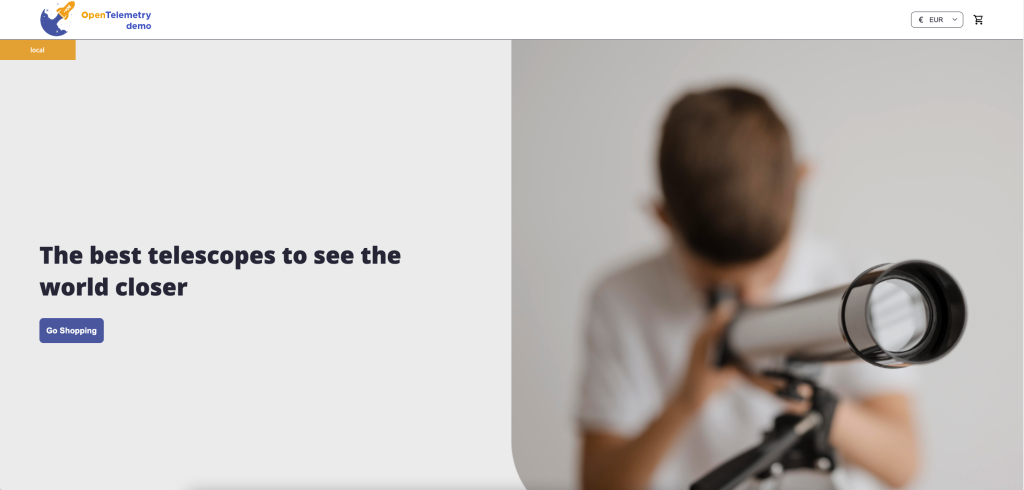
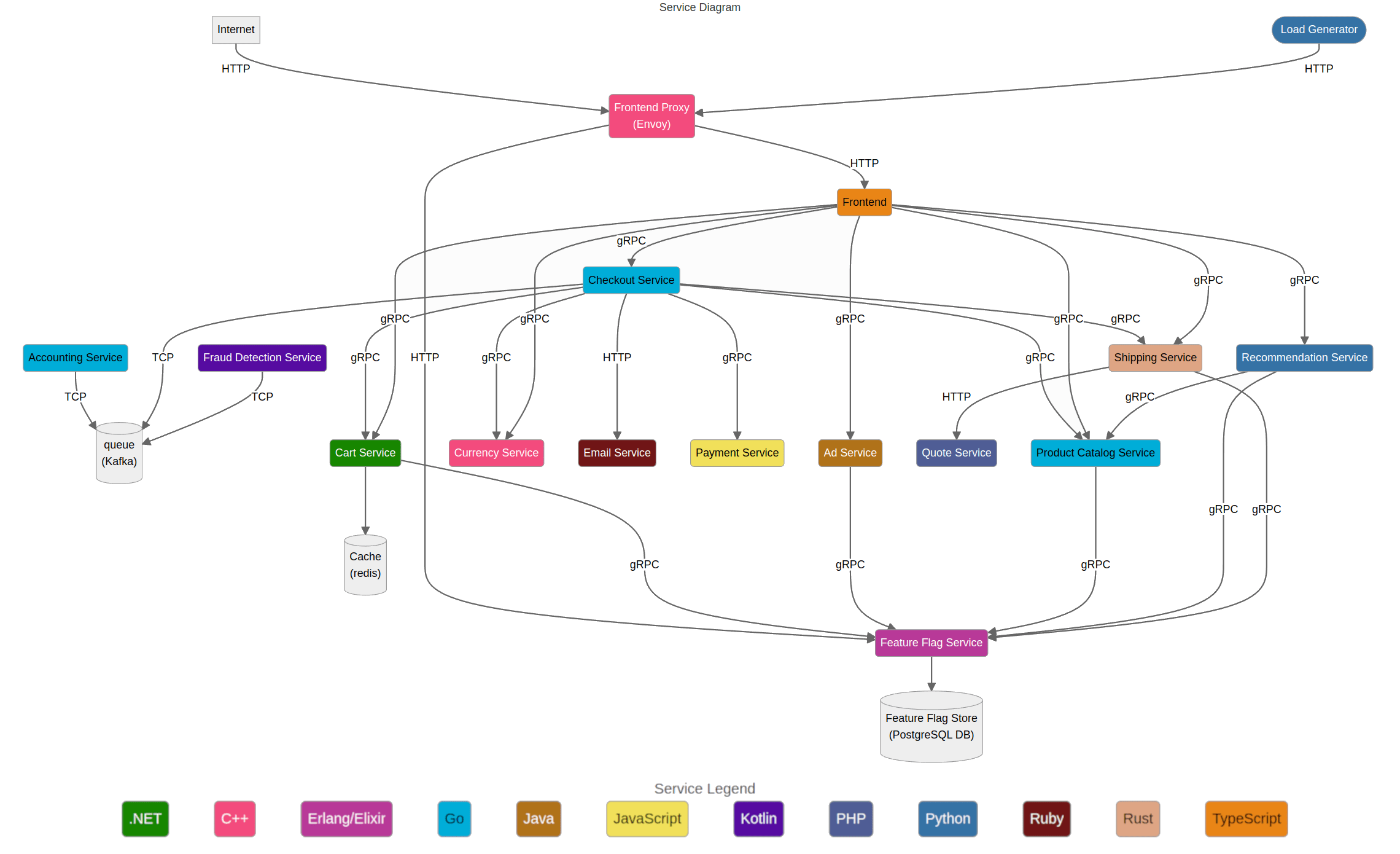
**OpenTelemetry – Dynatrace Integration**

