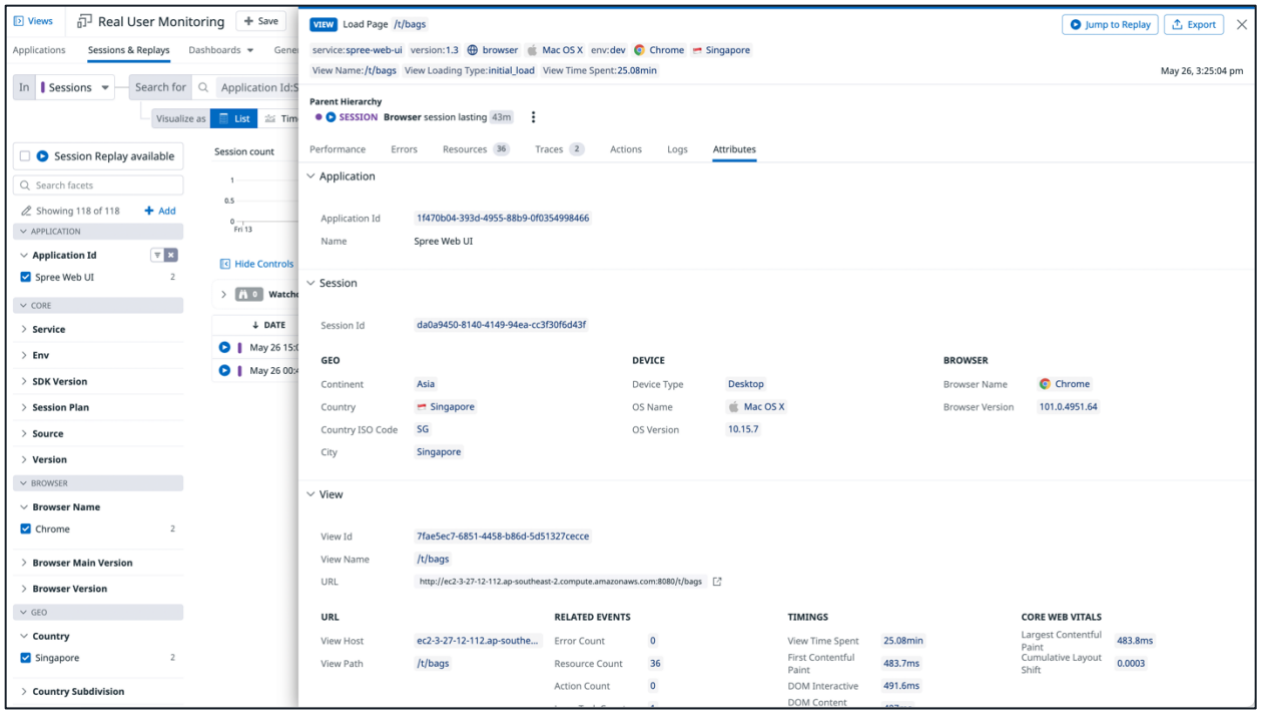
Click on **Traces** tab, this shows that our RUM to Trace co-relation is working fine. This makes it so useful to analyse problems. You can pin-point whether the problem is related to rendering response time of your web page (RUM) or the problem lies in the code(APM). Accordingly, you can work with appropriate team to fix the issue.



Explore the different **Resources** loaded on this page with details about **Duration, Size, Type etc.**

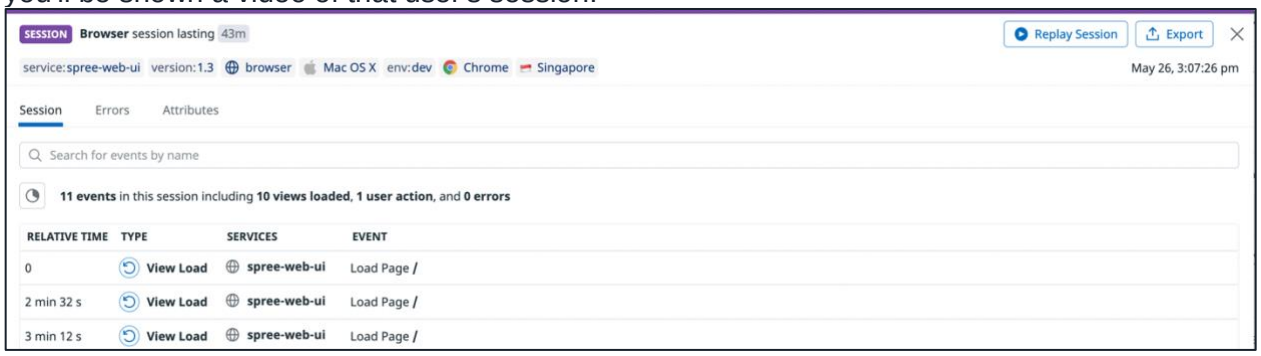


You also have quick access to all the **Attributes** associated to this Page View:



You have all information at your disposal for RCA and all with single clicks.

5. Notice the button that says **Replay Session** at the top right of the panel? Click on it and you'll be shown a video of that user's session.

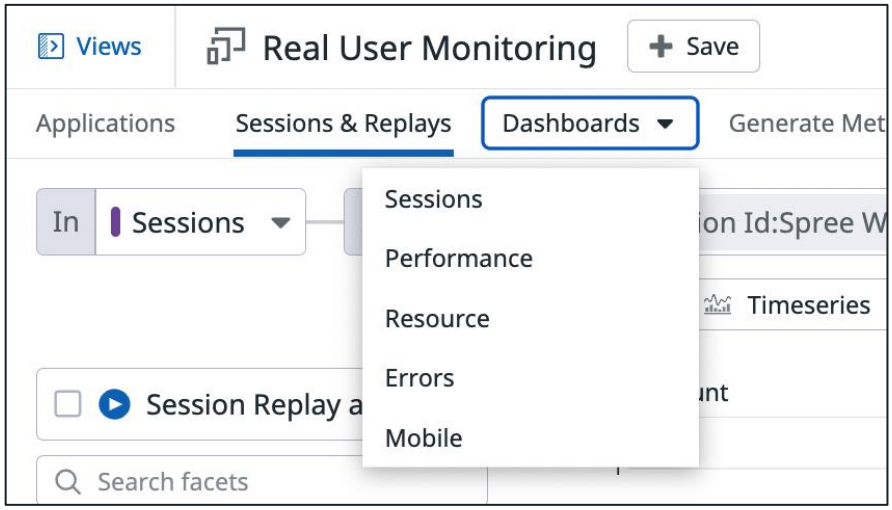


The **startSessionReplayRecording()** function from earlier is what allows you to play back how a user interacted with your site to determine what led to an error or performance issue.

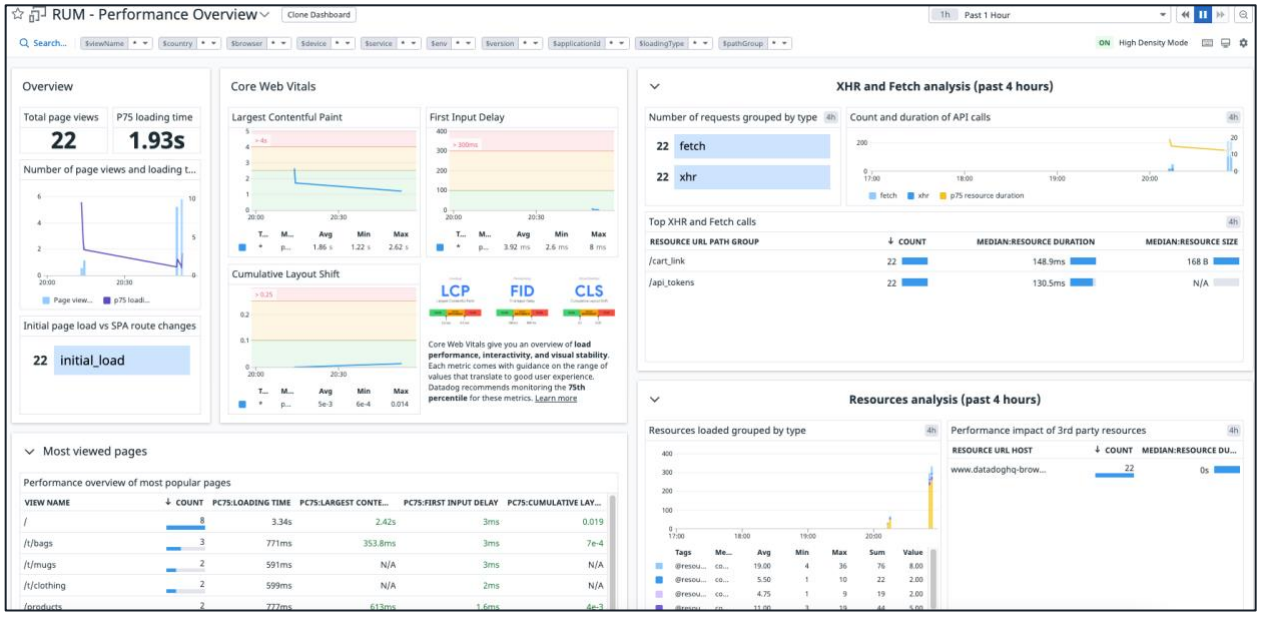
Dashboards:

1. You can now close the page load and user session panels. You should have a view of your application with a **Dashboards** dropdown at the top If you closed the page for your RUM application or don't see that dropdown, you can open your application again from the RUM Application List.

2. By clicking on the **Dashboards** dropdown, you can view the dashboards Datadog automatically creates for RUM Applications:



3. Take a look at the **Performance** dashboard. The performance overview dashboard offers a bird's-eye view of RUM applications. It is separated into several sections:

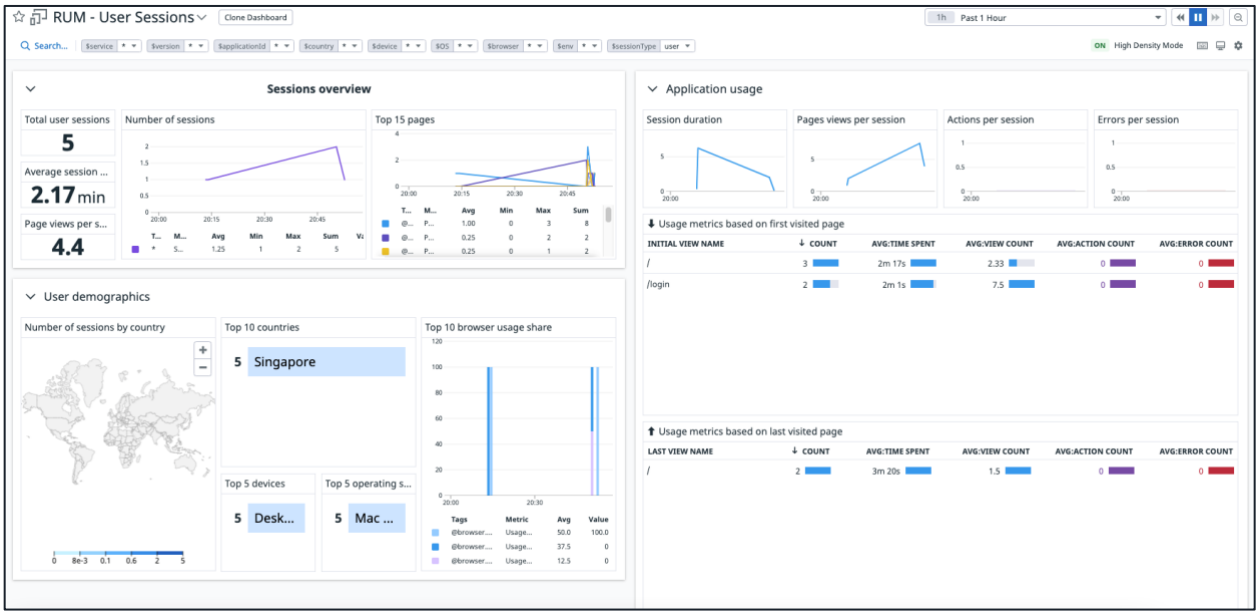


- **Overview:** This contains the number of overall page views in this time span and some top-level metrics on load time.
- **Core Web Vitals:** Gives you an overview of load performance, interactivity, and visual stability.
- **Most Viewed Pages:** Shows the same metrics from the previous sections but organized by page path.
- **Long tasks analysis:** Shows what long thread blocking tasks users are experiencing.
- **XHR and Fetch analysis:** Provides insight into what API requests are being used and their performance.
- **Resources analysis:** Shows the resources loaded grouped by type (e.g., JS, CSS, etc).

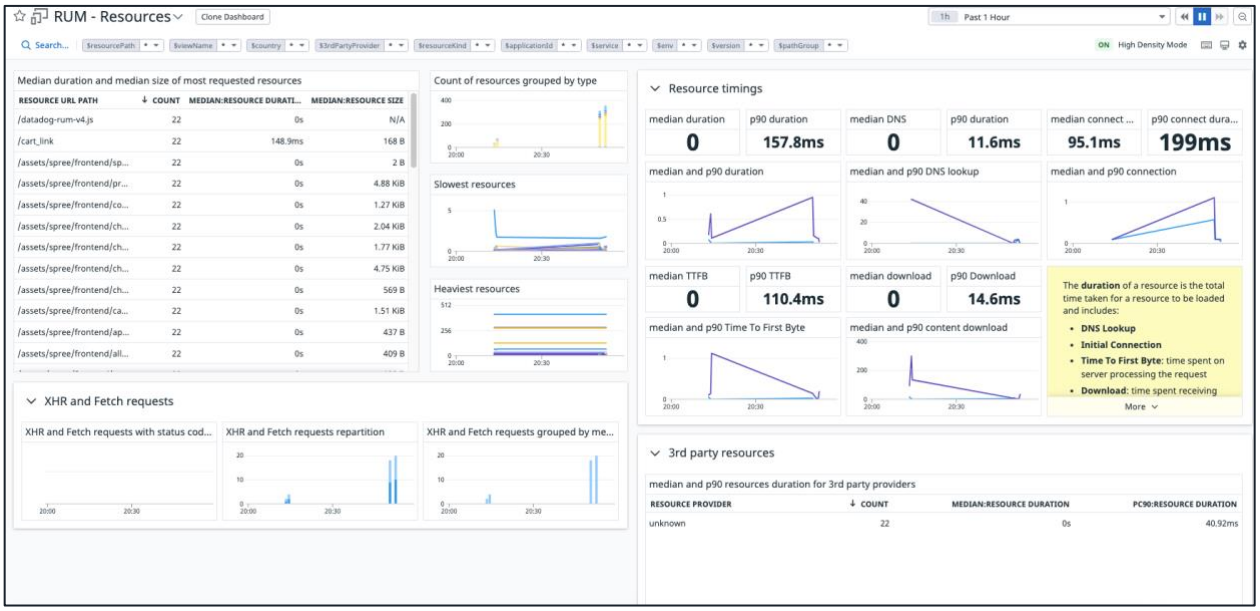
4. If you go back to your RUM Application List, you can also check out the **Sessions**, **Resource**, and **Errors** dashboards to view more details for those.

You can easily access these by hovering the **Dashboards** on the left-hand navigation and entering RUM in the search field provided.

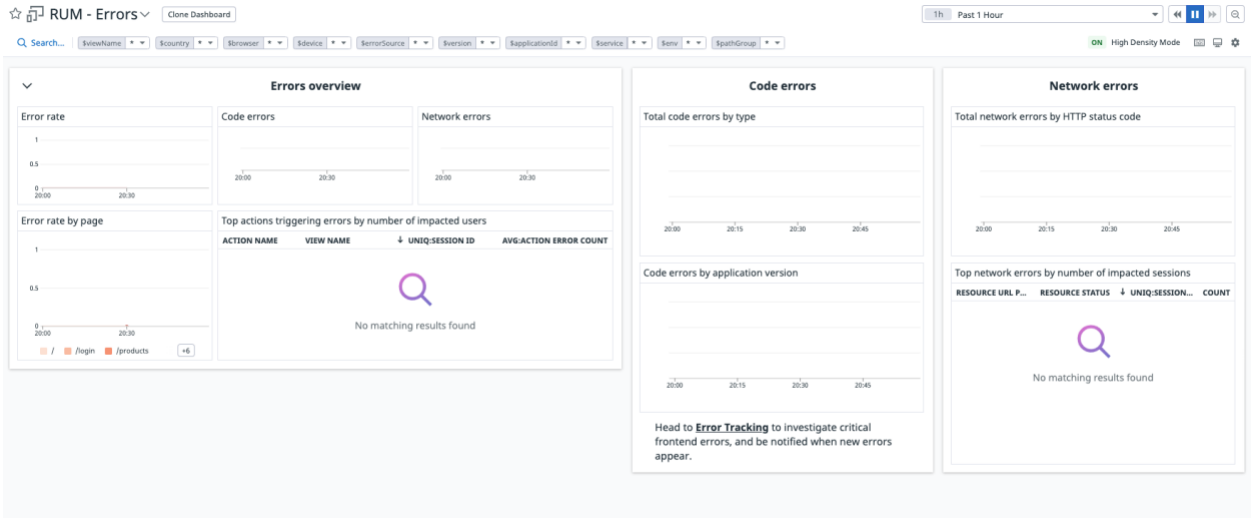
RUM – User Sessions Dashboard:



RUM – Resources Dashboard:



RUM – Errors Dashboard:



107 Synthetics Lab

Synthetic tests allow you to observe how your systems and applications are performing using simulated requests and actions from around the globe. Datadog tracks the performance of your webpages and APIs from the backend to the frontend, and at various network levels (HTTP, SSL, DNS, WebSocket, TCP, UDP, ICMP, and gRPC) in a controlled and stable way, alerting you about faulty behavior such as regressions, broken features, high response times, and unexpected status codes.

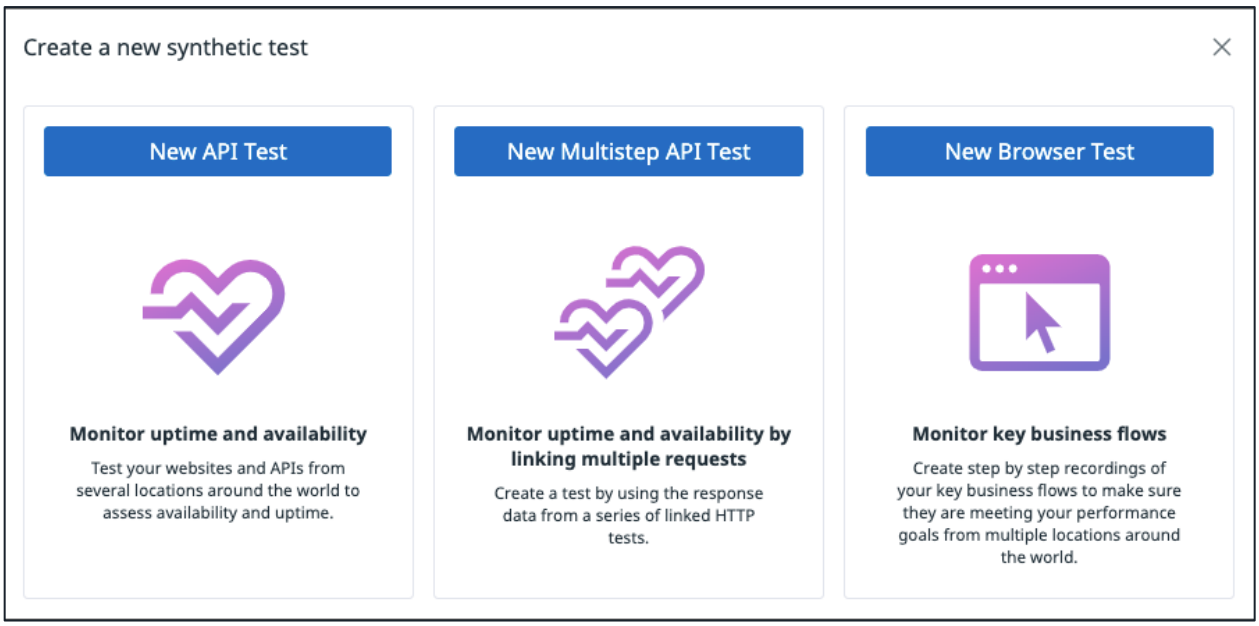
For the first part in API testing, we will be creating API test using https://reqres.in/ to test user login using a **Single API Test**, then create a **Multiple Step API** test to get a data from a single user by using the id return from the response of the first api

For the second part, we will be creating a browser test for the spree ecommerce app by using private location.

107-1 API Test

1. Go to https://app.datadoghq.com/synthetics/tests, top right-hand corner click **New Test**

2. Click **New API Test**



3. Choose request type as **HTTP**. Enter URL as **POST https://reqres.in/api/login**

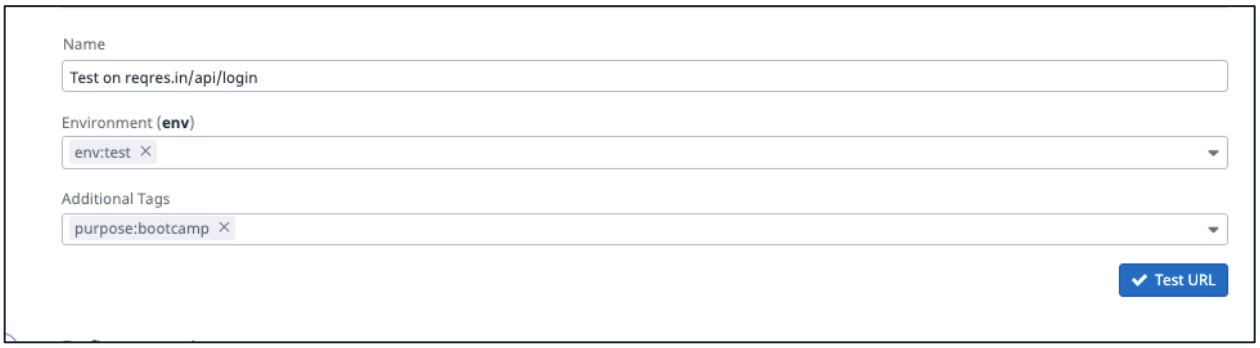
Click and expand **Advanced Options**

Click on **Request Body** and change Body Type to **application/json**

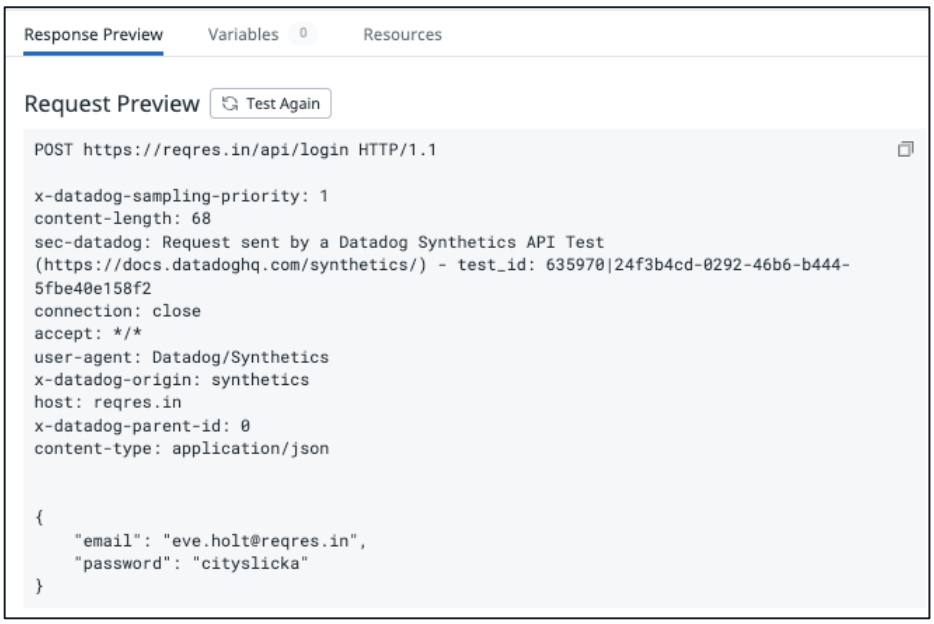
Enter the login detail:

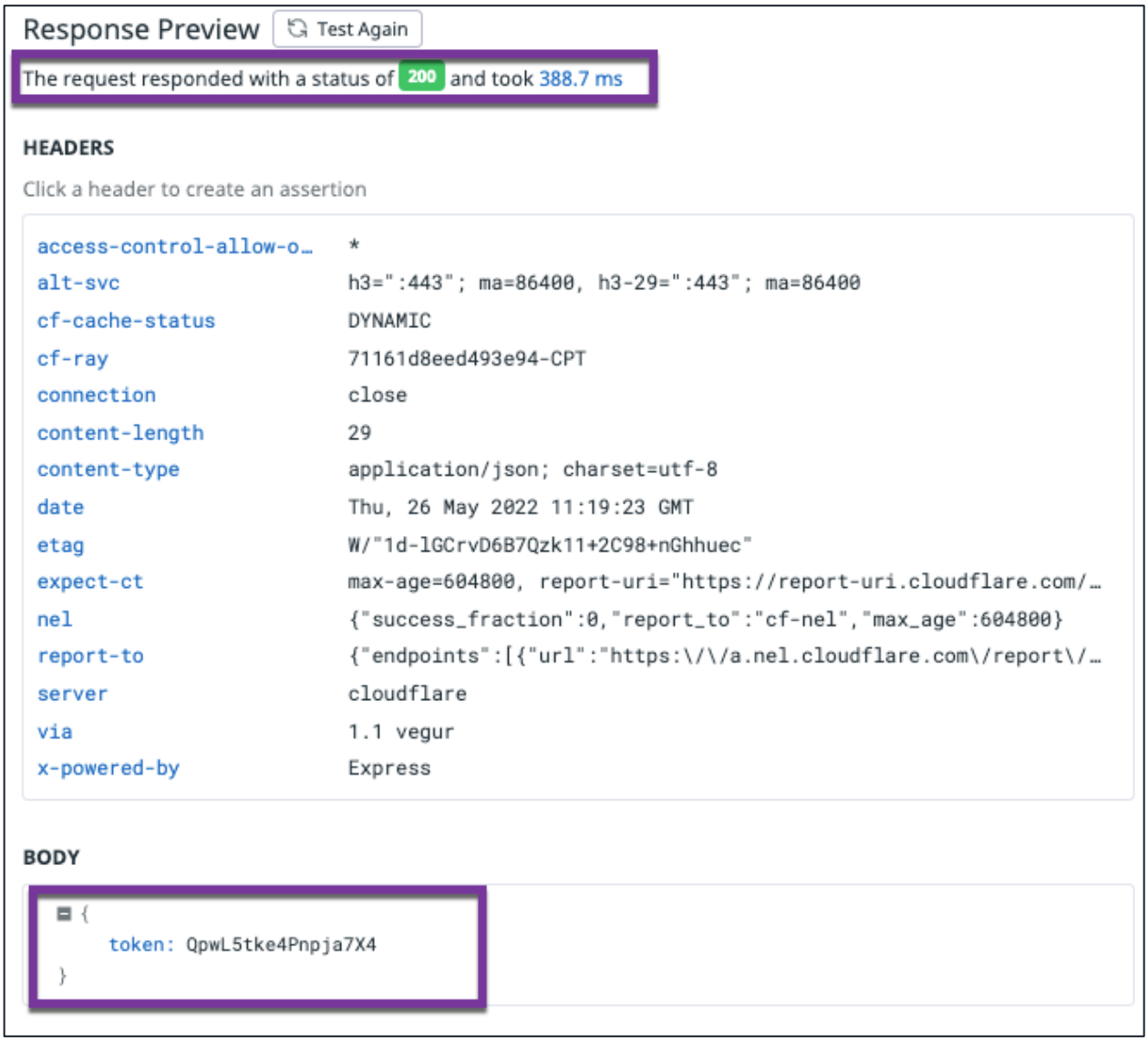
```
{
    "email": "eve.holt@reqres.in",
    "password": "cityslicka"
}
```

Continue to fill up the rest of the section as in below screenshot and click **Test URL** button:



You will see the result of the **Request Review, Response Preview** as well as the body showing on the right hand side of the screen.

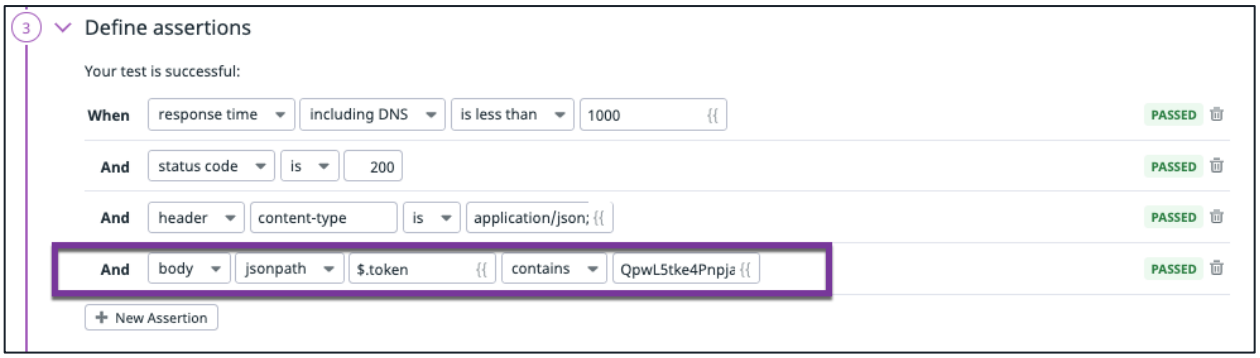




4. You will see the **assertions** have been automatically setup as part of the **Test URL**, review the condition and make adjustment as needed. For now, let's keep them default and include a **New Assertion** to ensure that we are getting the token in the response body.

To do that, simply click on the line token in the body

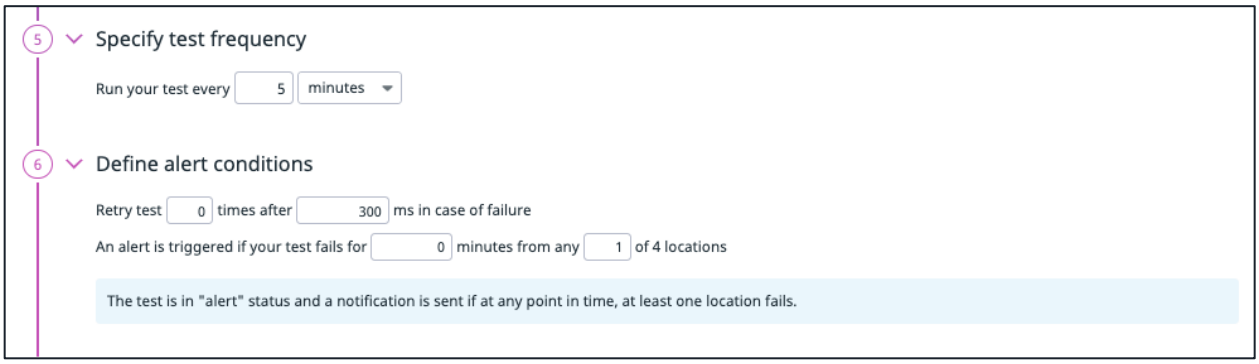
You will notice that a new assertion has been added for the token



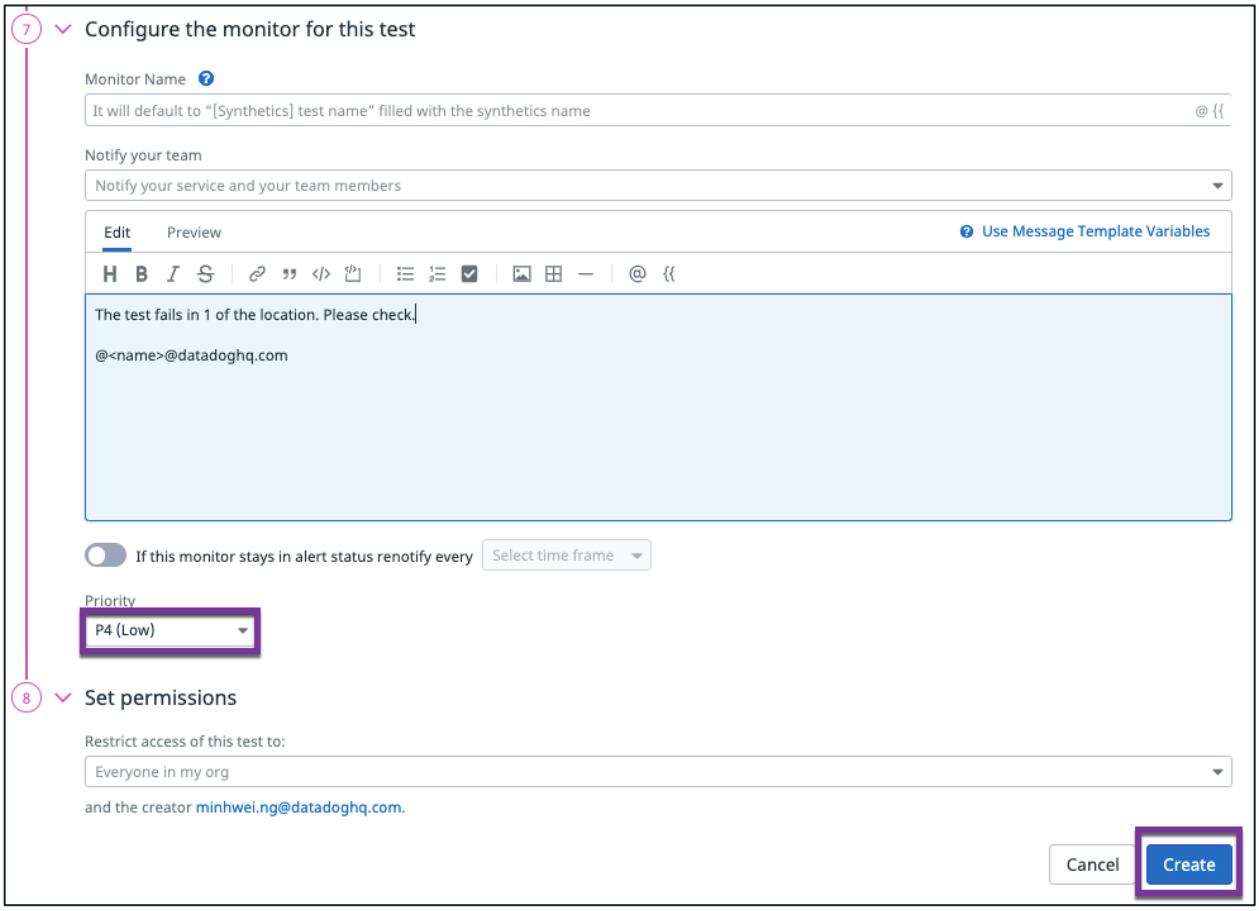
Now, let change the condition to matches **regex** and update **^[a-zA-Z0-9]+$** since the token value is not fixed. Ensure PASSED as an indication that the regex is working.

5. Select the following location:

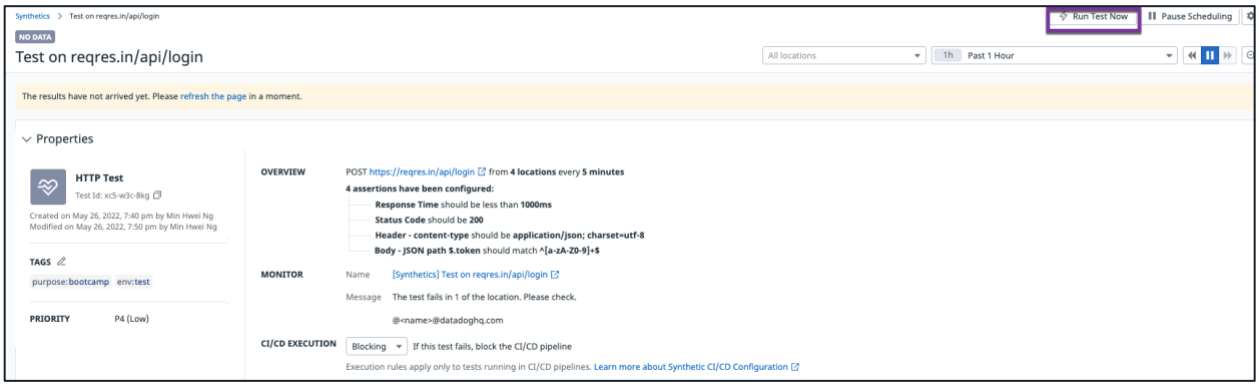
Setup the test frequency to **5 minutes**



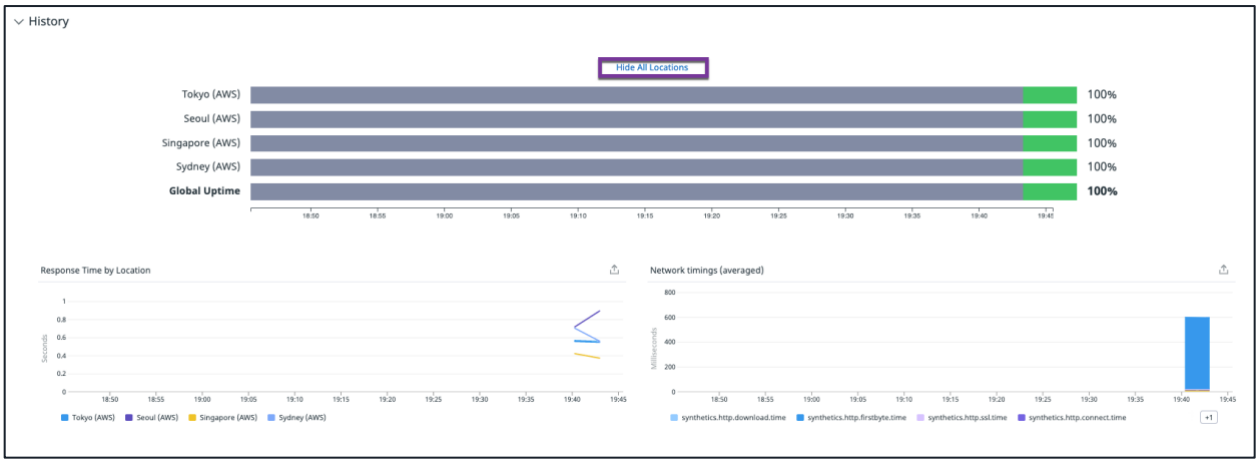
Configure the monitor and alert message, set priority to **P4 Low** and click **create**



Validation:
You can click on **Run Test Now**, or wait for the schedule to kicks in the next 5 minutes



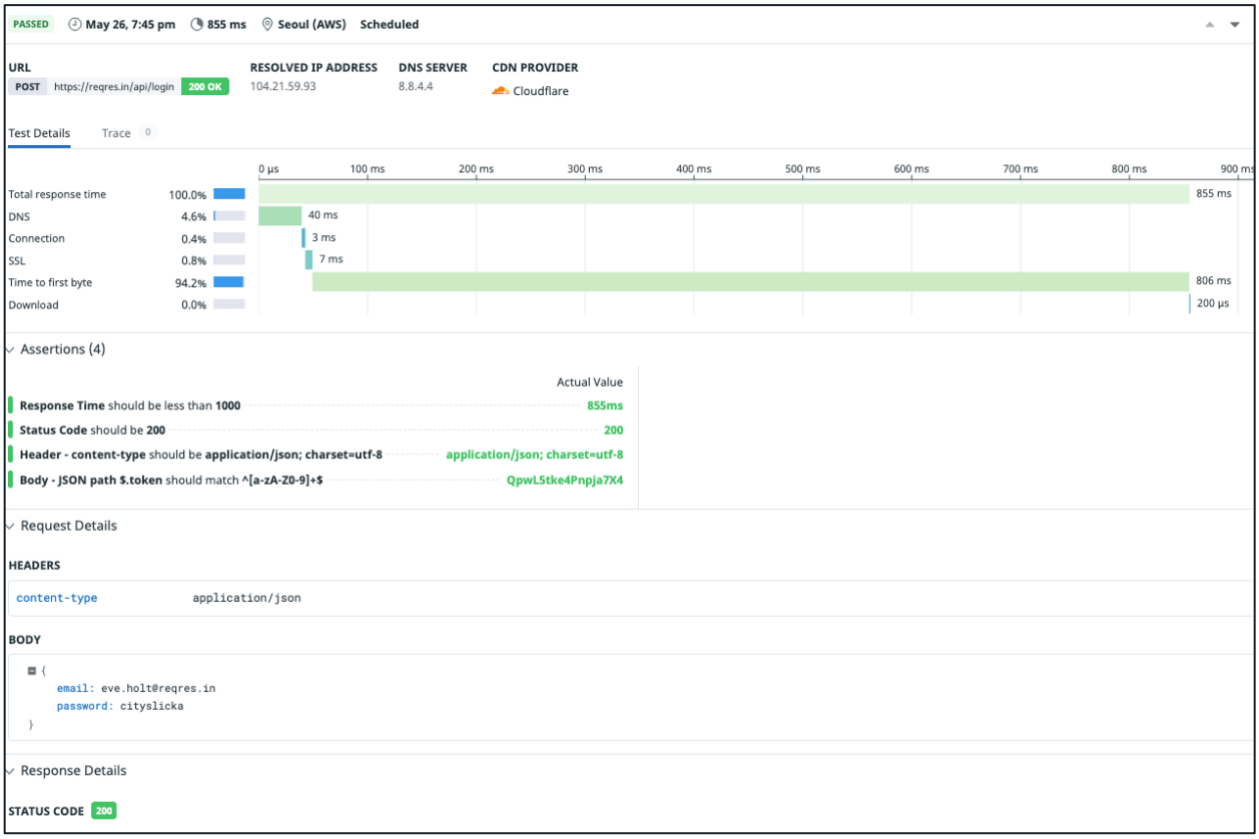
Expand to see all location:



As you scroll below you can see the test run whether it is Manually triggered or Scheduled

STATUS	DATE	LOCATION	RUN TYPE
PASSED	1 min ago May 26, 2022, 7:45 pm	Seoul (AWS)	Scheduled
PASSED	1 min ago May 26, 2022, 7:45 pm	Singapore (AWS)	Scheduled
PASSED	1 min ago May 26, 2022, 7:45 pm	Sydney (AWS)	Scheduled
PASSED	1 min ago May 26, 2022, 7:45 pm	Tokyo (AWS)	Scheduled
PASSED	3 mins ago May 26, 2022, 7:43 pm	Sydney (AWS)	Manually triggered
PASSED	3 mins ago May 26, 2022, 7:43 pm	Seoul (AWS)	Manually triggered
PASSED	3 mins ago May 26, 2022, 7:43 pm	Singapore (AWS)	Manually triggered
PASSED	3 mins ago May 26, 2022, 7:43 pm	Tokyo (AWS)	Manually triggered
PASSED	6 mins ago May 26, 2022, 7:40 pm	Seoul (AWS)	Scheduled
PASSED	6 mins ago May 26, 2022, 7:40 pm	Sydney (AWS)	Scheduled
PASSED	6 mins ago May 26, 2022, 7:40 pm	Singapore (AWS)	Scheduled
PASSED	6 mins ago May 26, 2022, 7:40 pm	Tokyo (AWS)	Scheduled

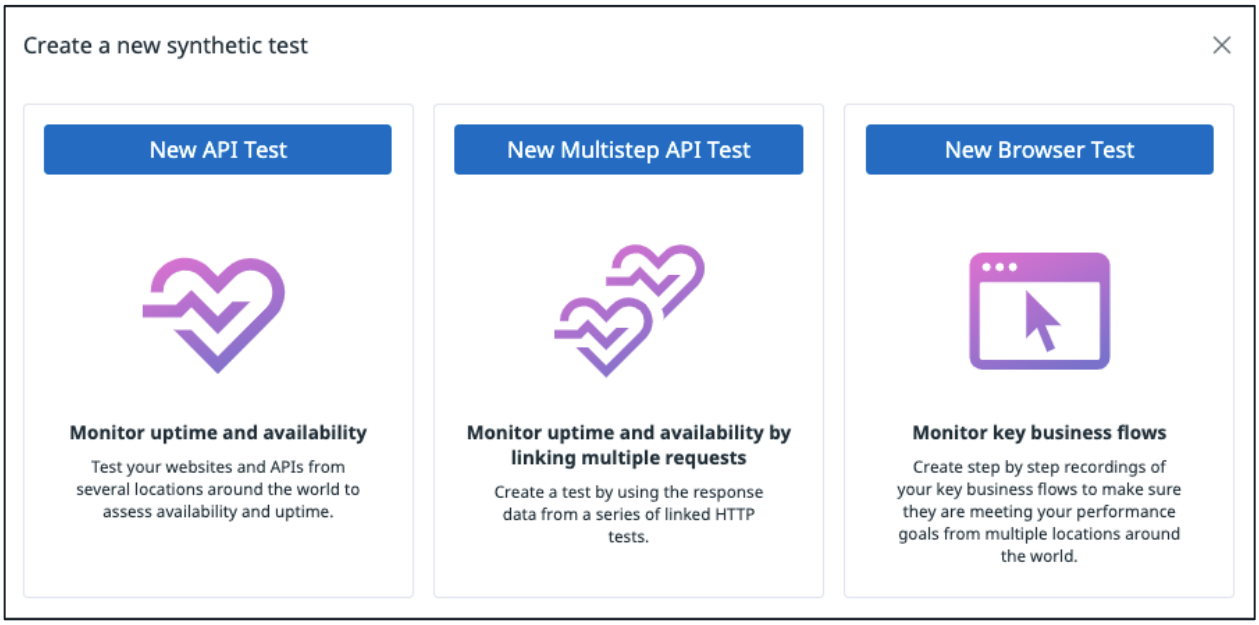
Select any test and see the detail in the side panel:


PASSED
May 26, 7:45 pm
855 ms
Seoul (AWS)
Scheduled
URL
POST https://reqres.in/api/login 200 OK
RESOLVED IP ADDRESS
104.21.59.93
DNS SERVER
8.8.4.4
CDN PROVIDER
Cloudflare
Test Details
Trace 0
0 µs
100 ms
200 ms
300 ms
400 ms
500 ms
600 ms
700 ms
800 ms
900 ms
Total response time 100.0% 855 ms
DNS 4.6% 40 ms
Connection 0.4% 3 ms
SSL 0.8% 7 ms
Time to first byte 94.2% 806 ms
Download 0.0% 200 µs
Assertions (4)
Actual Value
Response Time should be less than 1000 855ms
Status Code should be 200 200
Header - content-type should be application/json; charset=utf-8 application/json; charset=utf-8
Body - JSON path $.token should match ^[a-zA-Z0-9]+$ QpwL5tke4Pnpja7X4
Request Details
HEADERS
content-type application/json
BODY
{
email: eve.holt@reqres.in
password: cityslicka
}
Response Details
STATUS CODE 200