[OpenTelemetry](https://www.dynatrace.com/news/blog/what-is-opentelemetry-2/) is an exciting open source observability standard that provides a common collection of tools, APIs, and SDKs to help collect observability signals. Many companies are interested in experimenting with OpenTelemetry, but have struggled with the lack of a good reference implementation.

To tackle this challenge, the OpenTelemetry community created a [demo application](https://github.com/open-telemetry/opentelemetry-demo) to act as a showcase for the features and capabilities of this technology. The demo has been in active development since the summer of 2022 with Dynatrace as one of its leading contributors.


The demo application is a cloud-native e-commerce application made up of multiple microservices. Because it includes examples of 10 programming languages OpenTelemetry supports with SDKs, the application makes a good reference for developers on how to use OpenTelemetry. You can also use it to test different OpenTelemetry features and evaluate how they appear on backends. Moreover, you can use it as a framework for further customization.

OpenTelemetry demo application architecture diagram. Courtesy of the [OpenTelemetry authors](https://github.com/open-telemetry/opentelemetry-demo).

To run the demo, you’ll need either a Docker or Kubernetes environment. By default, the demo comes with Jaeger and Prometheus backends for displaying the collected traces and metrics, but you can easily configure alternative backends. In this example, we will be using Dynatrace.

**Using Dynatrace as backend to the OpenTelemetry demo application**

Dynatrace has supported OpenTelemetry since 2020. You can ingest OpenTelemetry data in two ways: using Dynatrace OneAgent technology or using OpenTelemetry Protocol (OTLP), which relies exclusively on open source technology. Both methods ingest data, but by using the Dynatrace OneAgent, users can automatically discover additional insights about their infrastructure, applications, processes, services and databases. However, in this example, we’ll deploy the demo application to send telemetry directly to Dynatrace using OTLP, so you can see how Dynatrace presents the OpenTelemetry data without the additional context OneAgent provides.

To set up the demo using Docker, follow the steps below. You can find additional deployment options in the [OpenTelemetry demo documentation](https://opentelemetry.io/docs/demo/).

**Step 1. Download the demo application**

git clone https://github.com/open-telemetry/opentelemetry-demo.git

cd opentelemetry-demo/

**Step 2. Configure the demo application to send telemetry to Dynatrace**

At the cloned repository’s root, create a new file called docker-compose.override.yml and paste the following into the file:

services:

otelcol:

environment:

- DT\_OTLP\_ENDPOINT

- DT\_API\_TOKEN

This ensures all necessary environment variables are passed to all services, specifically:

* DT\_OTLP\_ENDPOINT is the OTLP endpoint that is used by the collector to export to Dynatrace.
* DT\_API\_TOKEN is the token for your Dynatrace environment. Instructions for how to create that token follow in the next step.