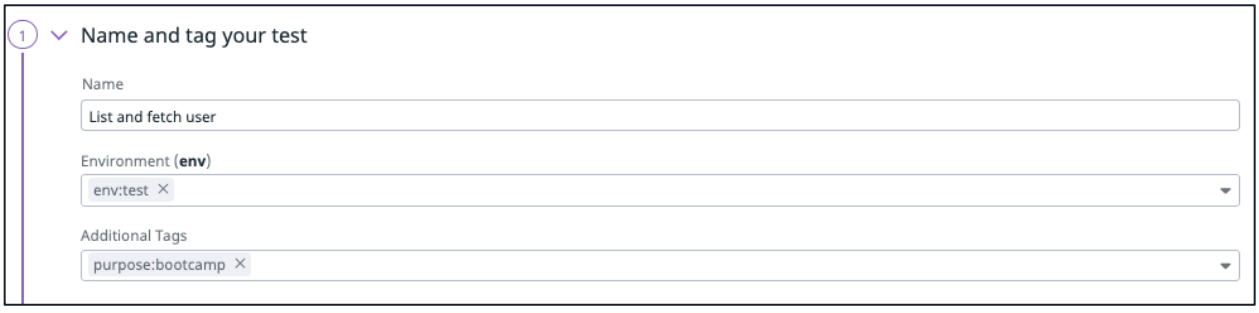
107-2 Multistep API Test

1. Go to https://app.datadoghq.com/synthetics/tests, top right hand corner click **New Test**

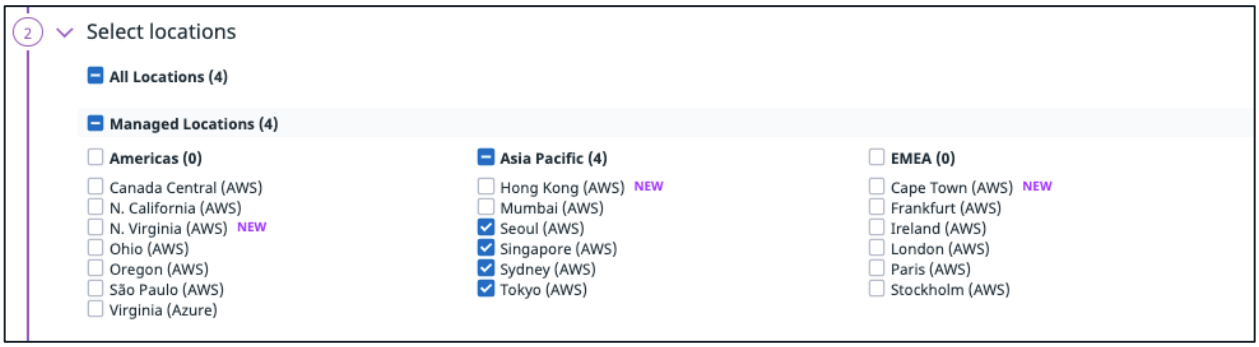
Click **New Multistep API Test**



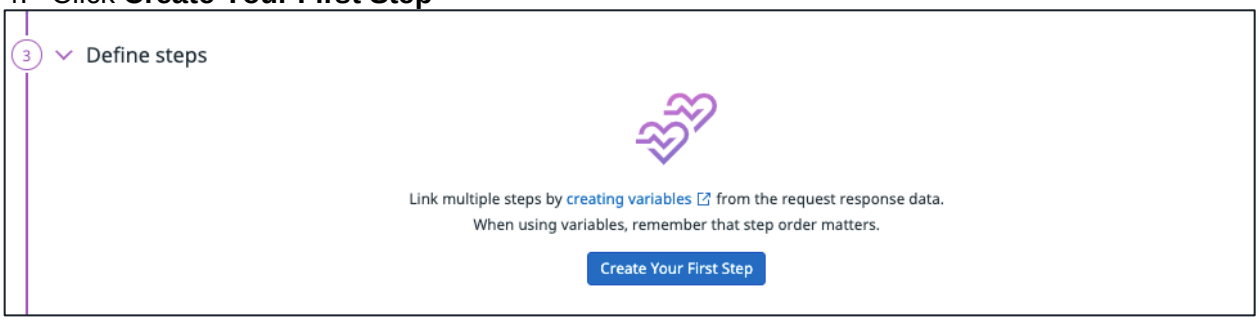
2. Enter the details as in below screenshots:



3. Select locations:

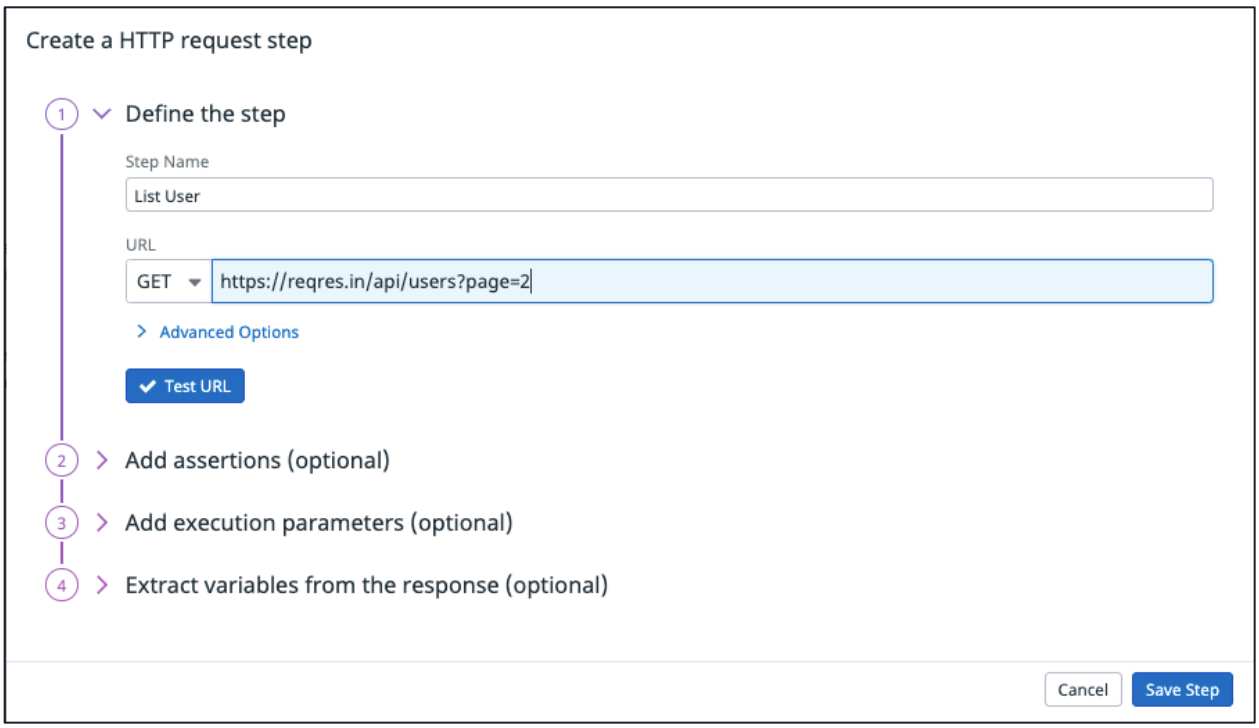


4. Click **Create Your First Step**

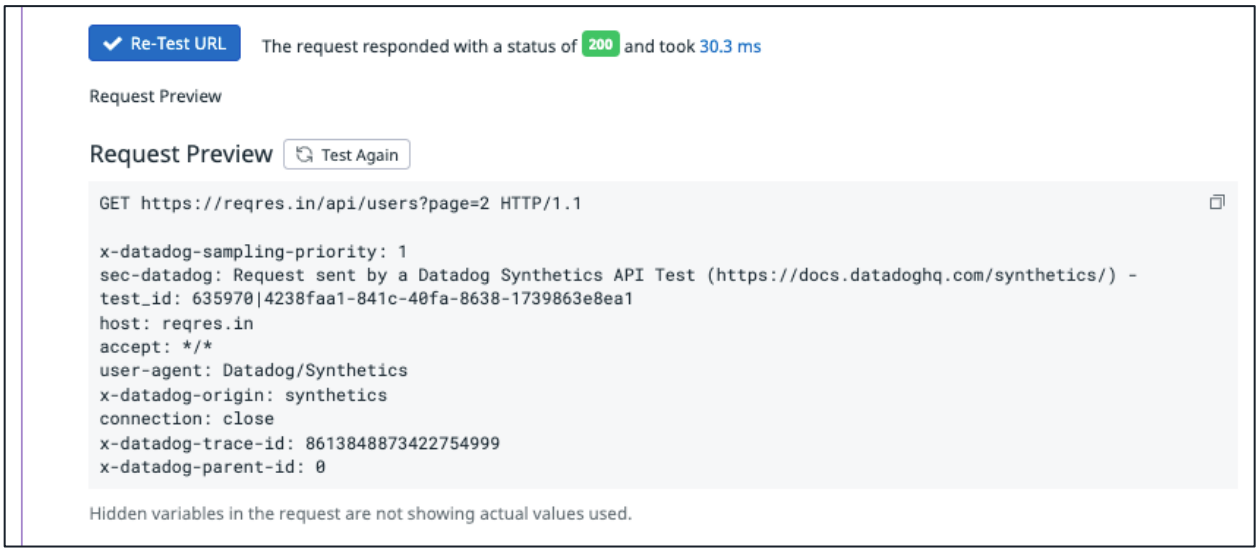


5. Provide the name of the step

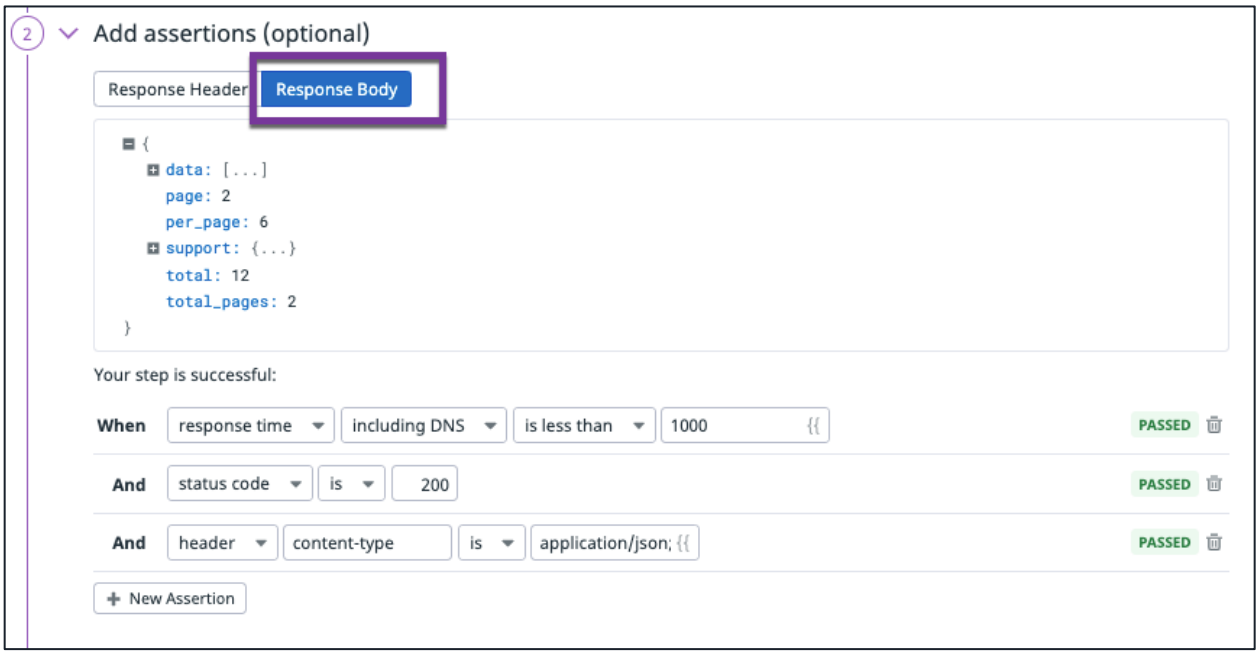
Enter **https://reqres.in/api/users?page=2** as URL and click **Test URL**



You will see the request and response body

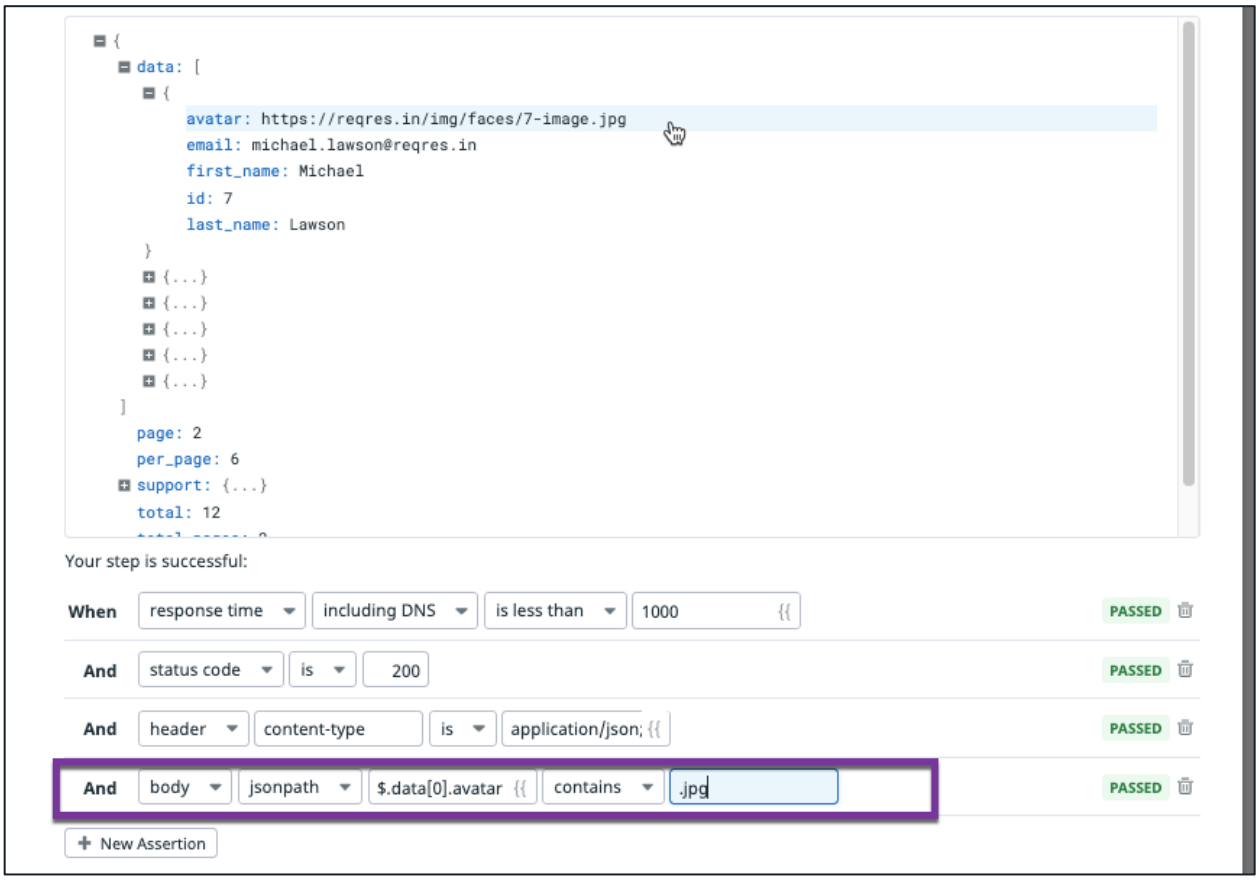


6. Click Response Body tab, you will see the payload of the response and the condition are setup as below (keep it default) and add **New Assertion**

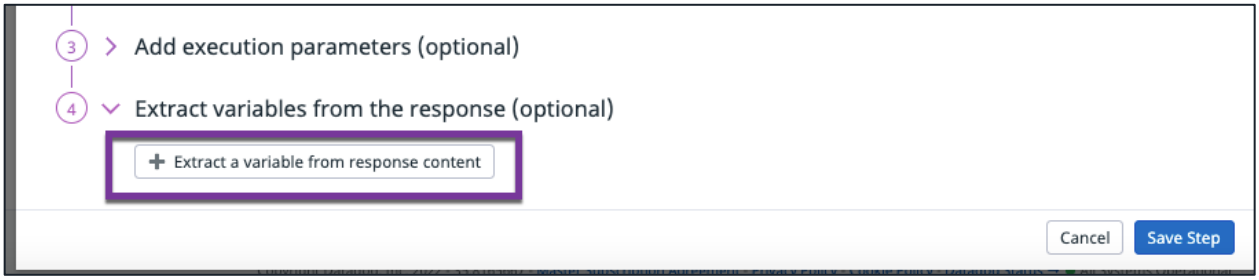


Click the first data array avatar and notice the assertion will automatically created for you.

We will update it to contains **.jpg**



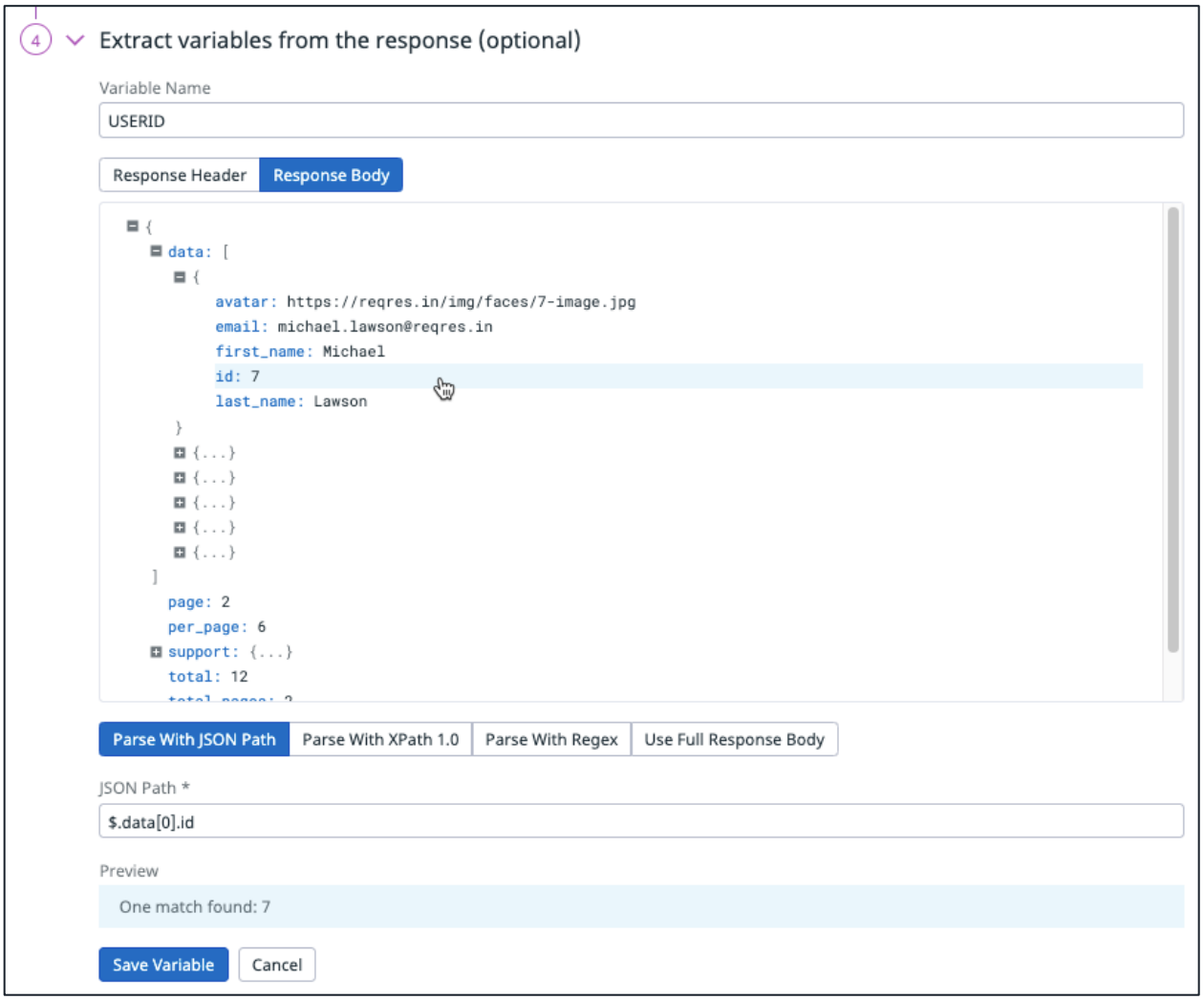
7. Scroll down to click **Extract a variable from response content**



Enter variable name as **USERID** and select the Response Body tab

From here, click on the data array id, notice the preview says **7** which is the id value

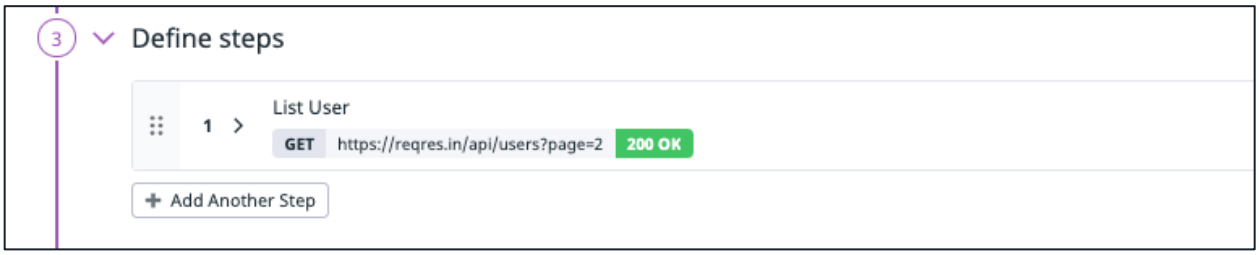
Then, **Save Variable**



8. And then, Save Step

Now you should see the first step is created

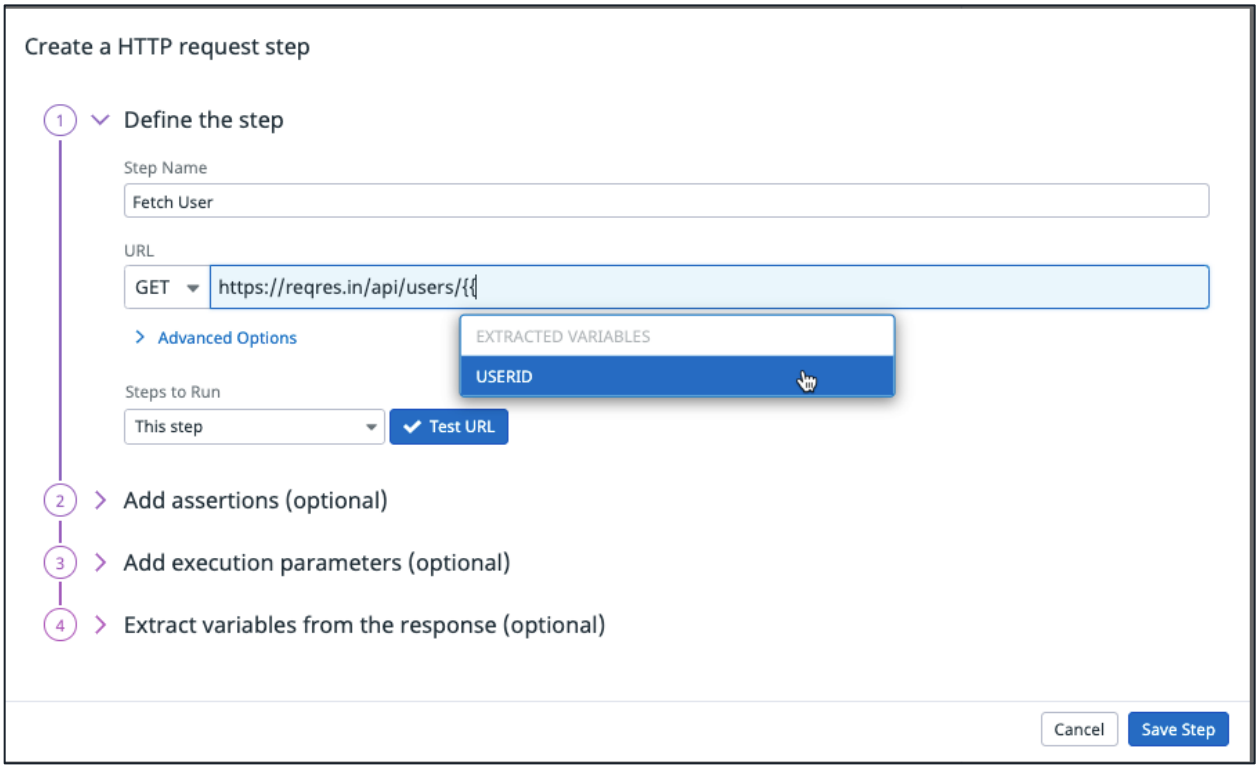
9. Click **Add Another Step**



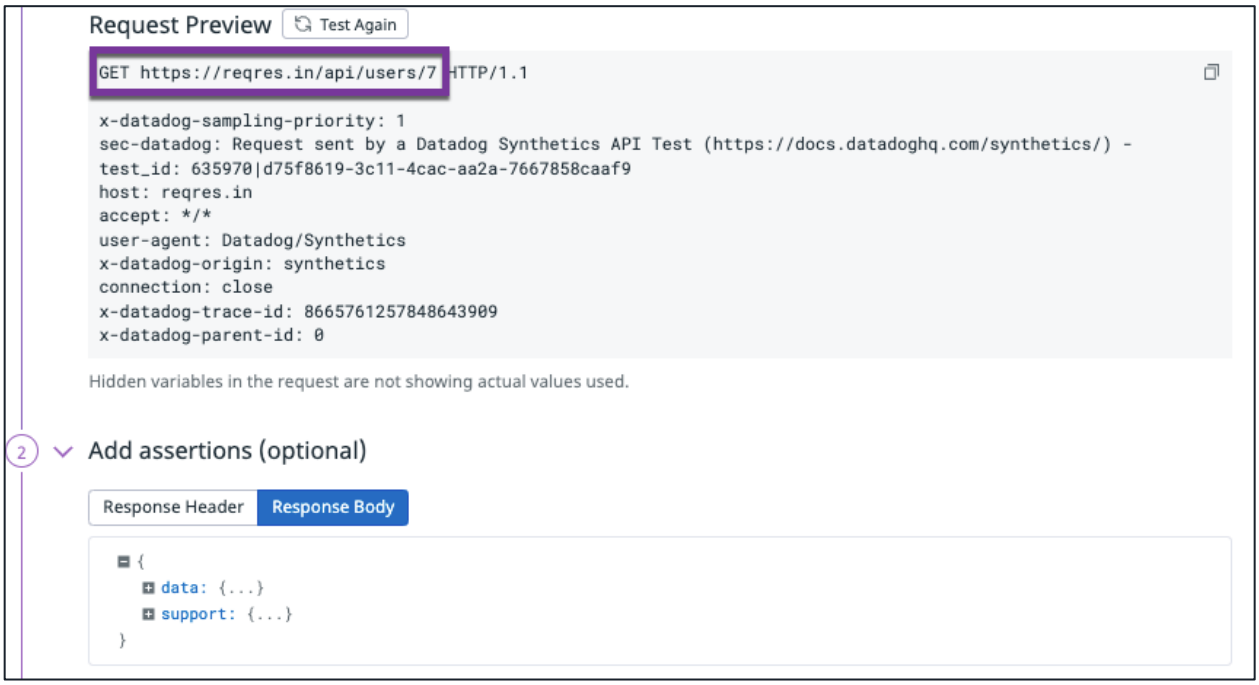
Enter Step Name as **Fetch User**

Input the URL as **https://reqres.in/api/users/** and enter {{ to bring out the variable and select **USERID** (this is the variable that was created in the first step)

Once done, click Test URL

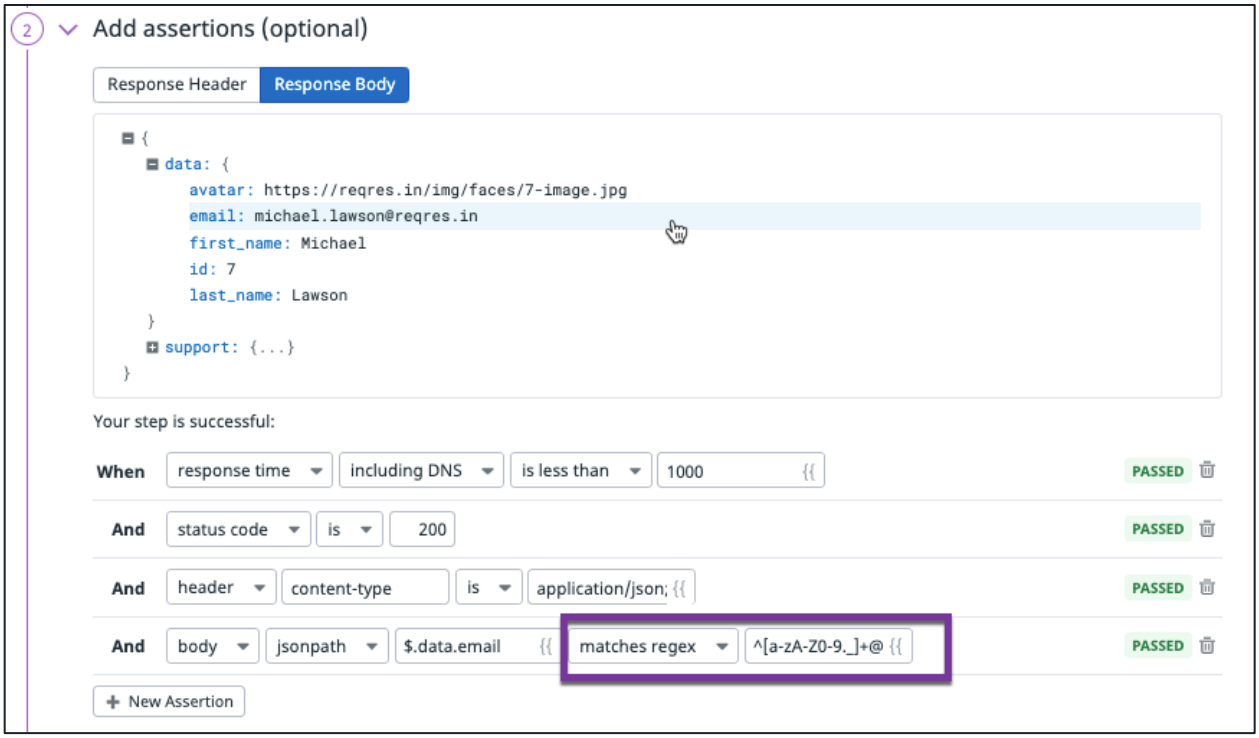


Notice the request preview, the full URL is shown **https://reqres.in/api/users/7**, with the variable **USERID** as **7**

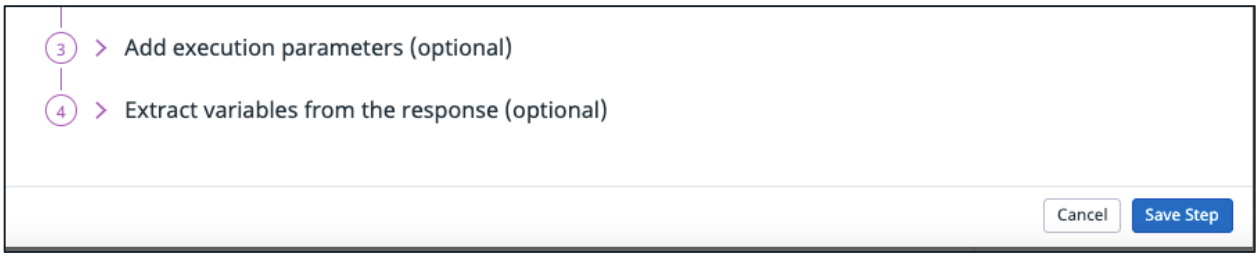


10. Now, let's add addition assertion to ensure that the email match the following regex

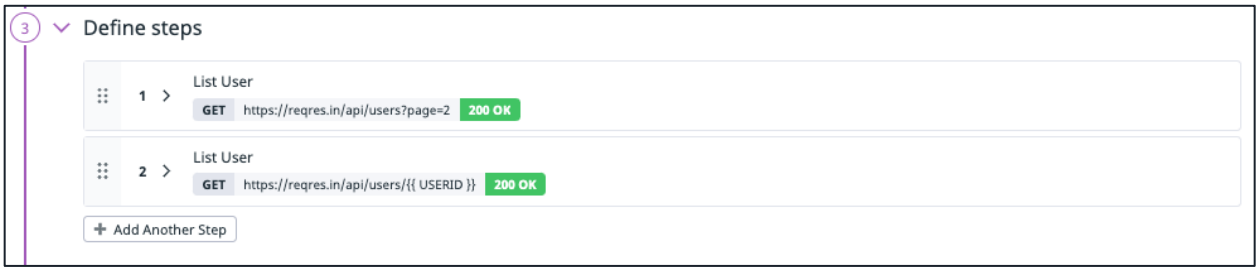
```
^[a-zA-Z0-9._]+@[a-zA-Z0-9.]+\.[a-zA-Z0-9]{2,3}$
```



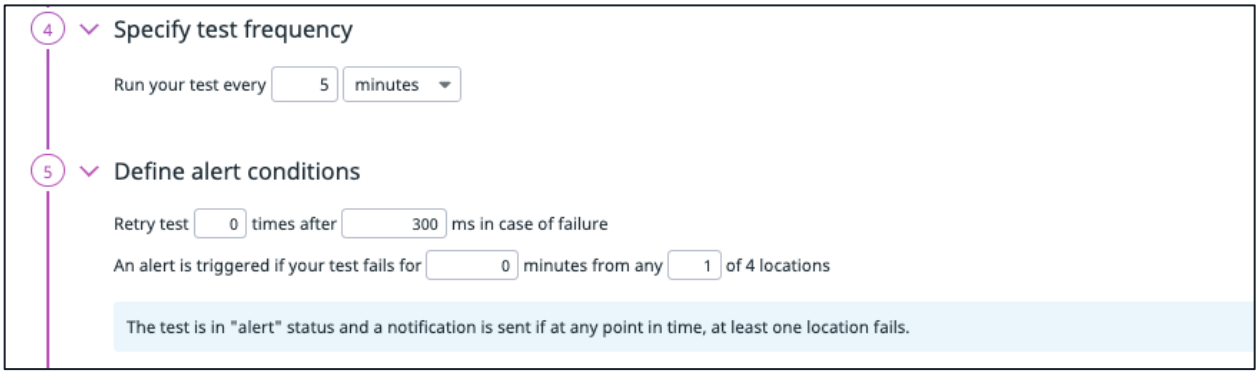
Once done, click **Save Step**



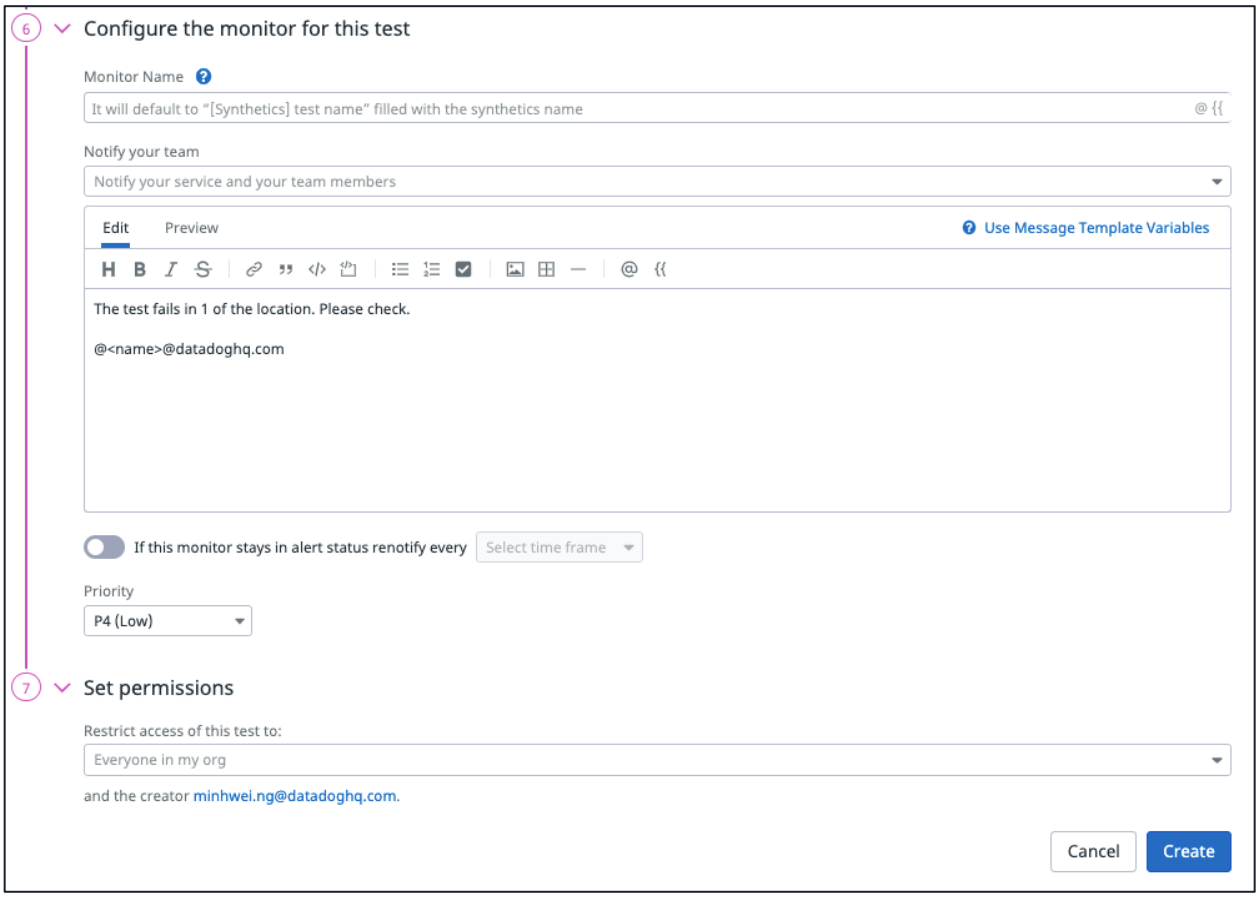
You should see two steps now



11. **Specify the test frequency** and alert conditions

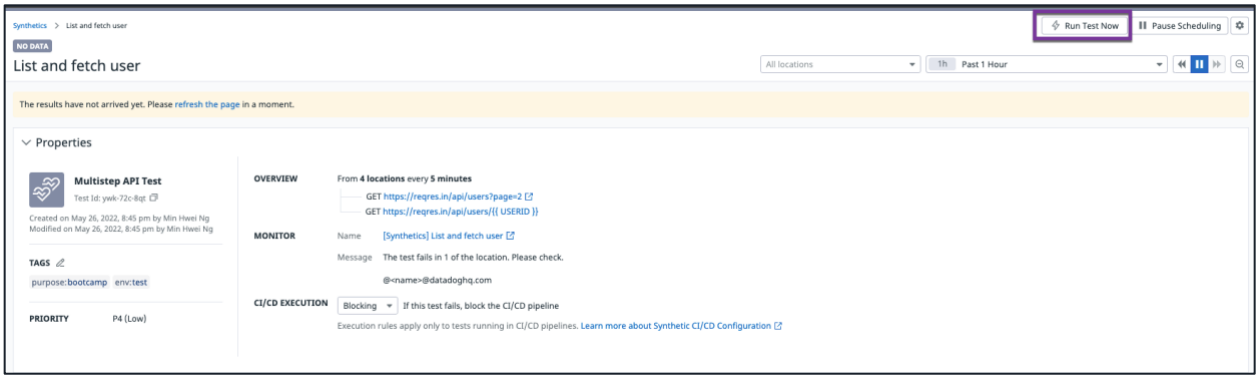


12. Configure the monitor and alert message, set priority to **P4 Low** and **create**

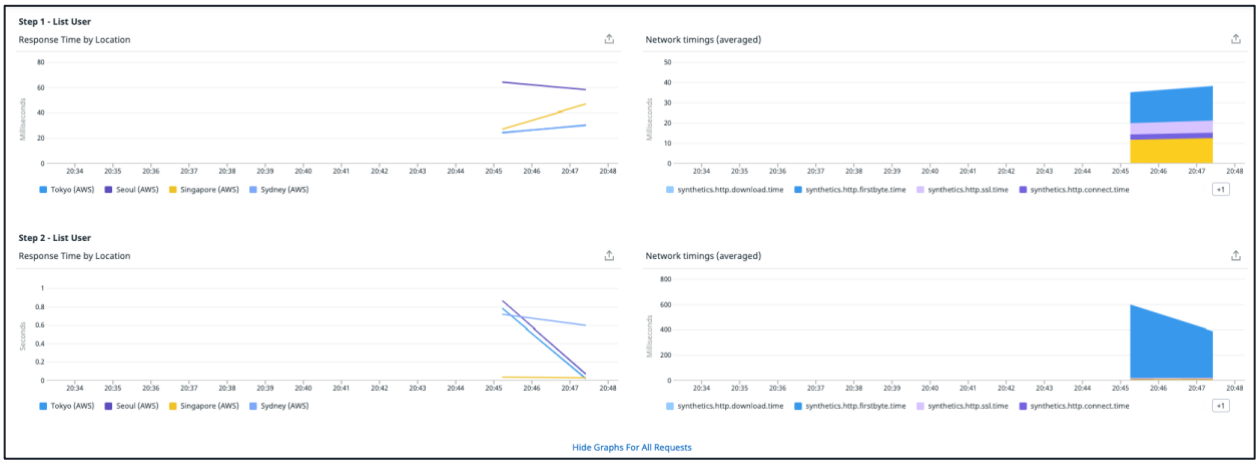


Validation:

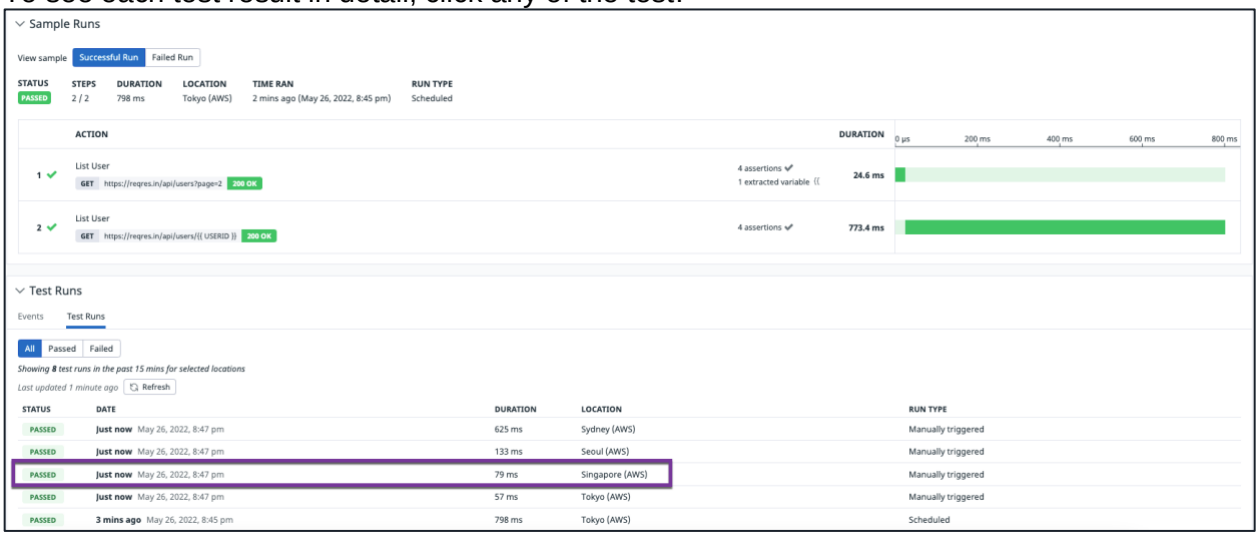
You can click on Run Test Now, or wait for the schedule to kicks in the next 5 minutes



Refresh the page and scroll down to see the details of the test result. You can expand to see all location, and show graphs for all requests

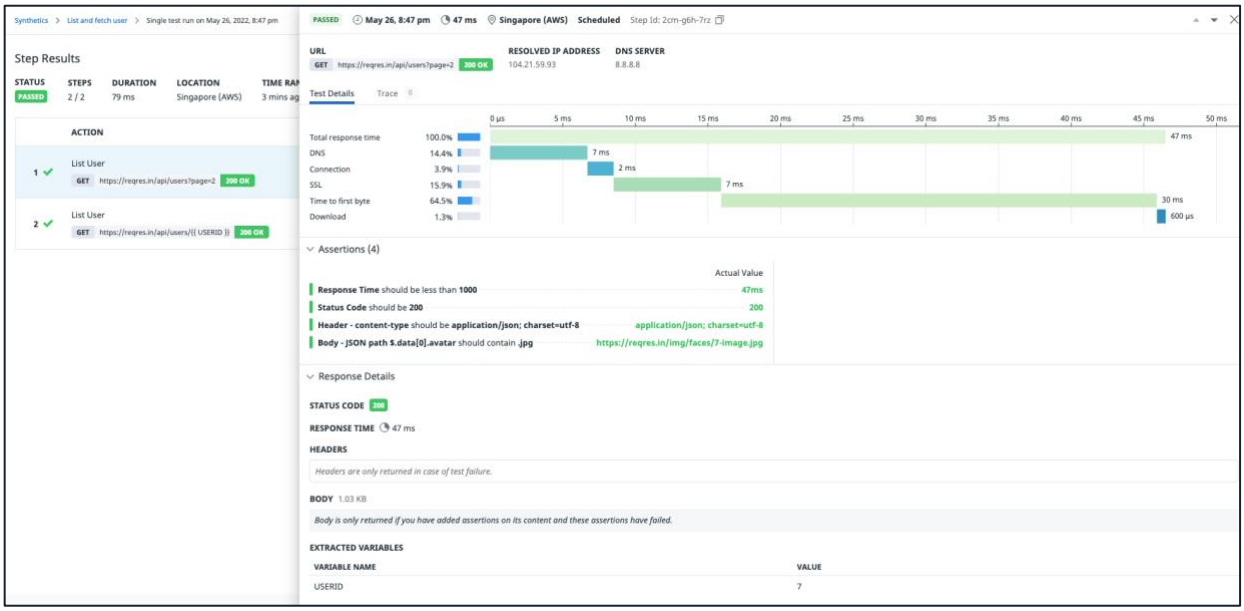


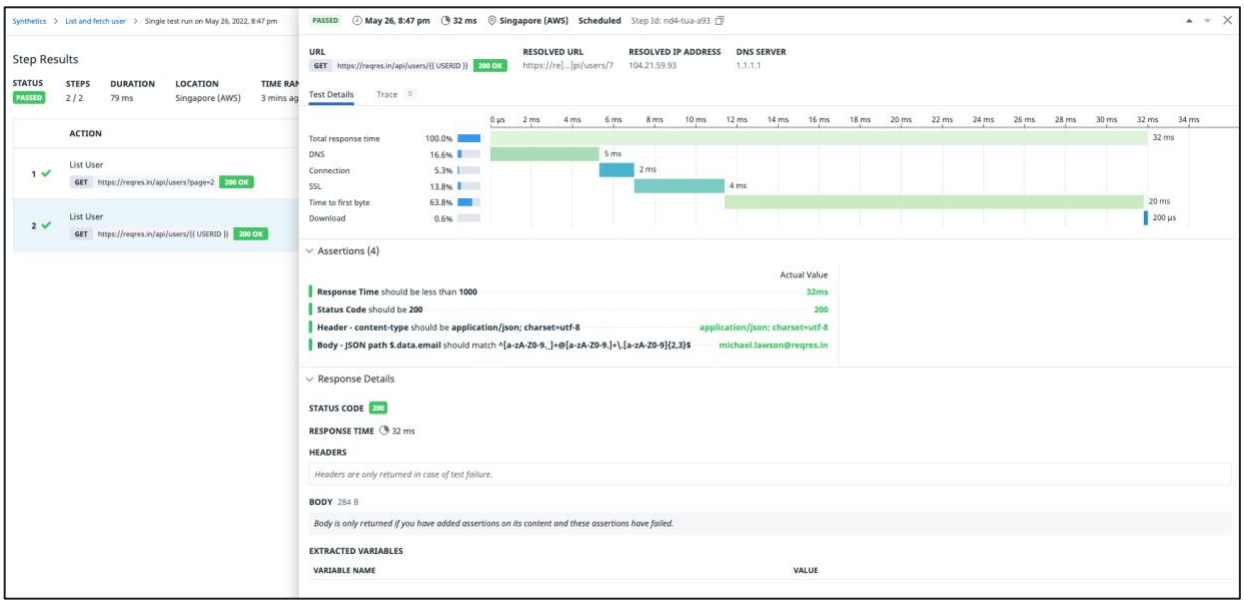
To see each test result in detail, click any of the test:



It will show you the step results of the test:

As you click on each of this test step, the side panel show more insight into the result:





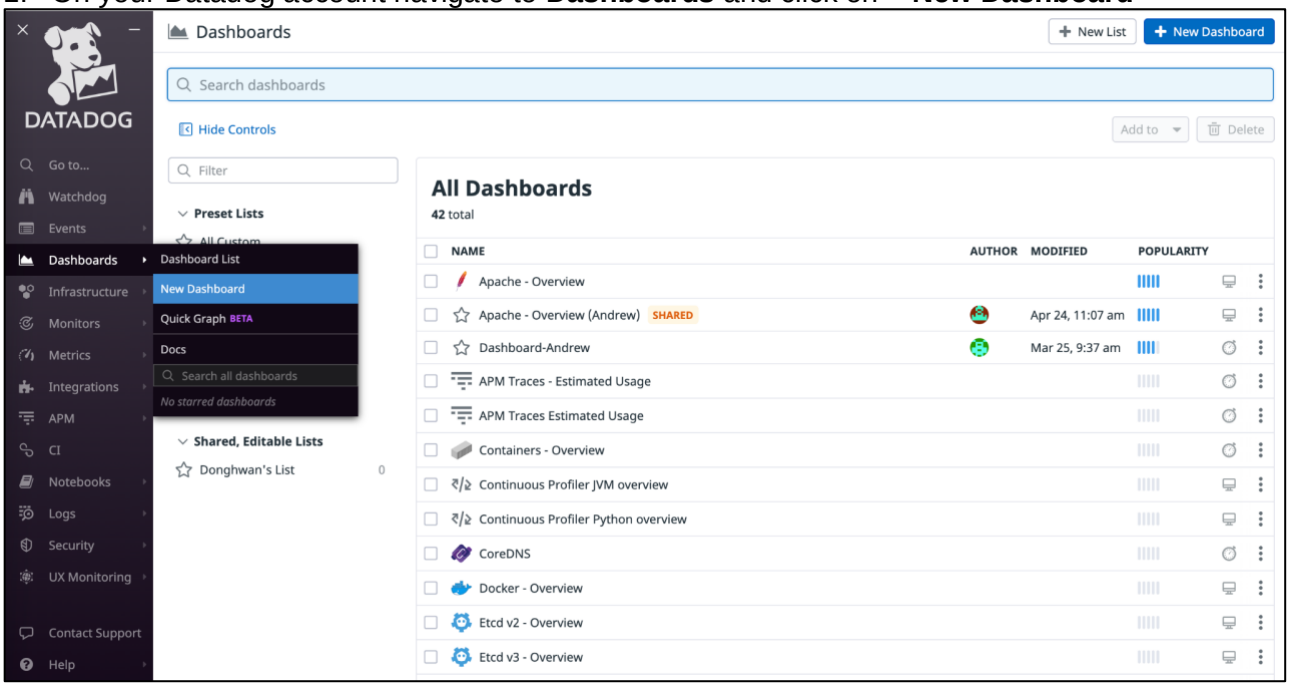
108 Dashboards Lab

A dashboard is Datadog's tool for visually tracking, analysing, and displaying key performance metrics, which enable you to monitor the health of your infrastructure.