We will create all of these variables in step 3.

Next, go to src/otelcollector/otelcol-config-extras.yml. This file contains a configuration that will be merged with the collector’s configuration.

Copy the following content to the file:

exporters:

*# otlp/http exporter to Dynatrace.*

otlphttp/dynatrace:

endpoint: "${DT\_OTLP\_ENDPOINT}"

headers:

Authorization: "Api-Token ${DT\_API\_TOKEN}"

processors:

cumulativetodelta:

filter/histograms:

error\_mode: ignore

metrics:

metric:

- 'type == METRIC\_DATA\_TYPE\_HISTOGRAM'

service:

pipelines:

traces/dynatrace:

receivers: [otlp]

processors: [batch]

exporters: [otlphttp/dynatrace]

metrics/dynatrace:

receivers: [otlp, spanmetrics]

processors: [filter/histograms, batch, cumulativetodelta]

exporters: [otlphttp/dynatrace]

logs/dynatrace:

receivers: [otlp]

processors: [batch]

exporters: [otlphttp/dynatrace]

This configuration will export traces, metrics, and logs to Dynatrace using OTLP (OpenTelemetry protocol). The configuration also includes an optional span metrics connector, which generates Request, Error, and Duration (R.E.D.) metrics from span data. All the needed components are available out of the box in the OpenTelemetry collector contrib distribution, which is included in the demo application.

**Step 3. Set up your Dynatrace account and environment variables**

Create a Dynatrace account. If you don’t have one, you can use a [trial account](https://www.dynatrace.com/signup/).

Next, create an access token that includes scopes for the following:

* Ingest OpenTelemetry traces (openTelemetryTrace.ingest)
* Ingest metrics (metrics.ingest)
* Ingest logs (logs.ingest)

For details, see [Dynatrace API – Tokens and authentication](https://docs.dynatrace.com/docs/dynatrace-api/basics/dynatrace-api-authentication) in the Dynatrace documentation.

Export the environment variables. Make sure to replace the placeholder values with your token and environment ID. This example illustrates how to pass the token most easily using the terminal. In a production environment, you would configure the token using Kubernetes secrets or other secure secret handling mechanisms.

export DT\_API\_TOKEN=dt0c01.MY\_SECRET\_TOKEN

export DT\_OTLP\_ENDPOINT=https:*//{your-env-id}.live.dynatrace.com/api/v2/otlp*

export OTEL\_EXPORTER\_OTLP\_METRICS\_TEMPORALITY\_PREFERENCE=delta

**Step 4. Deploy the application**

When everything is prepared, deploy your demo application.

**docker compose up --no-build**

If you use ARM architecture (for example, a MacBook with Apple silicon), remove the --no-build option to build the images locally.

The first time running the demo, it takes a couple of minutes to download all the needed images. Afterward, the demo starts instantly, and the load generator automatically begins creating data.

You can open the demo application UI to see the app in action:

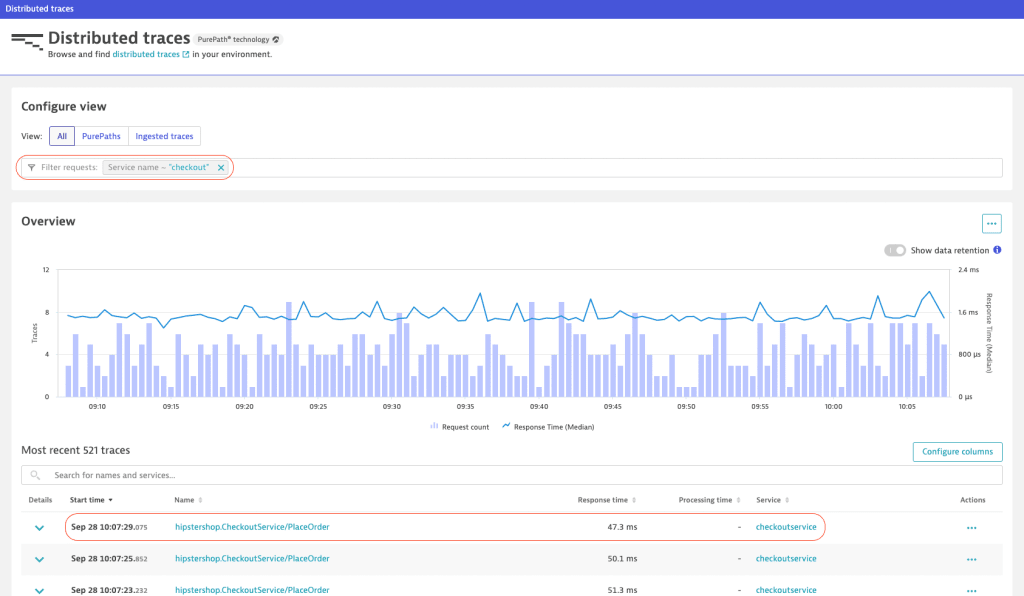
**http:*//localhost:8080/***

**Use Dynatrace to observe traces, metrics, and logs**

Open your Dynatrace environment in the browser (using the link from the account creation step). Using the navigation on the left, open the “Distributed traces” page. When the application is running, traces, metrics, and logs start appearing on Dynatrace.

**View traces**

In this view, you can filter traces. For example, to visualize traces for the checkout service, you can enter “checkout” in the Filter requests field, as shown in the following image.



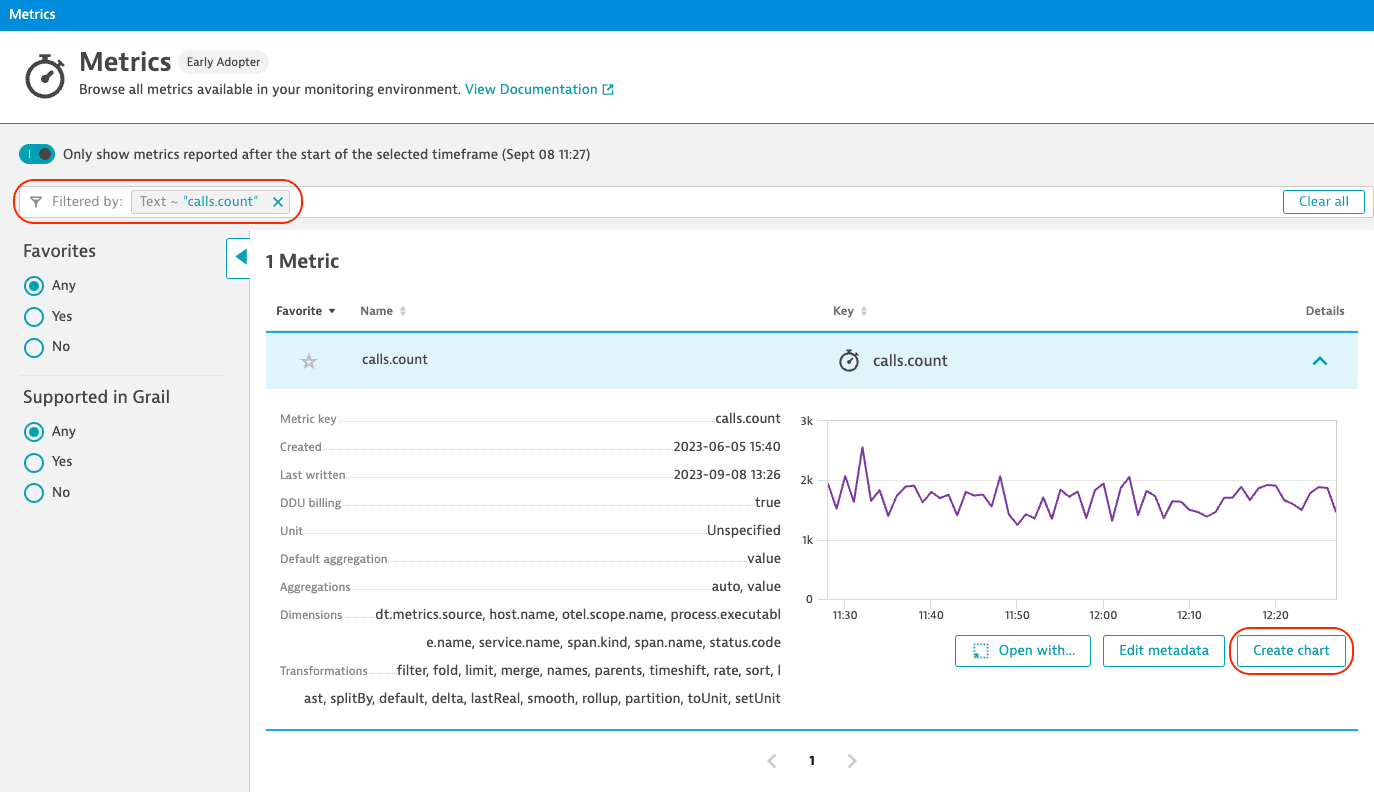
If you select one of the traces, you can see the different spans going through multiple services, as shown in the following image.