108-1 Create Dashboard

In this lab we will learn to create dashboard and explore some of the widgets.

1. On your Datadog account navigate to **Dashboards** and click on + **New Dashboard**



2. Enter a dashboard name and choose a layout option. In this lab we will use the **New Dashboard,** Click **New Dashboard**.

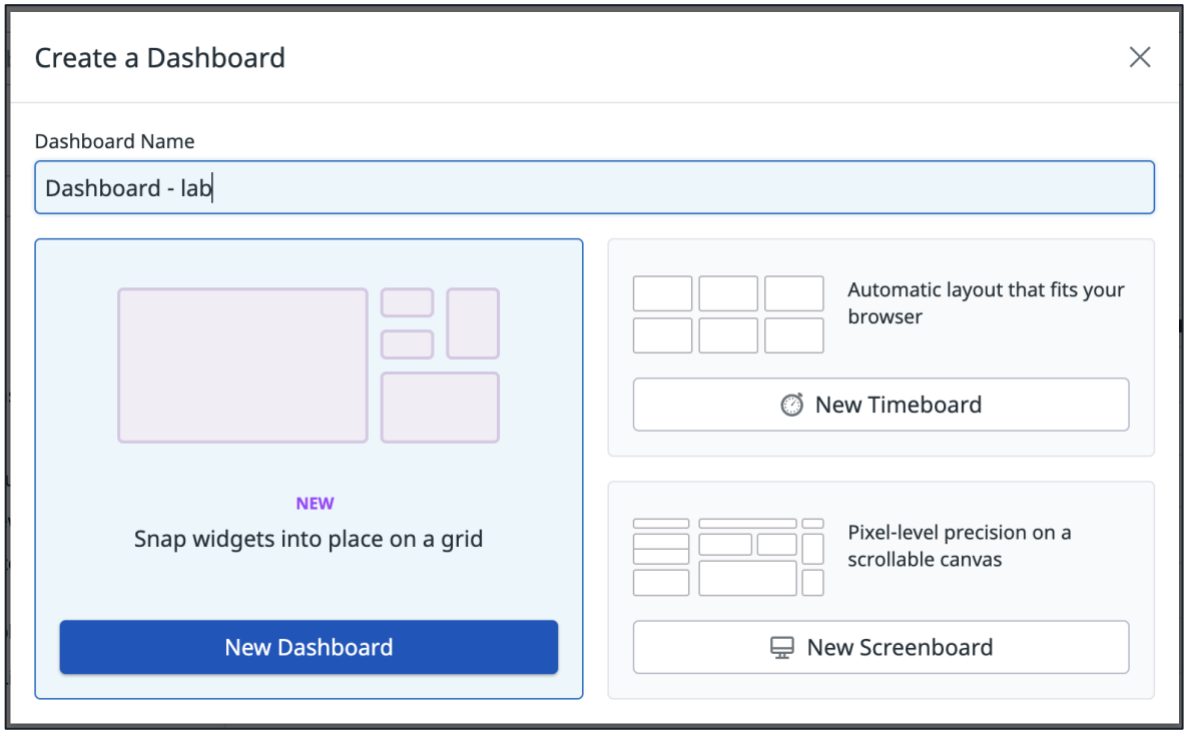
Dashboards
Dashboards are on a grid-based layout, which can include a variety of objects such as images, graphs, and logs. They are commonly used as status boards or storytelling view, which update in real-time and can represent fixed points in the past. They also work well for debugging.

Timeboards
Timeboards have automatic layouts and represent a single point in time—either fixed or real-time—across the entire dashboard. They are commonly used for troubleshooting, correlation, and general data exploration.

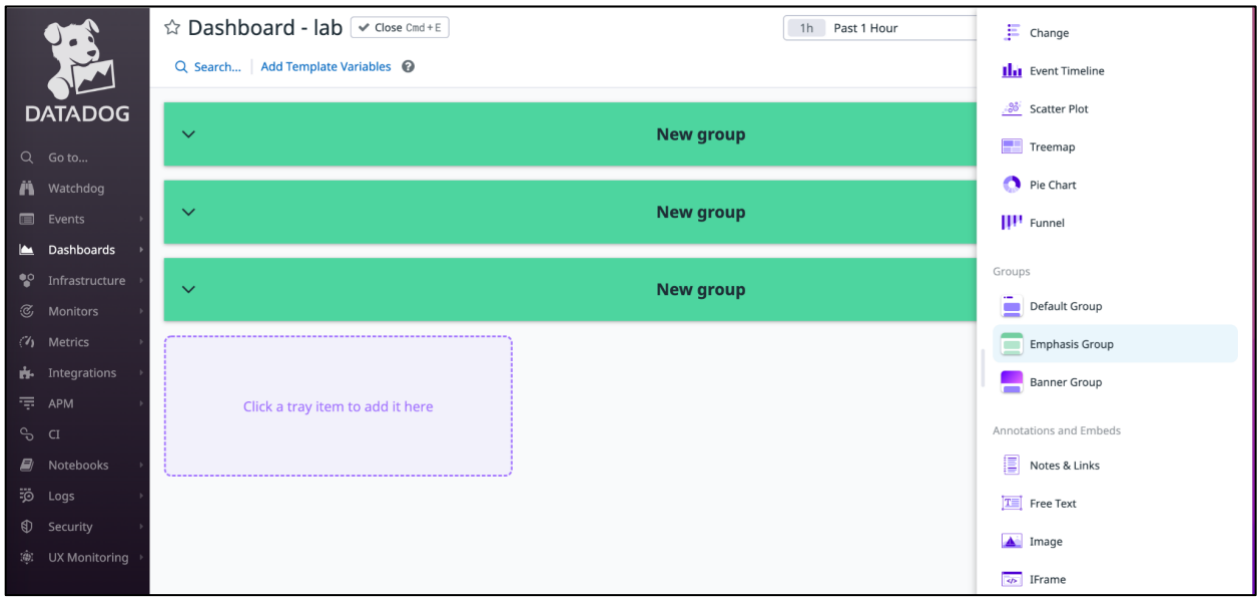
Screenboards

Screenboards are dashboards with free-form layouts which can include a variety of objects such as images, graphs, and logs. They are commonly used as status boards or storytelling views that update in real-time or represent fixed points in the past.

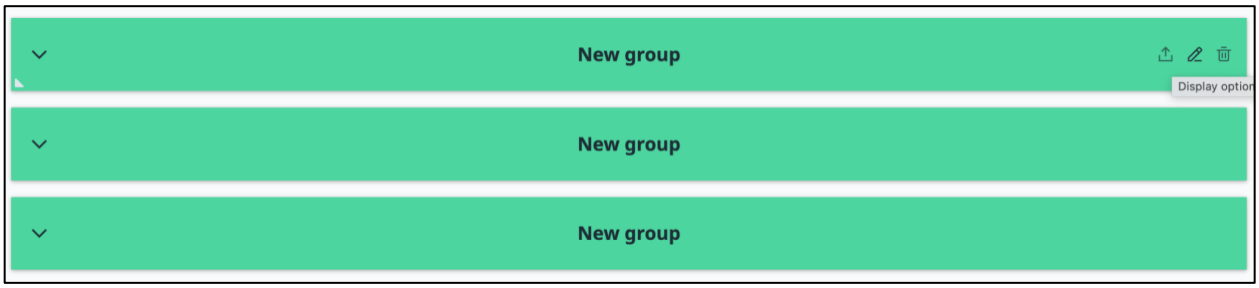
More details can be found here: https://docs.datadoghq.com/dashboards/



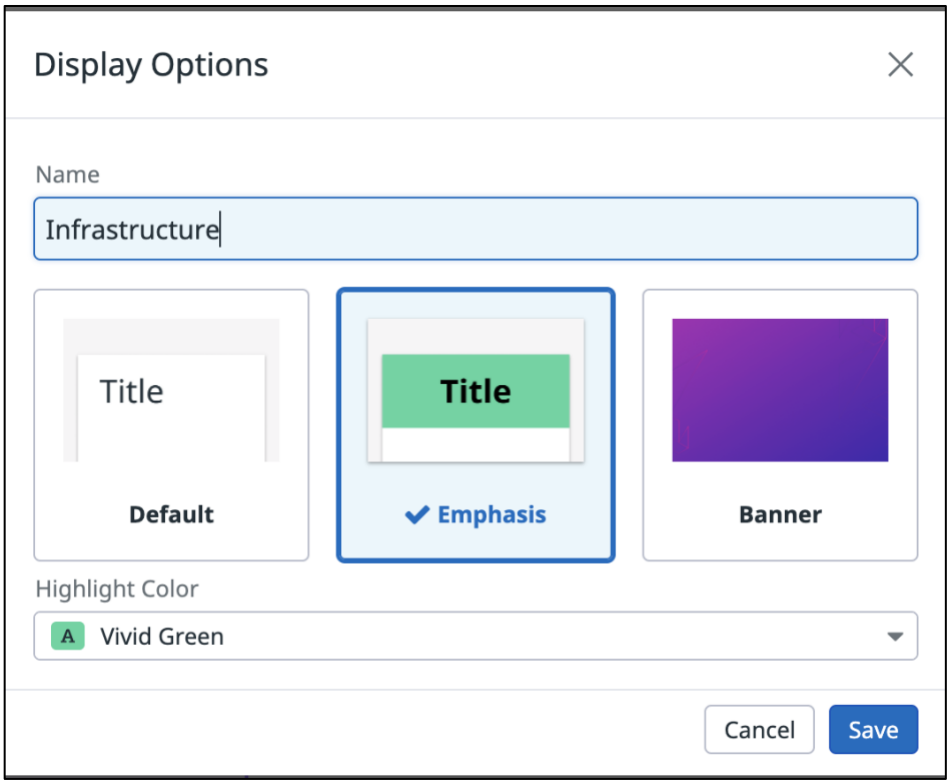
3. From the list of widget options in the **Add Widgets** left side panel, select **Emphasis Group** under the **Groups** sub-category of widgets. Add **three** different **Emphasis Group**.



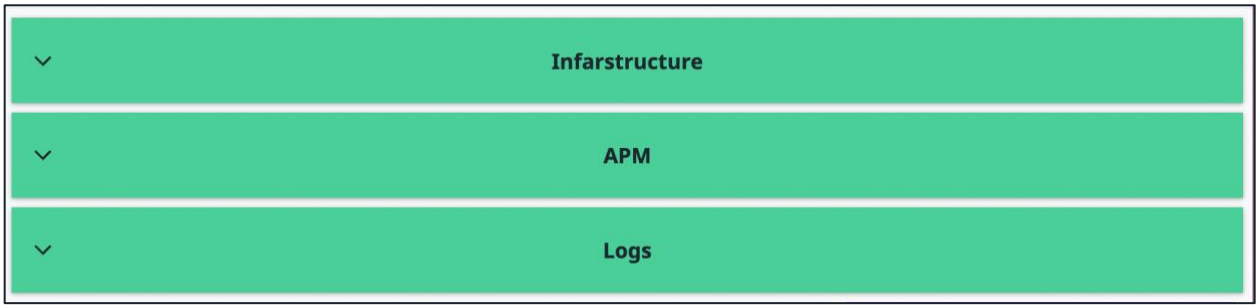
4. On the newly added **Emphasis Group** widget on the dashboard, click the **Edit** button as shown in the below screenshot to give name to one of the **Emphasis Group**.



5. Provide the **Name** as **Infrastructure** and click **Save**.



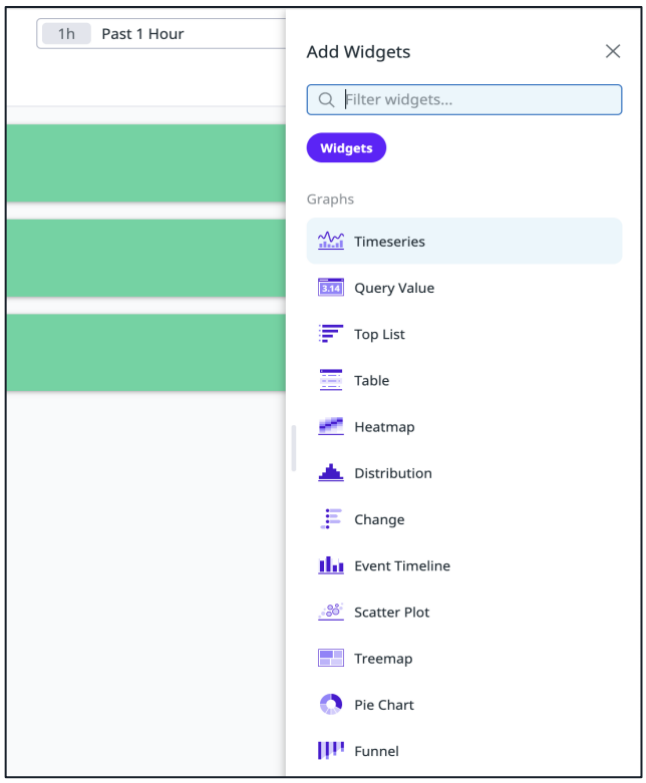
6. Repeat steps #4 and #5 to name the other 2 **Emphasis Group** as **APM and Logs.** The outcome should look as below:



As a next steps we will be adding some Infrastructure, APM and Logs related widgets by copying a preset widget, creating a new one, and finally we will layout the dashboard nicely.

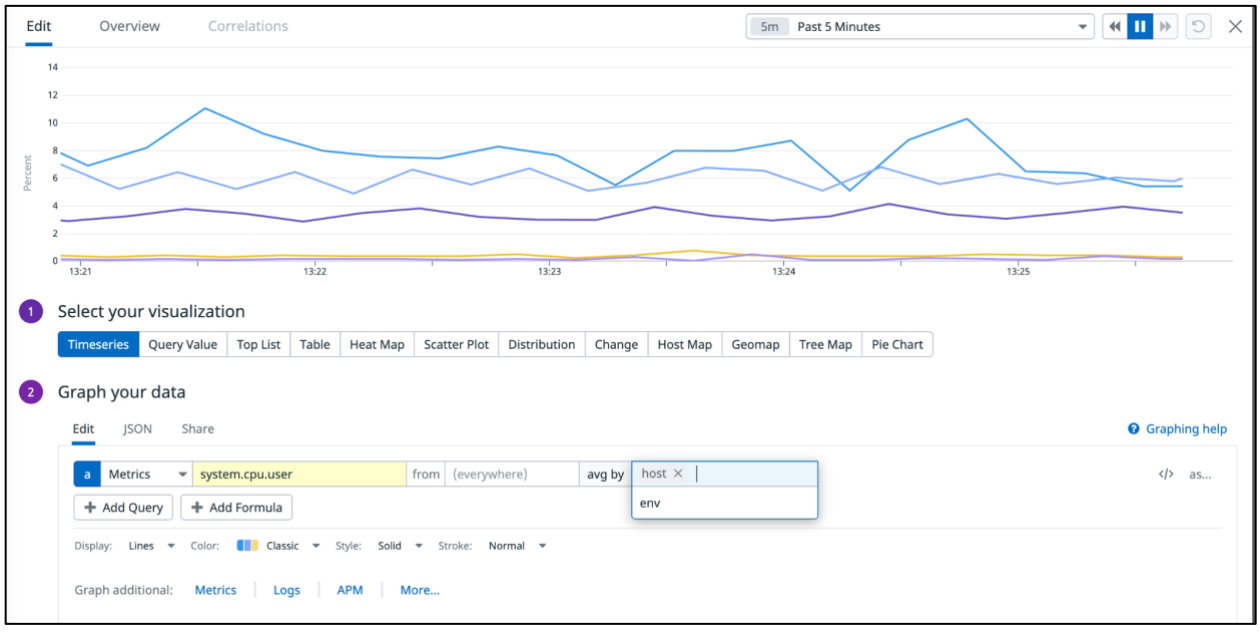
Create Widgets:

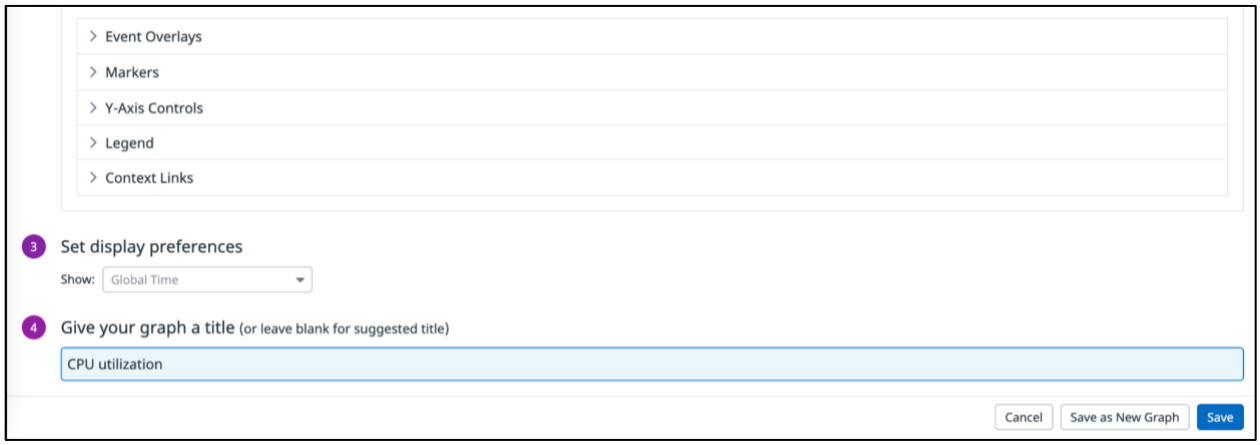
7. Click **Add Widgets** and select **Timeseries**



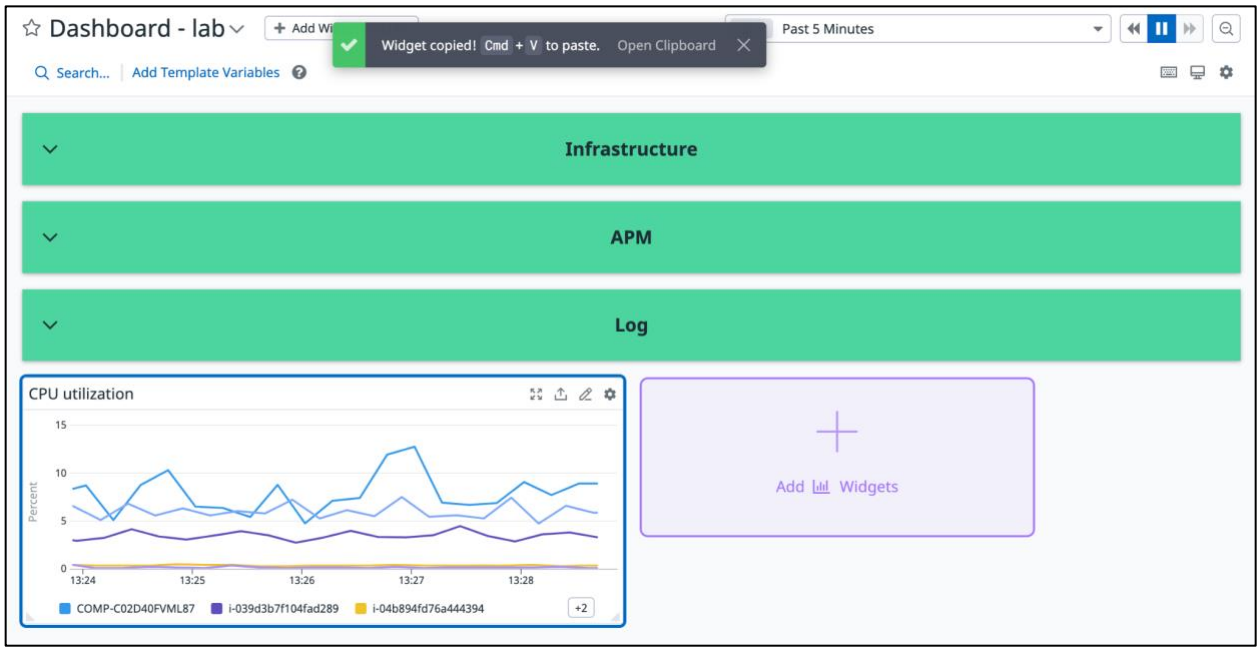
8. In this timeseries widgets we will add CPU utilization metric for all hosts reporting to your Datadog account. Under **2. Graph your data** Add
   Metrics: **system.cpu.user**
   Avg by **host**

   Under **4. Give your graph a title** provide the title as **CPU Utilization**.

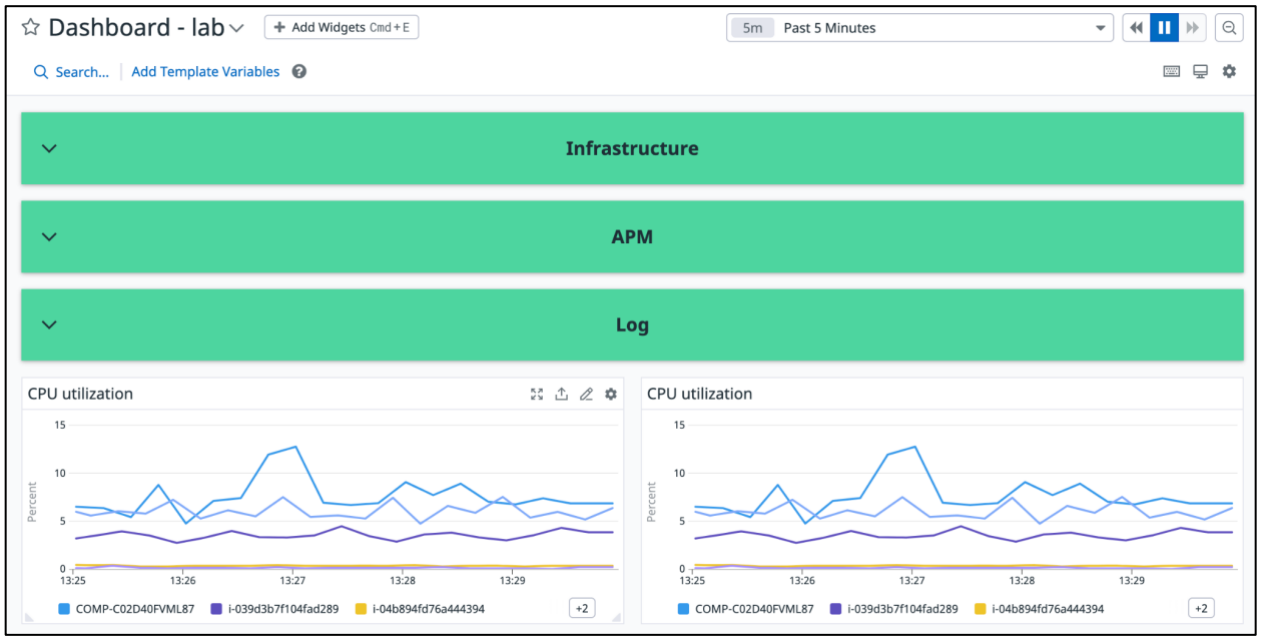




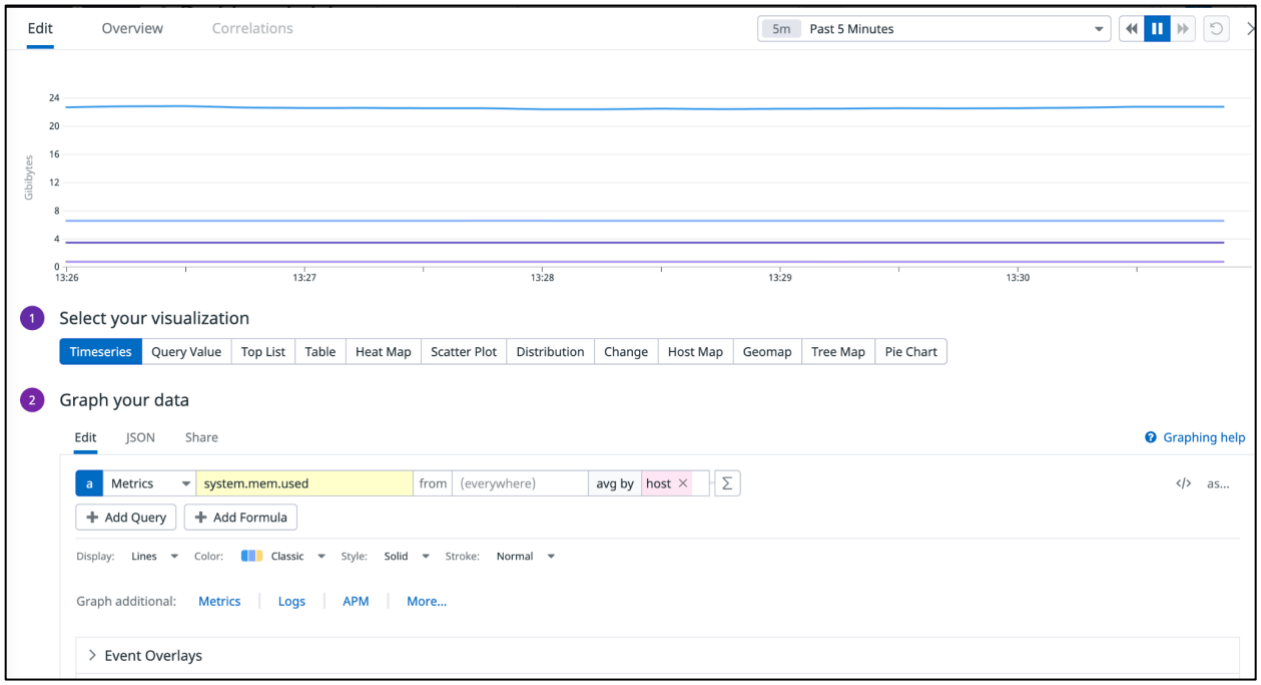
9. We will use the newly created **CPU Utilization** timeseries widgets to create a **Memory Utilization** by Copy/Paste. Click on **CPU Utilization** widget and press **Cmd/Ctrl+C** on your keyboard.



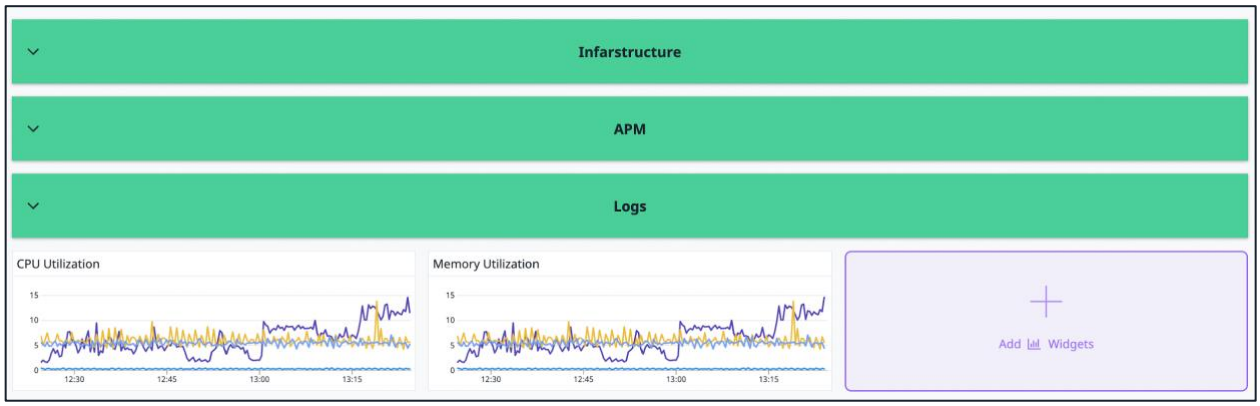
10. Press **Cmd/Ctrl+V** to paste the widget you copied. The newly copied widget appears at the bottom of the dashboard.



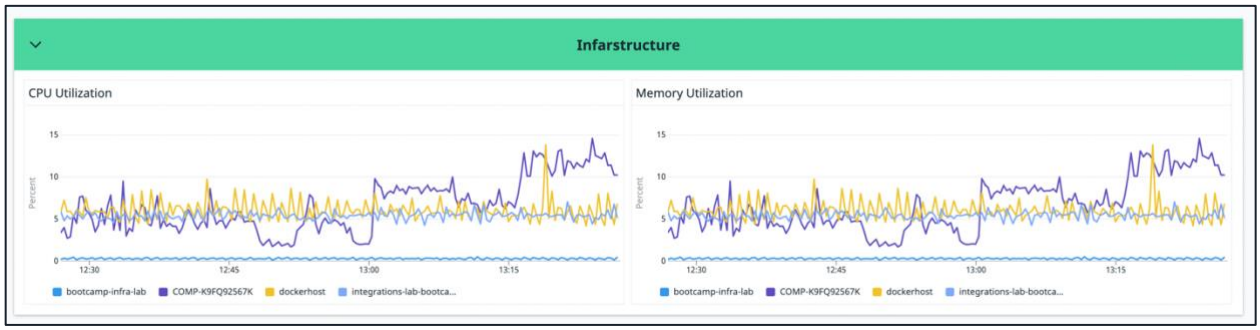
11. Now let's edit one of the **CPU Utilization** widgets by clicking on the edit icon on the widget. On the widget edit screen, change the Metrics from **system.cpu.user** to **system.mem.used** and title from **CPU Utilization** to **Memory utilization**.



12. Now we have **CPU Utilization** and **Memory Utilization** widgets added to our dashboard.



13. Now, let's layout the dashboard nicely by dragging and dropping the **CPU utilization** and **Memory Utilization** widget under the **Infrastructure Emphasis Group** widget

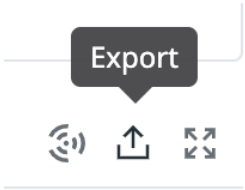


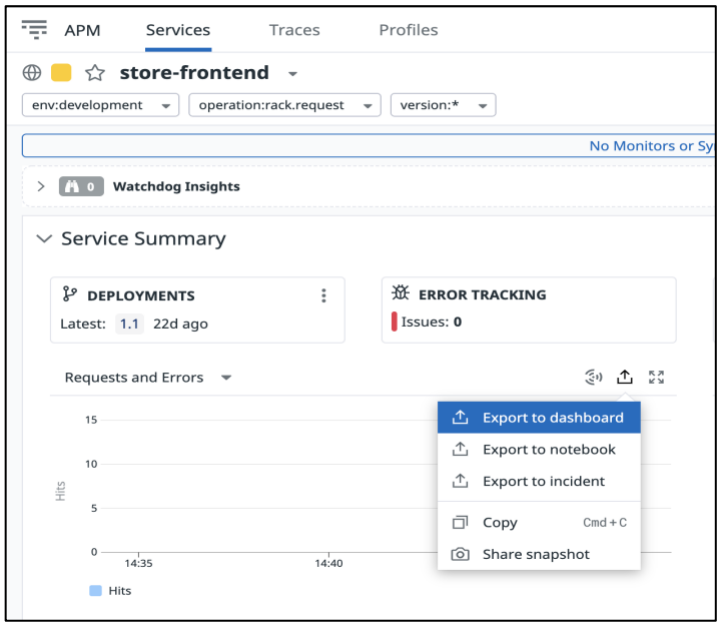
Copy a pre-set Widget:

You might have already noticed that throughout the Infrastructure, APM, Logs and other monitoring products; Datadog provides OOTB graphs and other useful widgets which could be copied directly to your custom dashboards without having to go through the hassle of creating new ones. Let's explore in our next steps of this lab for how to go about it:

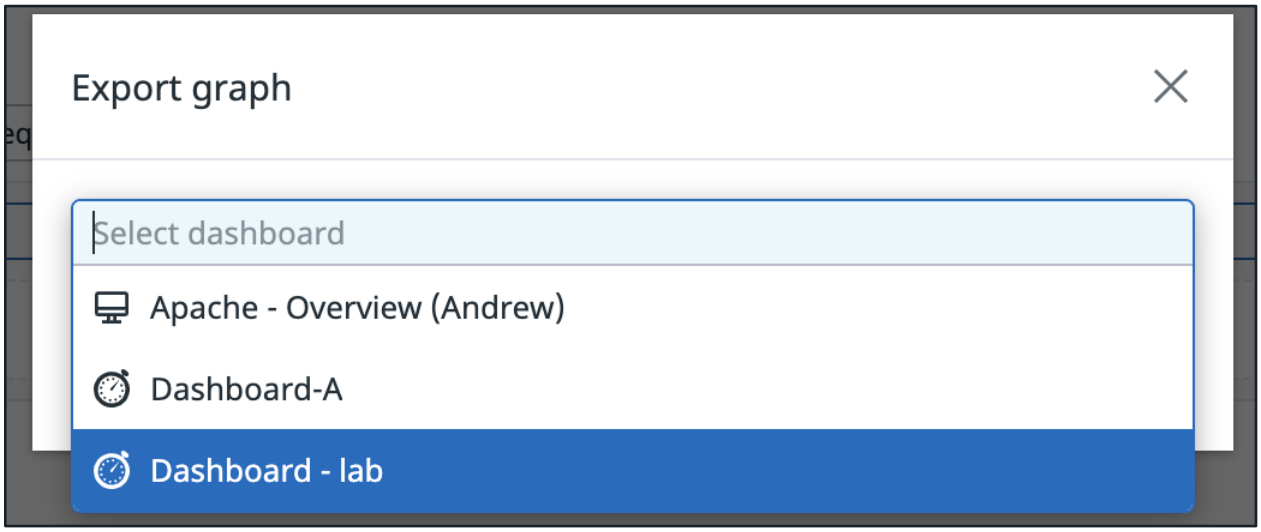
14. We will be adding some APM metrics to this Dashboard. Open your Datadog account in a new browser tab and navigate to **APM -> Services** and click on **store-frontend**. The **store-frontend** service overview page is loaded, you would notice that under **Service Summary** we have valuable graphs which can be directly added to our custom dashboard.

    On the **Request and Errors and Latency** graph widget Click **Export** button and select **Export to dashboard**

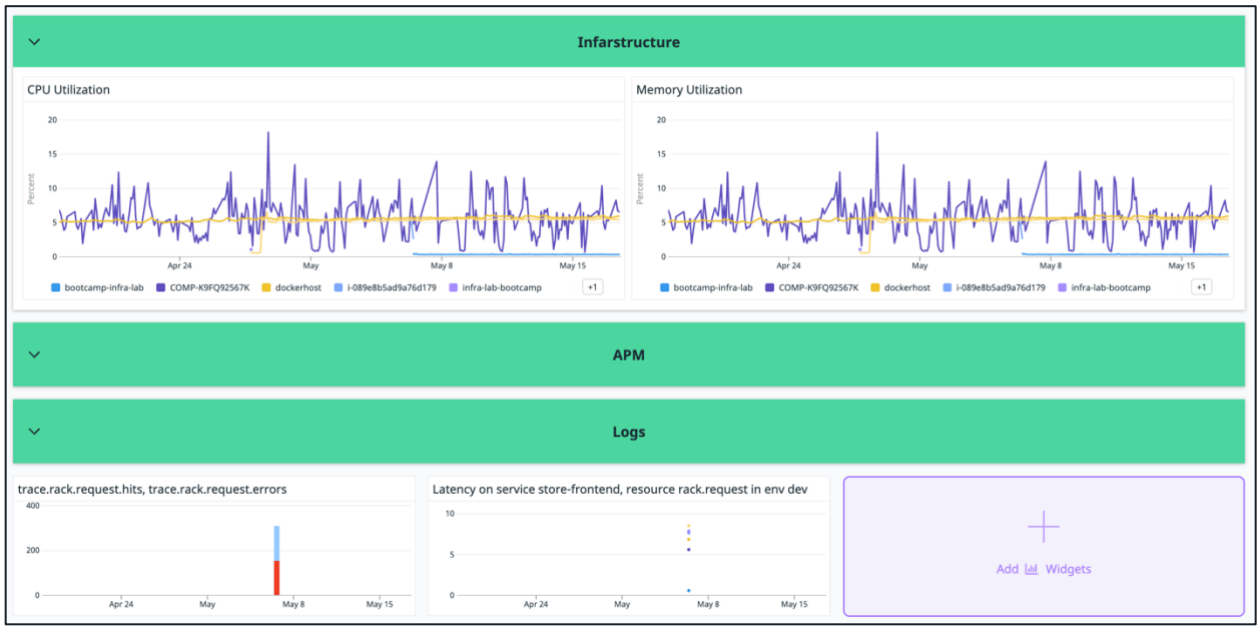




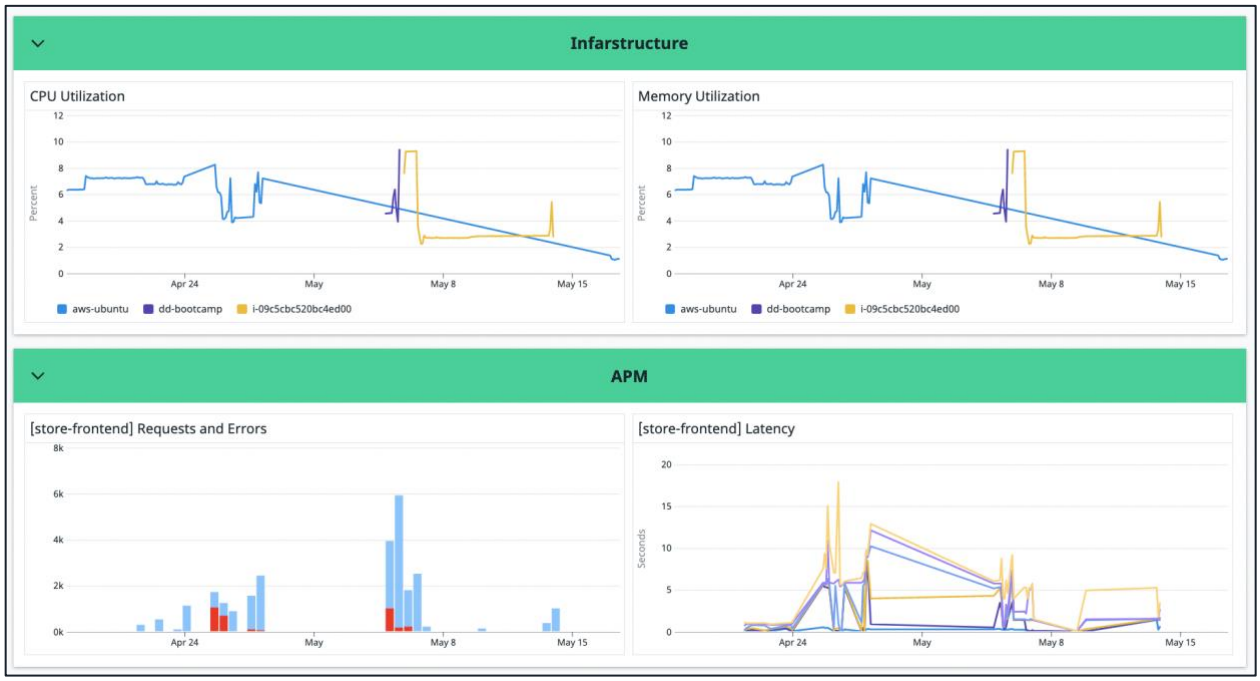
15. On the **Export graph** pop-up, select **Dashboard - lab** from the drop-down list



16. Refresh the browser tab you were using to create the dashboard **Dashboard – lab.** Notice that **Request and Errors and Latency** for **store-frontend** is added at the bottom of the dashboard.

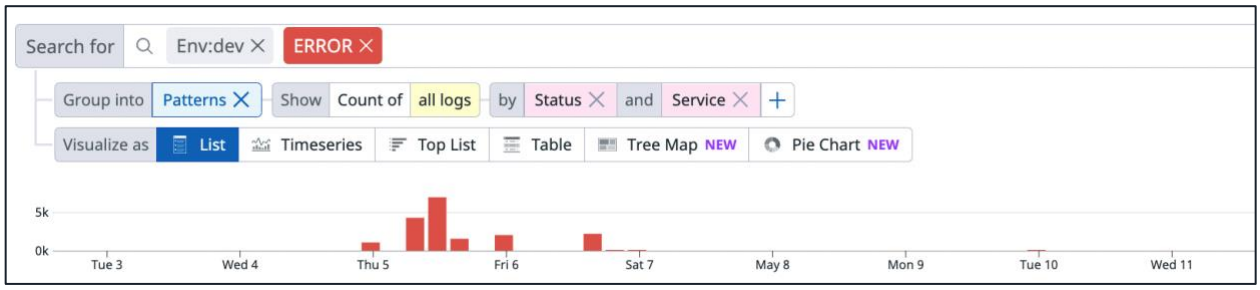


17. Edit the graph title to **[store-frontend] Requests and Errors** and **[store-frontend] Latency**. Finally, drag and drop the widget to **APM Emphasis Group** widget

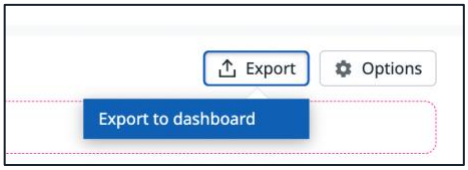


Now that we have added some widgets to our Infrastructure and APM emphasis group, let's learn to add some Logs related information to our dashboard.

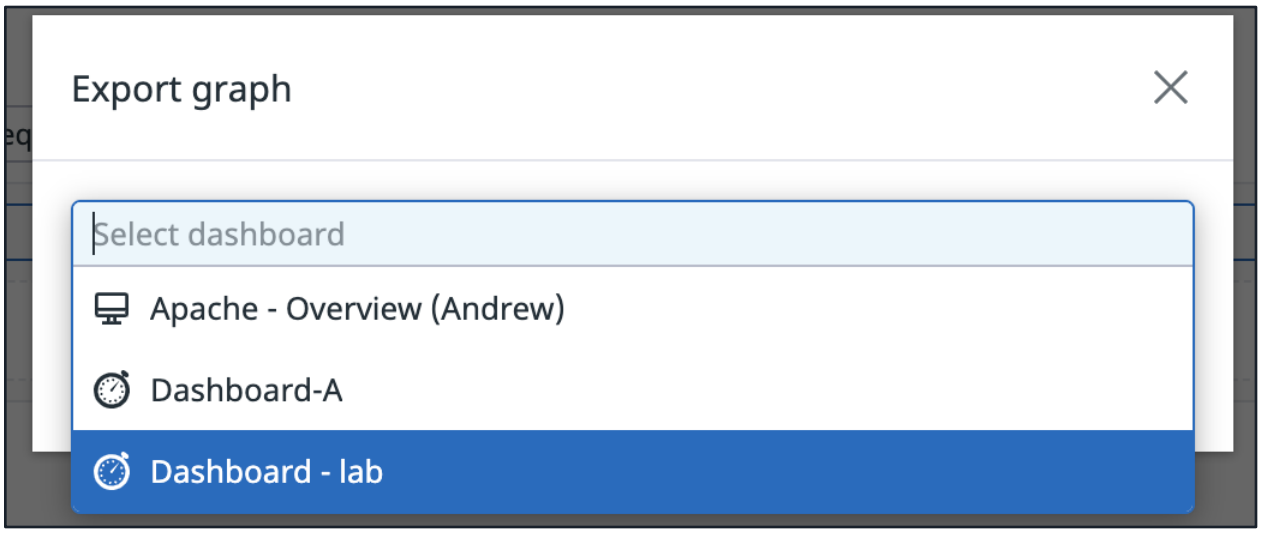
18. Open your Datadog account in a new browser tab and navigate to **Logs -> Patterns**. In the **Search for** section on the top, type in **env:dev status:error** or use the facets to make the appropriate selection.