A screenshot of a computer

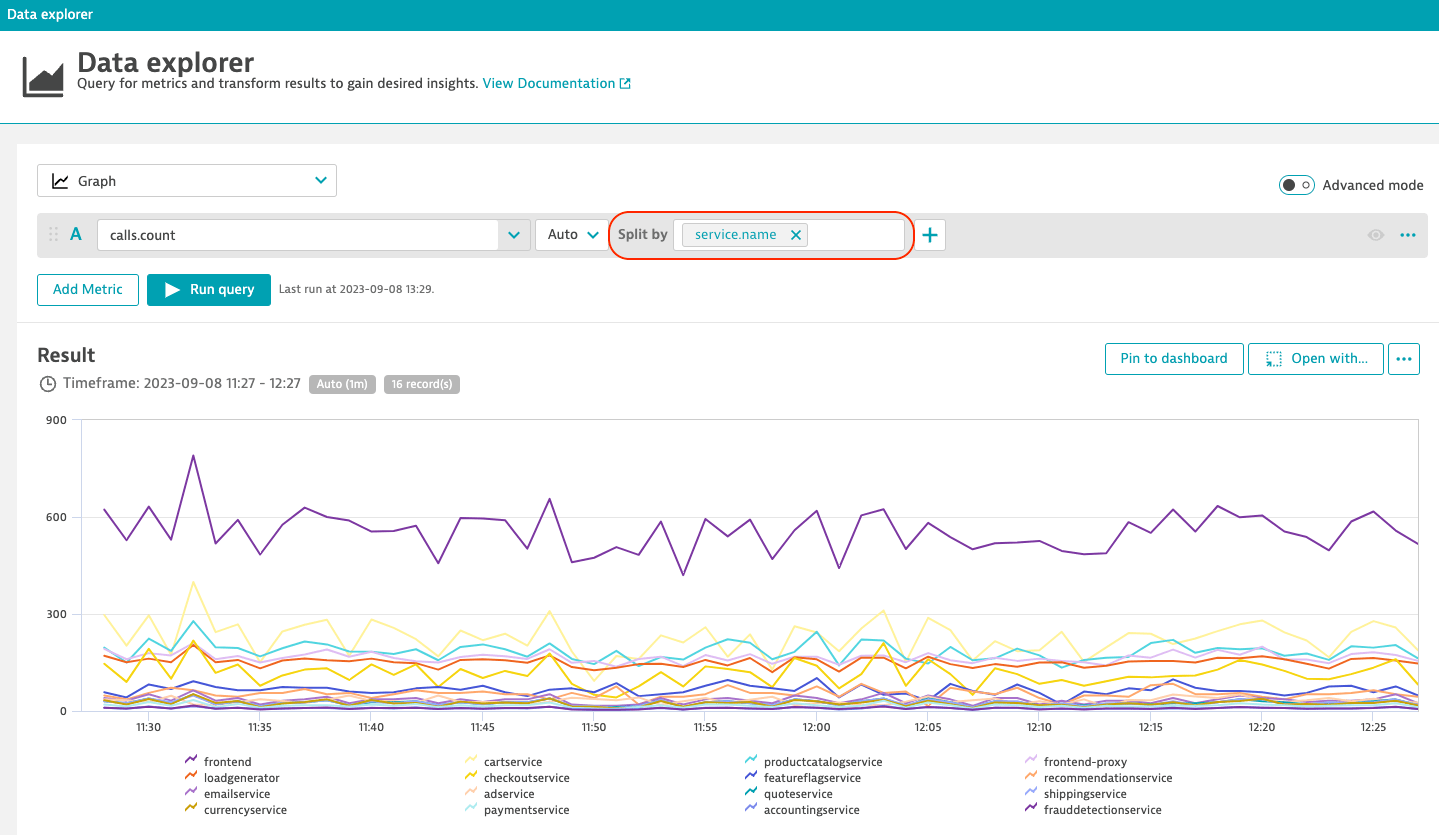
Description automatically generated

**View metrics**

Next, take a look at the received metrics by opening the “Metrics” view in the left navigation. You should see a lot of different metrics. Filter using calls.count to see the metric created by the span metrics connector.