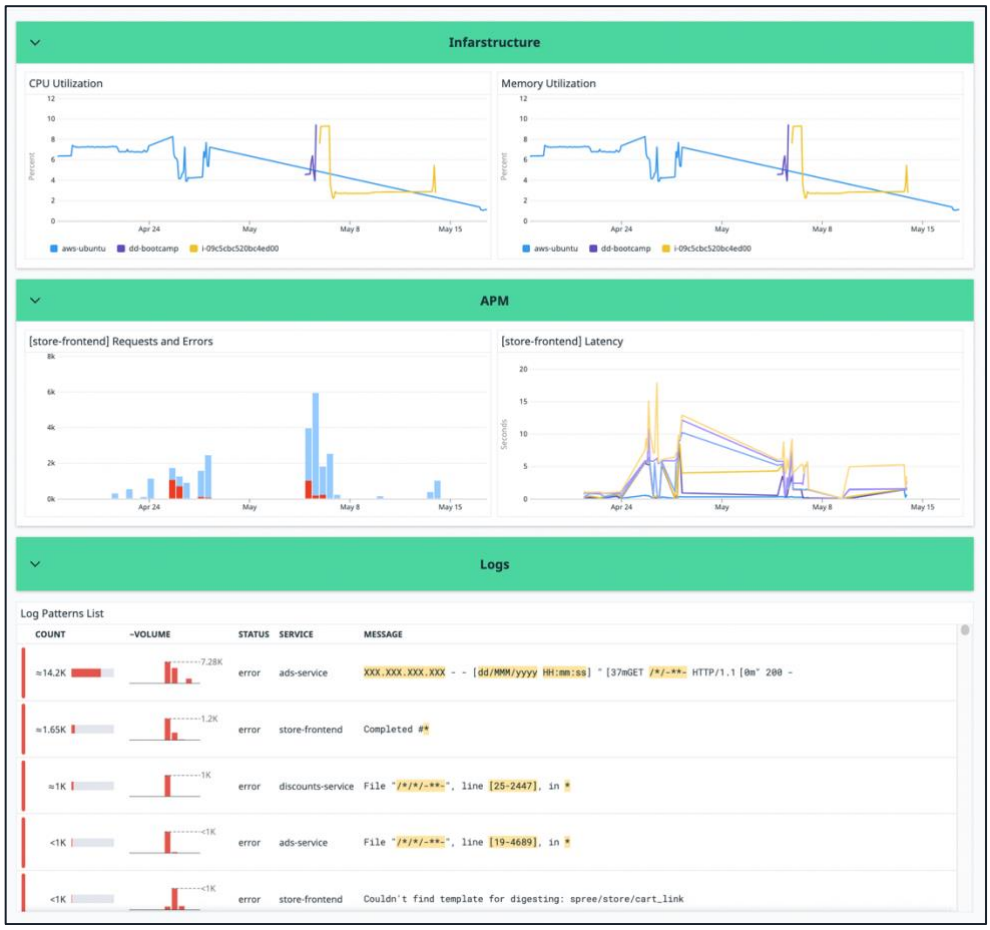
19. Click on the **Export** button and select **Export to dashboard**



20. On the **Export graph** pop-up, select **Dashboard - lab** from the drop-down list



21. Refresh the browser tab you were using to create the dashboard **Dashboard – lab.** Notice that **Log Patterns List** is added at the bottom of the dashboard. Finally, drag and drop the widget to **Log Emphasis Group** widget. The newly created dashboard looks as below



This may not be the best designed dashboard, but it's a good start to iterate on. You can update and clone dashboards, and you can export them as JSON to store on your hard drive or in a Git repository.

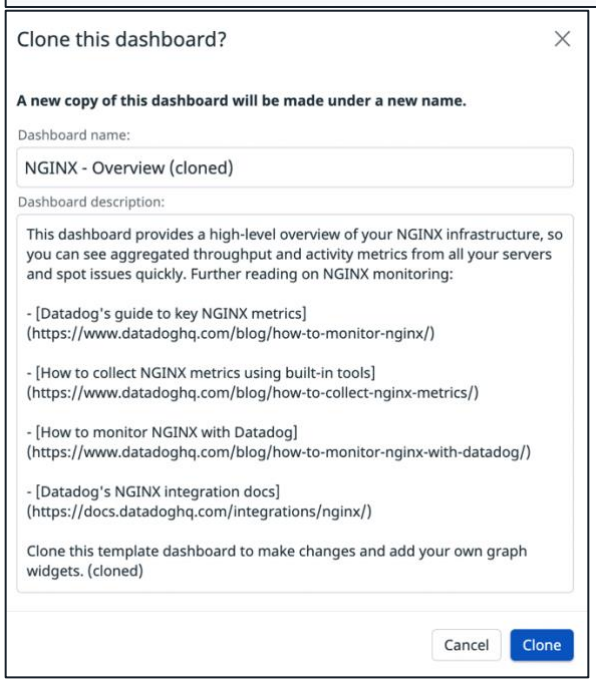
108-2 Import an Existing Dashboard

It is important to note that often you will create a best practice dashboard by adding various graphs, widgets etc. for a particular use case or in a non-production environment and for different personas. With export and import features you can reuse an existing best practice dashboard with minimal changes in the same or different Datadog platform. This lab will help you learn about how to import an existing best practice dashboard for the e-commerce StoreDog application.

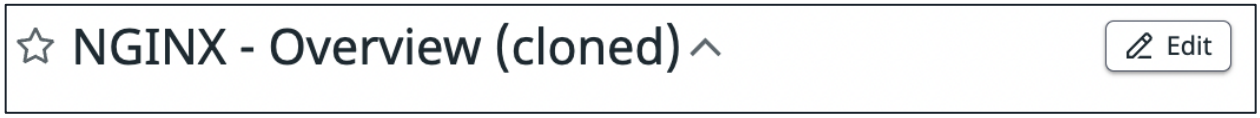
1. On your Datadog account navigate to **Dashboards** and click on **New Dashboard**
2. Give the dashboard name as **[Bootcamp] Observability Dashboard** and click on **New Dashboard** button
3. From the right corner click on gear box 
4. Select **Import dashboard JSON..**
5. **Drag and drop or browse** to import the dashboard JSON. Note: You can download the dashboard JSON file from [G-drive link: Ask your instructor]
6. Hit the **Yes, Replace** button. You will notice that the entire dashboard is imported within seconds.
7. Explore the dashboard which can be your one-stop-shop for understanding the overall health of the e-commerce StoreDog application from Infrastructure to APM to Logs to RUM.

108-3 Integrations Dashboard

1. On your Datadog account navigate to **Dashboards** and click on **Dashboards List**
2. From your left **Preset Lists** click on **All Integrations**
3. In the search panel type ***NGINX – Overview.*** Click on **NGINX – Overview** to open the dashboard. This is OOTB dashboard Datadog provides for Nginx integrations.
4. Click **Clone Dashboard** located on the top,



5. Edit the cloned dashboard



6. Name the dashboard name as ***[Storedog] NGINX – Overview*** Notice that along with OOTB Integration dashboard, Datadog also provide some monitoring tips

This exercise was to help you understand that Datadog provides OOTB Integration Dashboard as a starting point. This encourages reusability by cloning and modifying the dashboards based on your requirements. The possibilities are numerous and will be left as open exploration, however few of the things that you may want to do:

a. Merge multiple Integrations dashboard into ONE, like a one-stop-shop
b. You may want to bring together multiple integrations metrics/graphs/etc. established for your applications onto 1 dashboard along with Metrics, Traces and Logs. Which could be like a full stack observability dashboard.
c. High level monitor summary dashboard for all Integrations
d. Etc.

109 Monitors and SLOs

As we discussed most of our customers have thousands or tens of thousands of metrics to deal with. There are hundreds of integrations that have been created to help you monitor nearly every aspect of your enterprise. All this data can be a bit overwhelming; how do you process them all? Which ones do you add to your monitors?

The first step is to break the metrics down in to a few buckets or categories - three areas of metrics. Knowing about these three types of metrics will help you construct monitors. Create monitors of sets of these metrics in ways that are useful to you and contain information you need to know.

Think first about the questions you will have or will be getting about your environment and metrics. Formulate those questions, and then think of the metrics that you would gather together for creating proactive monitors, so you can get the answers to those question.

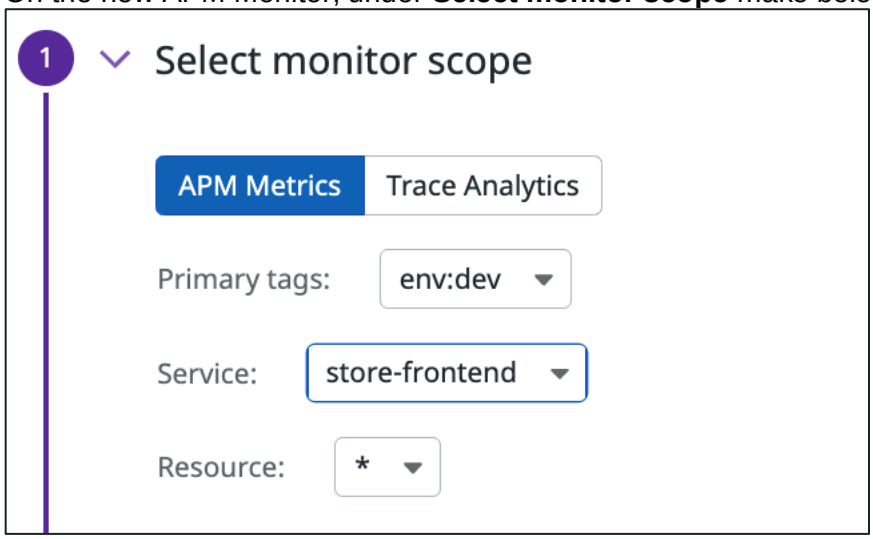
In this lab we will learn to create some Work and Resource metrics monitors.

109-1 Create Work Monitors

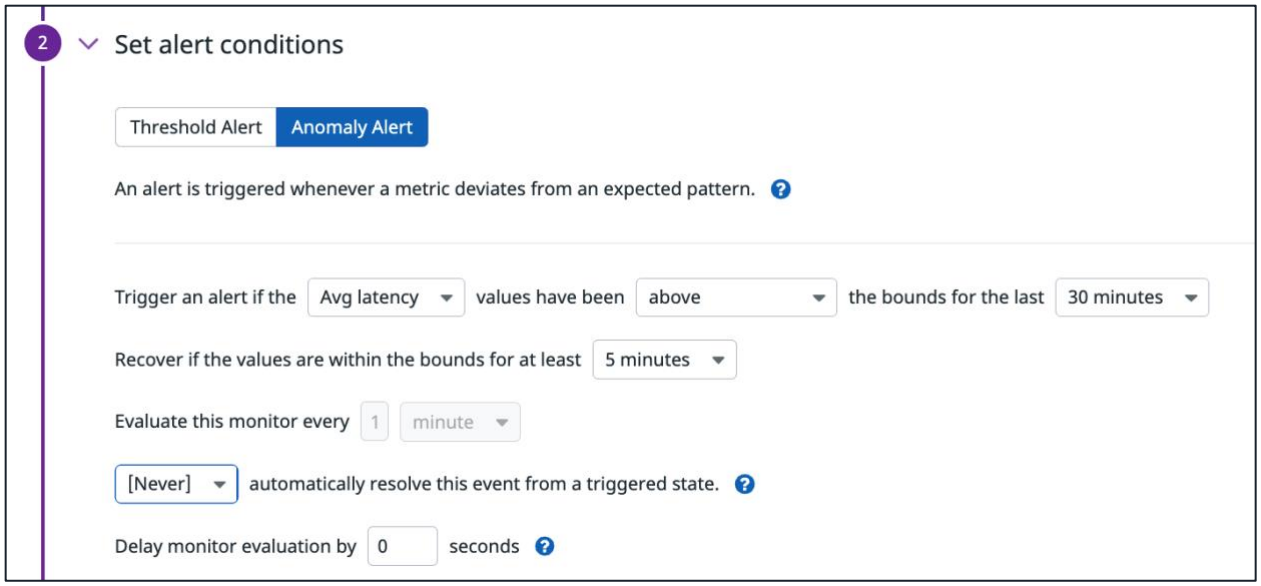
We will look at creating Latency and Error monitors.

APM Latency Monitors (Anomaly Alert):

1. On your Datadog account navigate to **Monitors** and click on **New Monitor**
2. From the list of Custom monitors select **APM**
3. On the new APM Monitor, under **Select monitor scope** make below selections:



4. As we will creating an anomaly-based alert, select **Anomaly Alert** under **Set alert conditions** and make below selections:



5. Move to **Say what's happening** section to add below as subject and body.

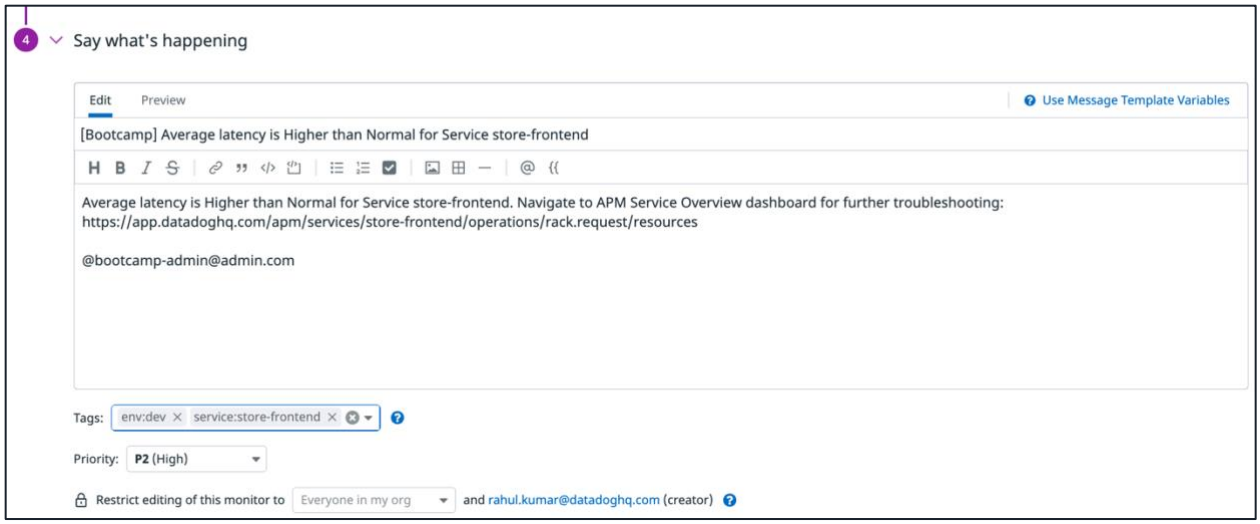
   Subject: [Bootcamp] Average latency is Higher than Normal for Service store-frontend
   Body:

Average latency is Higher than Normal for Service store-frontend. Navigate to APM Service Overview dashboard for further troubleshooting:
https://app.datadoghq.com/apm/services/store-frontend/operations/rack.request/resources

@bootcamp-admin@admin.com

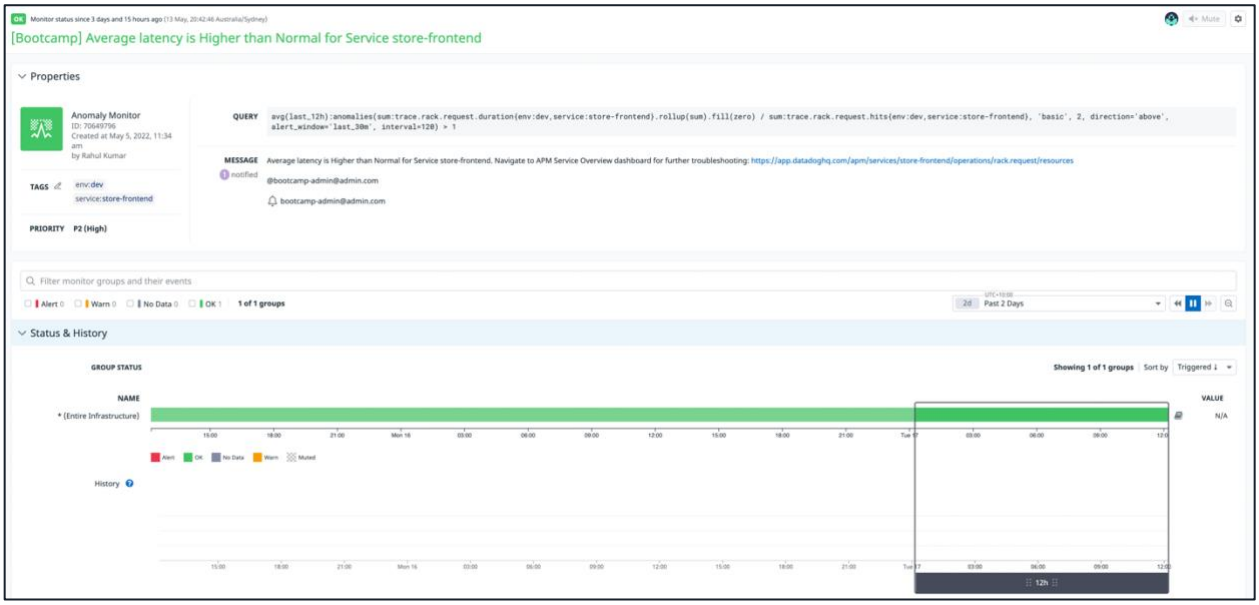
You may choose to add more tags. We will keep the **Priority** set to **P2 (High).**

The **Say what's happening** section of monitor should look as below:



6. Hit **Create** to create the monitor.

Your monitor should look like this:



It will be a few minutes before points appear in the **EVALUATION GRAPH** at the bottom of the status page.

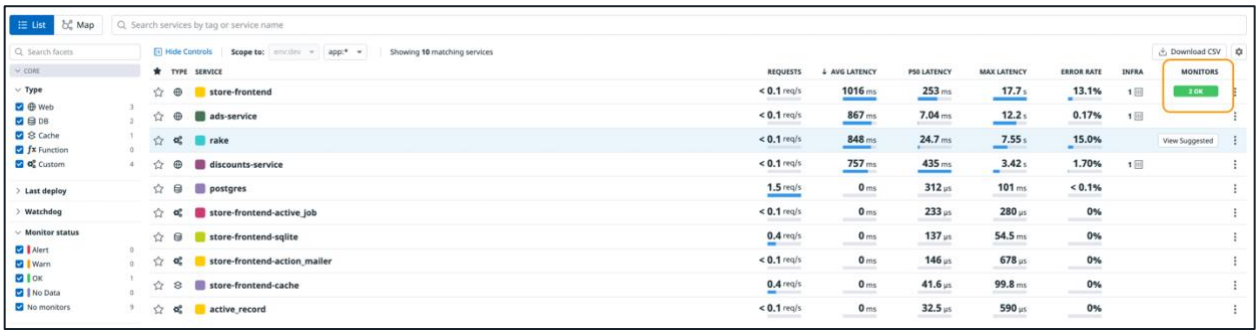
Scroll to the bottom of the monitor page and examine the **Events & Watchdog** section. The **Events** tab displays all events associated with this monitor, including a summary of its creation.

Watchdog is an algorithmic feature for APM performance and infrastructure metrics that automatically detects potential application and infrastructure issues. It can take days to weeks for Watchdog to collect enough data to detect anomalies, so you won't see anything in this tab during the short duration of this lab. You can learn more about Watchdog in the docs.

Monitors and APM

Datadog will automatically display an indication of a service's monitor in several APM pages. These indicators link to the associated monitor's status page:

Under the **MONITORS** column of the **Service List** page:



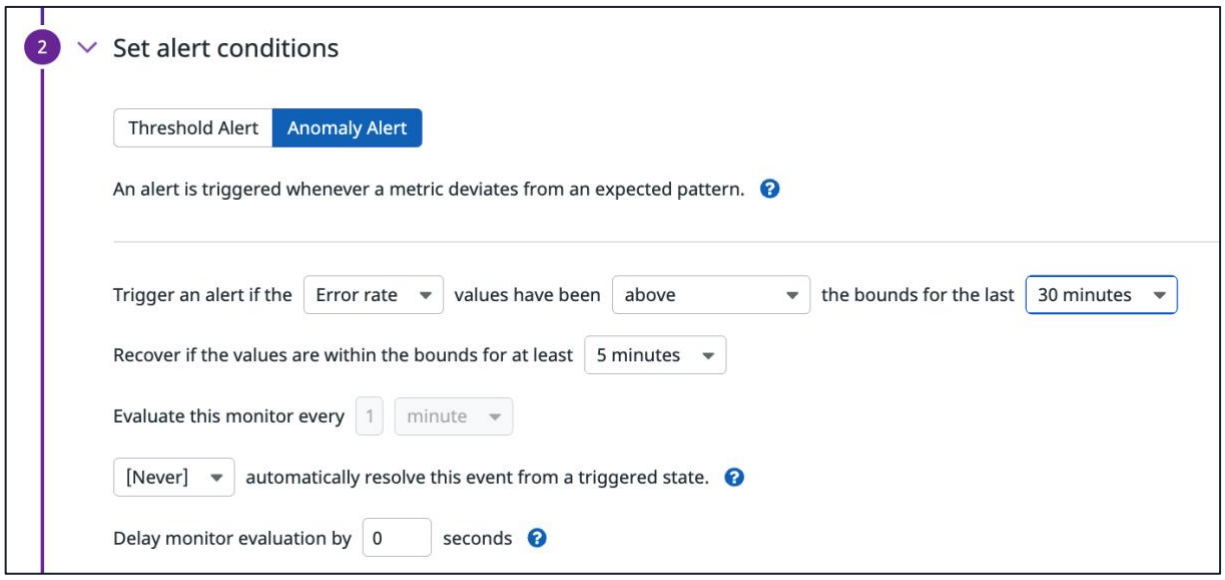
At the top of the **Service details** page:

Also, explore the Monitor status in the **Service Map** view.

Surfacing monitor status throughout APM pages makes it very easy to check on the status and history of the key metrics that are important to you.

APM Errors Monitor (Anomaly Alert):

1. Repeat steps 1 to 3 as in above lab.
2. As we will creating an anomaly-based alert, select **Anomaly Alert** under **Set alert conditions** and make below selections:



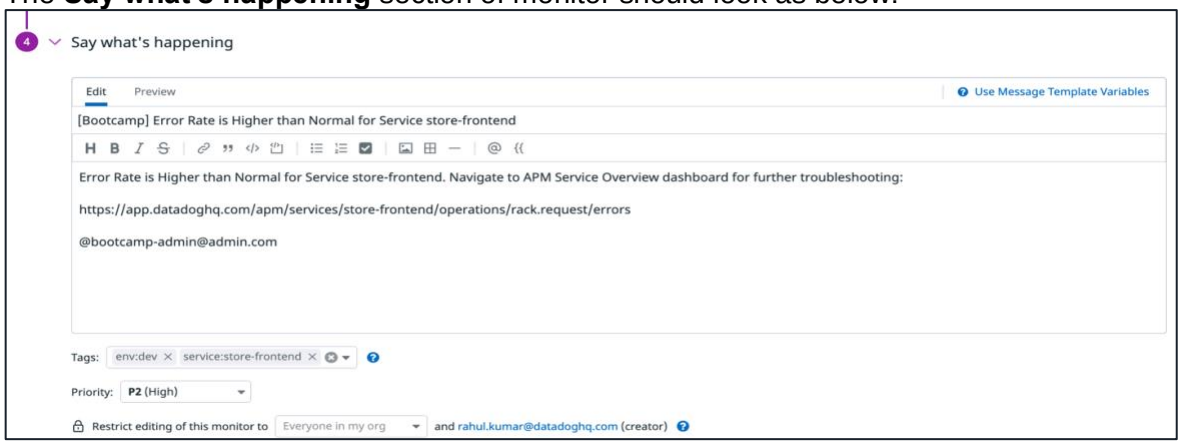
3. Move to **Say what's happening** section to add below as subject and body.

Subject: [Bootcamp] Error Rate is Higher than Normal for Service store-frontend
Body:

```
Error Rate is Higher than Normal for Service store-frontend. Navigate to APM Service Overview dashboard for further troubleshooting:

https://app.datadoghq.com/apm/services/store-frontend/operations/rack.request/errors

@bootcamp-admin@admin.com
```

You may choose to add more tags. We will keep the **Priority** set to **P2 (High).**

The **Say what's happening** section of monitor should look as below:



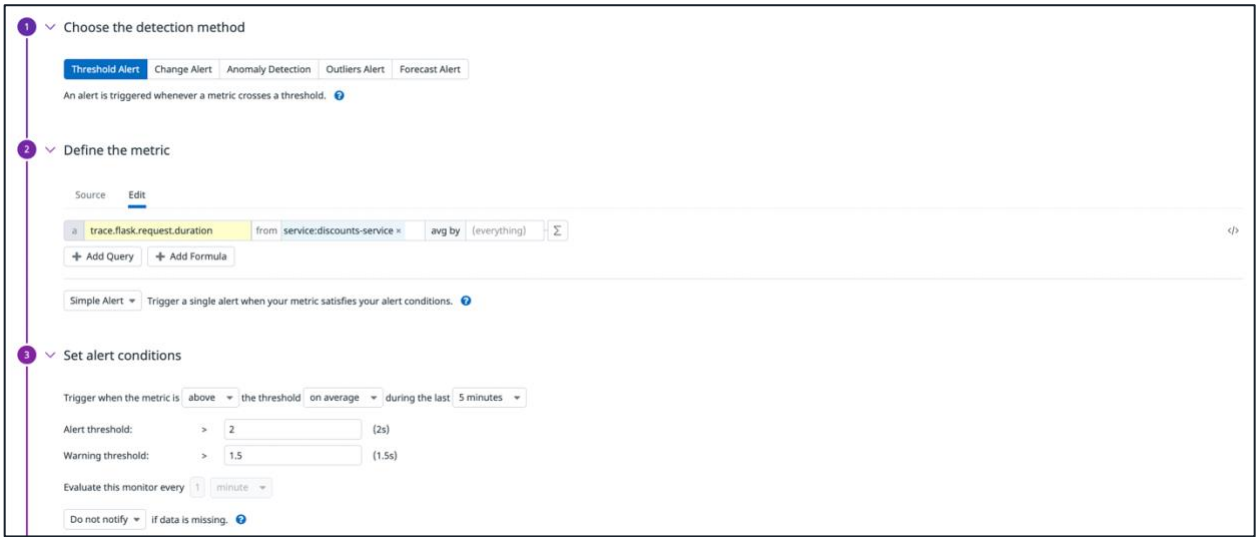
4. Hit **Create** to create the monitor.

APM Latency | Metric Monitor (Threshold Alert):

We will create a monitor that will alert you when the Flask request time for the discounts service exceeds 2 seconds.

1. On your Datadog account navigate to **Monitors** and click on **New Monitor**
2. From the list of Custom monitors select **Metric**
3. Under **Define the metric**, set Metric to **trace.flask.request.duration**
   For From, select **service:discounts-service**.
4. Under **Set alert conditions**, set **Alert threshold to 2**, and **Warning threshold to 1.5**.

Your new Metric Monitor configuration should look like this:



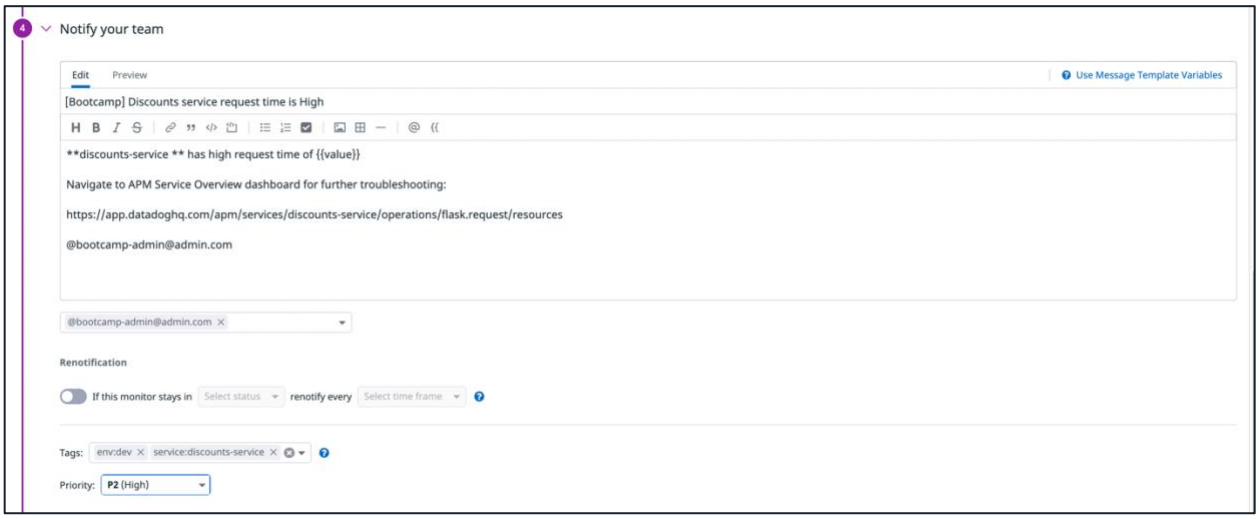
5. Move to **Say what's happening** section to add below as subject and body.

   Subject: [Bootcamp] Discounts service request time is High
   Body:

```
**discounts-service ** has high request time of {{value}}

Navigate to APM Service Overview dashboard for further troubleshooting:

https://app.datadoghq.com/apm/services/discounts-service/operations/flask.request/resources

@bootcamp-admin@admin.com
```

   You may choose to add more tags. We will keep the **Priority** set to **P2 (High).**

   The **Say what's happening** section of monitor should look as below:



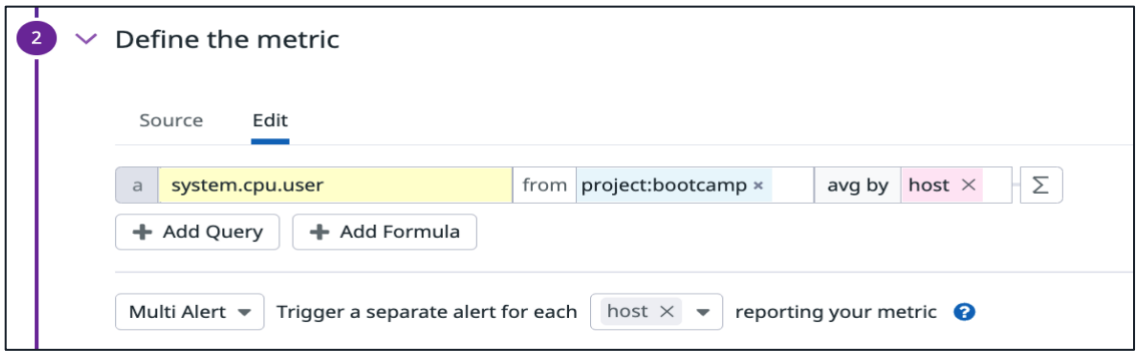
6. Hit **Create** to create the monitor.

109-2 Create Resource Monitors

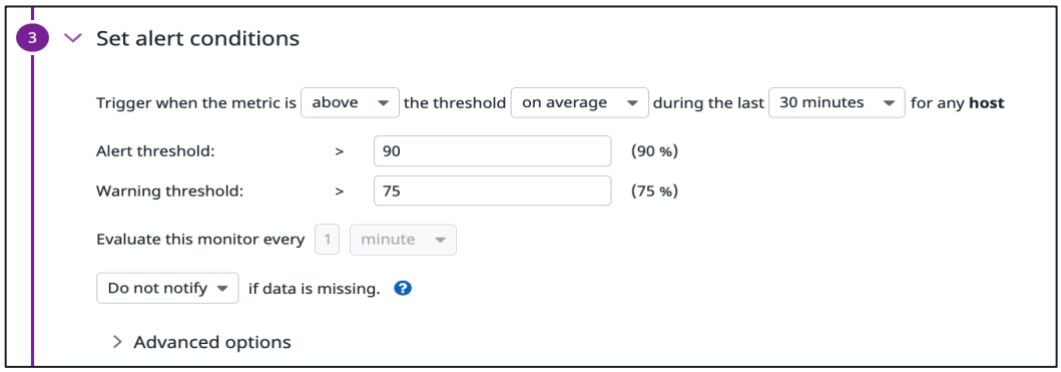
In this section we will create cpu and memory monitors to proactively alert issues related to infrastructure.

Infra CPU Usage Monitors (Threshold Alert):

1. On your Datadog account navigate to **Monitors** and click on **New Monitor**
2. From the list of Custom monitors select **Metric**
3. In the **Choose the detection method** make sure **Threshold Alert** is selected.
4. Make below selections for **Define the metric** section:



5. Under **Set alert conditions** set the **Alert threshold** as > **90** and the **Warning threshold** as > **75** during the last **30 minutes** for any host. The **Set alert conditions** section will look as below:



6. We will add below to **Notify your team** section:

Subject: [Bootcamp] CPU Usage is High on {{host.name}}
Body:

```
{{#is_alert}}

CPU usage is high on {{host.name}} with a value of {{value}}

Troubleshooting steps/dashboards:
1. Host Map :
https://app.datadoghq.com/infrastructure/map?fillby=avg%3Acpuutilization&filter=host%3A{{host.name}}
```

2. Troubleshoot Process adding up to CPU:
https://app.datadoghq.com/process?query=host%3A{{host.name}}
3. Troubleshoot running containers on the host:
https://app.datadoghq.com/containers?query=host%3A{{host.name}}

Notify(Pager duty action): pagerdurty@pagerduty.com

{{/is_alert}}

{{#is_warning}}
CPU usage is high on {{host.name}} with a value of {{value}}

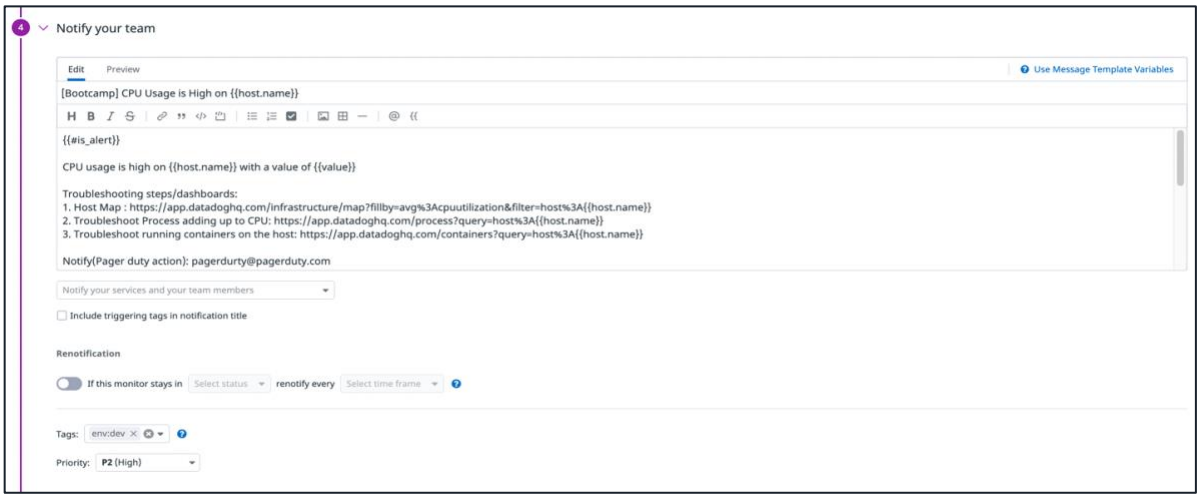
Troubleshooting steps/dashboards:
1. Host Map :
https://app.datadoghq.com/infrastructure/map?fillby=avg%3Acpuutilization&filter=host%3A{{host.name}}
2. Troubleshoot Process adding up to CPU:
https://app.datadoghq.com/process?query=host%3A{{host.name}}
3. Troubleshoot running containers on the host:
https://app.datadoghq.com/containers?query=host%3A{{host.name}}

Notify(Email Action): email@domain.com

{{/is_warning}}

You may choose to add more tags. We will keep the **Priority** set to **P2 (High).**

The **Say what's happening** section of monitor should look as below:



Click on the **Use Message Template Variables** to learn some examples.

7. Hit **Create** to create the monitor.

Infra Memory Usage Monitors (Threshold Alert):

In this lab we will learn to clone an existing monitor and make changes to it. This saves a lot of time when creating new monitors. We will use the **[Bootcamp] CPU Usage is High on {{host.name}}** monitor that we have just created. We will also learn to perform arithmetic operations in the monitor configuration.

1. On your Datadog account navigate to **Monitors** and click on **Manage Monitor**
2. Search for **[Bootcamp] CPU Usage is High**
3. As the monitor is displayed, hover on the right corner and click on the clone icon.

4. In the **Define the metric** section:
   a. We will change the metric from :**system.cpu.user** to **system.memory.used**.
   b. Click on **Add Query** and change the metric from **system.memory.used** to **system.memory.total**
   c. Apply the formula **(a / b) * 100** to calculate the percentage memory utilization

   The **Define the metric** section looks as below:

5. We will leave the **Set alert conditions** set to **90** and **75** for **Alert and Warning threshold** respectively.
6. We will make below changes to **Notify your team** section:

Subject: [Bootcamp] Memory Usage is High on {{host.name}}
Body:

```
{{#is_alert}}

Memory usage is high on {{host.name}} with a value of {{value}}

Troubleshooting steps/dashboards:
1. Host Map :
https://app.datadoghq.com/infrastructure/map?fillby=avg%3Asystem.mem.used&filter=host%3A{{host.name}}
2. Troubleshoot Process adding up to Memory:
https://app.datadoghq.com/process?query=host%3A{{host.name}}
3. Troubleshoot running containers on the host:
https://app.datadoghq.com/containers?query=host%3A{{host.name}}

Notify(Pager duty action): pagerdurty@pagerduty.com

{{/is_alert}}
```

{{#is_warning}}
Memory usage is high on {{host.name}} with a value of {{value}}

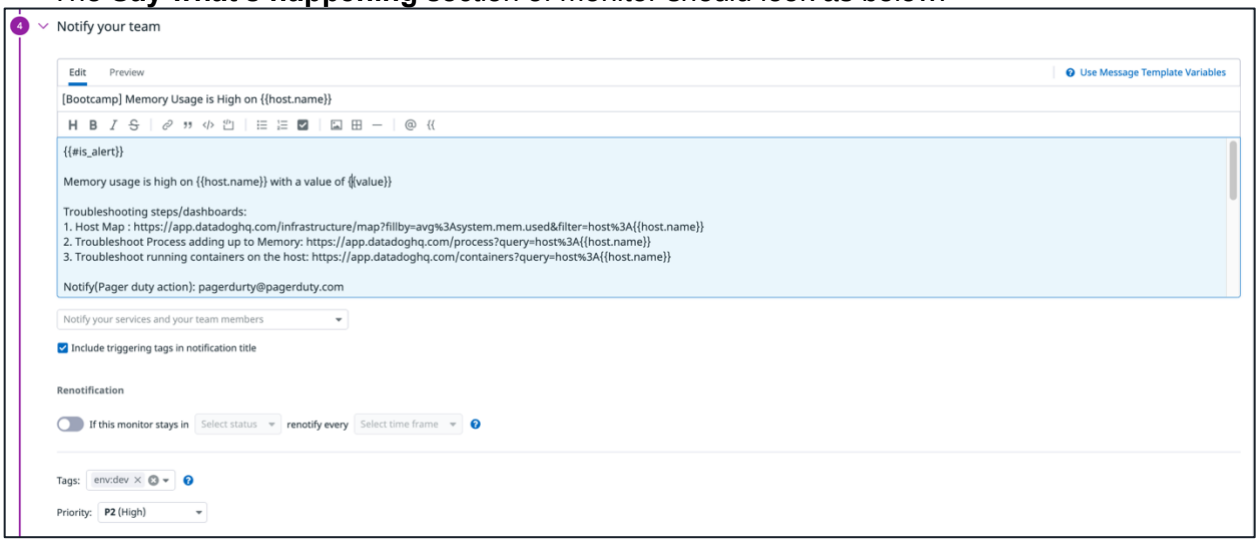
Troubleshooting steps/dashboards:
1. Host Map :
https://app.datadoghq.com/infrastructure/map?fillby=avg%3Asystem.mem.used&filter=host%3A{{host.name}}
2. Troubleshoot Process adding up to Memory:
https://app.datadoghq.com/process?query=host%3A{{host.name}}
3. Troubleshoot running containers on the host:
https://app.datadoghq.com/containers?query=host%3A{{host.name}}

Notify(Email Action): email@domain.com

{{/is_warning}}

You may choose to add more tags. We will keep the **Priority** set to **P2 (High).**

The **Say what's happening** section of monitor should look as below:

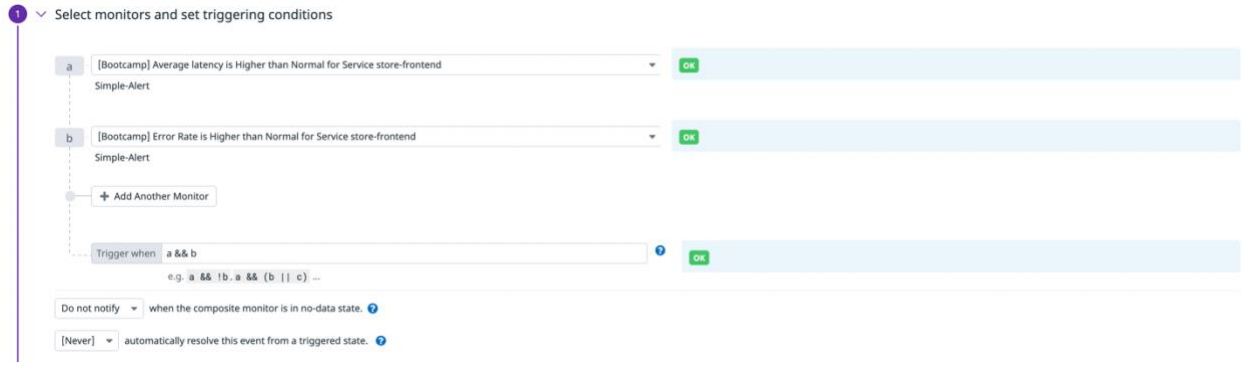


7. Hit **Create** to create the monitor.

109-3 Create Composite Monitors

In this lab we will learn to create a composite monitor using the monitors that we have created under section 107-1.

1. On your Datadog account navigate to **Monitors** and click on **New Monitor**
2. Click on **Composite**
3. Make selections as below:

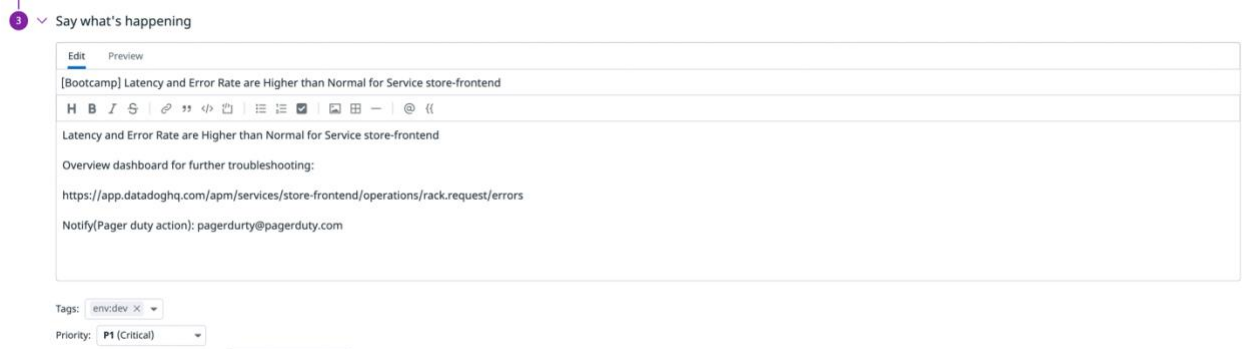


4. We will add below changes under **Say what's happening** section:
   Subject: [Bootcamp] Latency and Error Rate are Higher than Normal for Service store-frontend
   Body:

Latency and Error Rate are Higher than Normal for Service store-frontend Overview dashboard for further troubleshooting: https://app.datadoghq.com/apm/services/store-frontend/operations/rack.request/errors Notify(Pager duty action): pagerdurty@pagerduty.com

   You may choose to add more tags. We will keep the **Priority** set to **P1 (Critical).**

   The **Say what's happening** section of monitor should look as below:



8. Hit **Save** to create the monitor.

109-4 SLO monitors

SLOs provide a framework for defining clear targets around application performance. They measure a target percentage of a metric over time to track how well a service is meeting its obligation to its consumers.

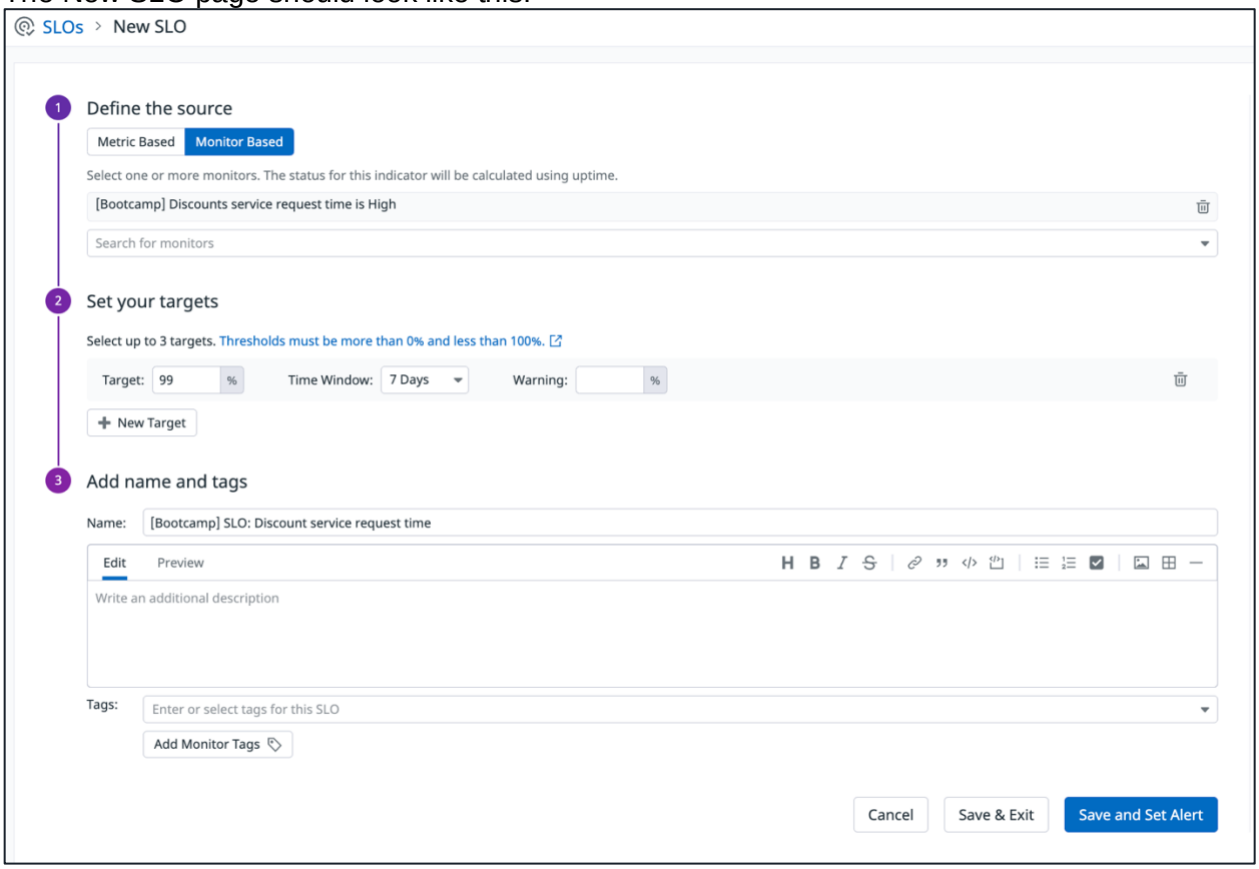
An example of an SLO might be, "the discounts service should be running 99% of the time for any 7-day period."

Now that you have a monitor on the discounts service request time, you can use that as the Service Level Indicator (SLI) for a new SLO.

Create this SLO for the discounts service:

1. Navigate to **Monitors > New SLO**.
2. Under Define the source, select **Monitor Based**.
3. Select the **[Bootcamp] Discounts service request time is High** monitor.
4. Under **Set your targets**, click the **New Target** button.
5. Leave the default **99% over 7 days.**
6. Under **Add name and tags** set **Name** to **[Bootcamp] SLO: Discount service request time**

The New SLO page should look like this:



7. Click **Save and Set Alert**.

8. On the Configure Alerts page, notice that you can specify whom to alert under **2. Notify Your Team**. Leave the default for this lab.

9. Under **Say what's happening**, in the large text area with the placeholder text, "Example Monitor Message:", paste **Discounts service request time budget depleted.**

10. Click the **Save & Exit** button.

11. The SLO details panel for this SLO will open on the SLOs page. Click the **Status & History** tab to see the SLO in action.