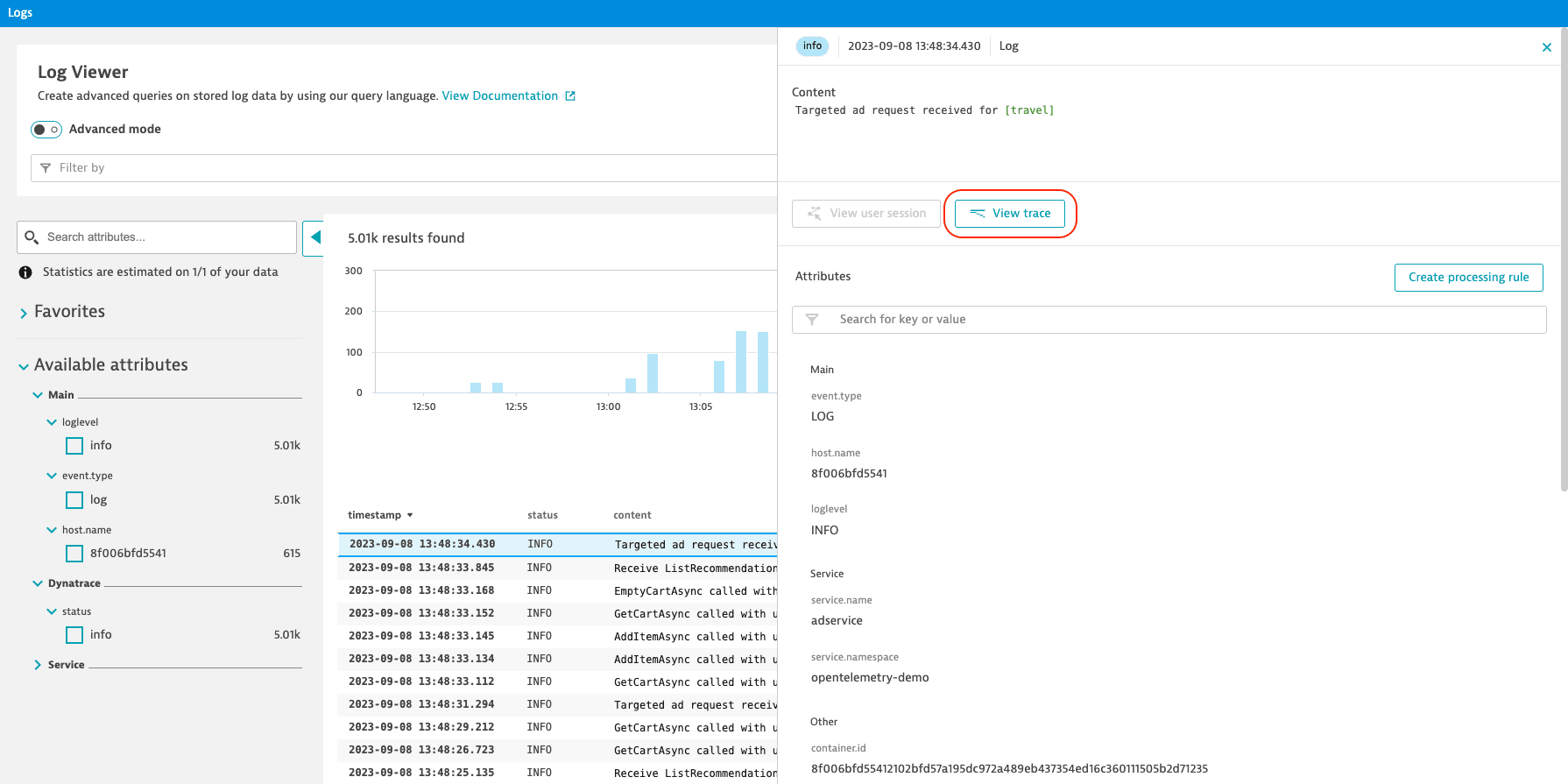


The span metrics connector shows the number of spans each demo app service creates. In the Data Explorer, use the “Split by” field with the value service.name to separate the number of spans per service. You can see that it’s the front-end service that creates the most spans.



**View logs**

One of the latest additions in the demo app is logs. To access logs in Dynatrace, navigate to the Log Viewer page. Select one of the log lines, and a sidebar appears with the available attributes. Select “View trace” to connect your log lines with traces and see in detail which request (distributed trace) led to this log line. The magic in the log correlation is the trace\_id and span\_id attributes OpenTelemetry libraries attach to each log message if available.



**Using feature flags: Simulating application failures use case**

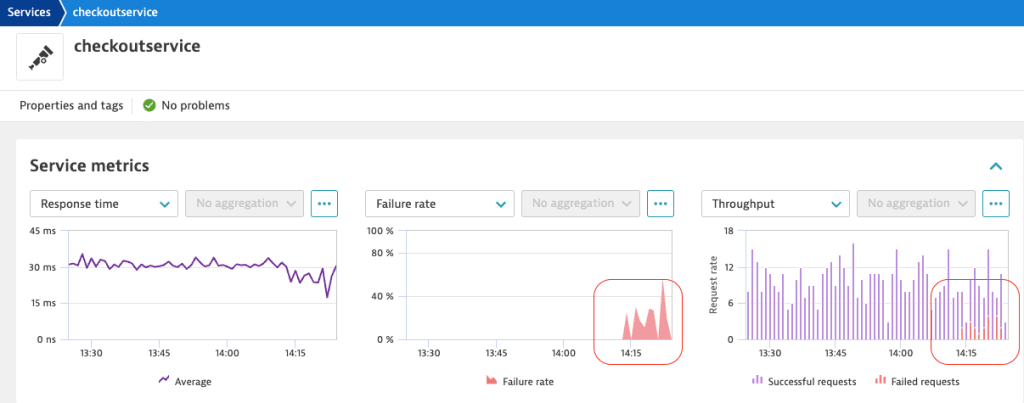
One interesting use case in the OpenTelemetry demo application is simulating application failures. This failure logic is built into the application and it can be enabled by using a feature flag service.

To open the feature flag UI, go to:

http:*//localhost:8080/feature*

You should see a feature flag called productCatalogFailure. To enable this feature flag, click edit, enable, then save.

Go to the Services page and select checkoutservice. After a minute, you should start seeing some of the checkout service requests failing.



Scroll down on the same page to see a list of traces and expand one of the failing traces, indicated by a red bar as in the picture below. In the status section of the span you can see that Dynatrace provides the reason for the failure: failed to get product #"OLJCESPC7Z.

