# S OpenSearch

# Documentation

uocumentation

INSTALL AND CONFIGURE	~
UPGRADE TO OPENSEARCH	~
OPENSEARCH	~
OPENSEARCH DASHBOARDS	~
SECURITY PLUGIN	~
SECURITY ANALYTICS PLUGIN	~
SEARCH PLUGINS	~
INDEX MANAGEMENT PLUGIN	~
REPLICATION PLUGIN	~

•

•

**OBSERVABILITY PLUGIN** 

 $\boldsymbol{\wedge}$ 

About Observability

**Observability security** 

**Event analytics** 

**Operational panels** 

### Trace analytics / Analyze Jaeger trace data



The trace analytics functionality in the OpenSearch Observability plugin now supports Jaeger trace data. If you use OpenSearch as the backend for Jaeger trace data, you can use the trace analytics built-in analysis capabilities. This provides support for OpenTelemetry (OTEL) formatted trace data.

When you perform trace analytics, you can select from two data sources:

- **Data Prepper** Data ingested into OpenSearch through Data Prepper.
- Jaeger Trace data stored within OpenSearch as its backend.

If you currently store your Jaeger trace data in OpenSearch, you can now use the capabilities built into trace analytics to analyze the error rates and latency. You can also filter the traces and look into the span details of a trace to pinpoint any service issues.

When you ingest Jaeger data into OpenSearch, it gets stored in a different index than the OTEL-generated index that gets created when you run data through Data Prepper. Use the data source selector in Dashboards to indicate on which data source you want to perform trace analytics.

Jaeger trace data that you can analyze includes span data, as well as service and operation endpoint data.

By default, each time you ingest data for Jaeger, it creates a separate index for that day.

To learn more about Jaeger data tracing, see the Jaeger open source documentation.

### **Data ingestion requirements**

To use trace analytics with Jaeger data, you need to configure error capability.

Jaeger data that is ingested for OpenSearch needs to have the environment variable ES\_TAGS\_AS\_FIELDS\_ALL set to true for errors. If data is not ingested in this format it will not work for errors and error data will not be available for traces in trace analytics with OpenSearch.

#### About data ingestion with Jaeger indexes

Trace analytics for non-Jaeger data use OTEL indexes with the naming conventions otelv1-apm-span-\* Or otel-v1-apm-service-map\*.

Jaeger indexes follow the naming conventions jaeger-span-\* Or jaeger-service-\*.

### How to set up OpenSearch to use Jaeger data

The following section provides a sample Docker compose file that contains the required configuration to enable errors for trace analytics.

#### Step 1: Run the Docker compose file

Use the following Docker compose file to enable Jaeger data for trace analytics with the ES\_TAGS\_AS\_FIELDS\_ALL environment variable set to true to enable errors to be added to trace data.

Copy the following Docker compose file contents and save it as docker-compose.yml.

### version: '3'

services:

opensearch-node1: # This is also the hostname of the container within the Docker n image: opensearchproject/opensearch:latest # Specifying the latest available ima container name: opensearch-node1

environment:

- cluster.name=opensearch-cluster # Name the cluster
- node.name=opensearch-node1 # Name the node that will run in this container
- discovery.seed\_hosts=opensearch-node1,opensearch-node2 # Nodes to look for w - cluster.initial\_cluster\_manager\_nodes=opensearch-node1,opensearch-node2 # No
- bootstrap.memory\_lock=true # Disable JVM heap memory swapping

- "OPENSEARCH\_JAVA\_OPTS=-Xms512m -Xmx512m" # Set min and max JVM heap sizes to ulimits:

memlock:

soft: -1 # Set memlock to unlimited (no soft or hard limit)

- hard: -1
- nofile:
- soft: 65536 # Maximum number of open files for the opensearch user set to hard: 65536
- volumes:

- opensearch-data1:/usr/share/opensearch/data # Creates volume called opensear ports:

- "9200:9200"
- "9600:9600" networks:

- opensearch-net # All of the containers will join the same Docker bridge netw

opensearch-node2:

image: opensearchproject/opensearch:latest # This should be the same image used container\_name: opensearch-node2

environment:

- cluster.name=opensearch-cluster
- node.name=opensearch-node2
- discovery.seed\_hosts=opensearch-node1,opensearch-node2
- cluster.initial\_cluster\_manager\_nodes=opensearch-node1,opensearch-node2
  - bootstrap.memory\_lock=true

```
- "OPENSEARCH_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
      nofile:
        soft: 65536
        hard: 65536
    volumes:
      - opensearch-data2:/usr/share/opensearch/data
    networks:
      - opensearch-net
 opensearch-dashboards:
    image: opensearchproject/opensearch-dashboards:latest # Make sure the version of
    container_name: opensearch-dashboards
    ports:
      - 5601:5601 # Map host port 5601 to container port 5601
    expose:
      - "5601" # Expose port 5601 for web access to OpenSearch Dashboards
    environment:
      OPENSEARCH_HOSTS: '["https://opensearch-node1:9200","https://opensearch-node2:
    networks:
      - opensearch-net
  jaeger-collector:
   image: jaegertracing/jaeger-collector:latest
    ports:
      - "14269:14269"
      - "14268:14268"
      - "14267:14267"
      - "14250:14250"
      - "9411:9411"
    networks:
      - opensearch-net
    restart: on-failure
    environment:

    SPAN_STORAGE_TYPE=opensearch

    ES_TAGS_AS_FIELDS_ALL=true

      - ES_USERNAME=admin
      - ES_PASSWORD=admin
      - ES_TLS_SKIP_HOST_VERIFY=true
    command: [
      "--es.server-urls=https://opensearch-node1:9200",
      "--es.tls.enabled=true",
    ]
    depends_on:
      - opensearch-node1
  jaeger-agent:
    image: jaegertracing/jaeger-agent:latest
    hostname: jaeger-agent
    command: ["--reporter.grpc.host-port=jaeger-collector:14250"]
    ports:
      - "5775:5775/udp"
      - "6831:6831/udp"
      - "6832:6832/udp"
      - "5778:5778"
    networks:
      - opensearch-net
    restart: on-failure
    environment:
      - SPAN_STORAGE_TYPE=opensearch
    depends_on:
      - jaeger-collector
 hotrod:
   image: jaegertracing/example-hotrod:latest
    ports:
      - "8080:8080"
    command: ["all"]
    environment:
      - JAEGER_AGENT_HOST=jaeger-agent
      - JAEGER_AGENT_PORT=6831
    networks:
      - opensearch-net
    depends_on:
      - jaeger-agent
volumes:
 opensearch-data1:
 opensearch-data2:
networks:
 opensearch-net:
```

#### Step 2: Start the cluster

Run the following command to deploy the Docker compose YAML file.

docker compose up -d

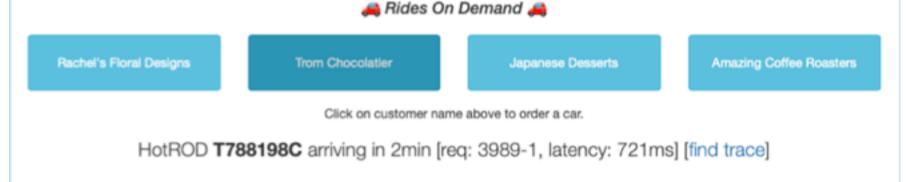
To stop the cluster, run the following command:

docker compose down

### Step 2: Generate sample data

Use the sample app provided with the Docker file to generate data. After you run the Docker compose file, it runs the sample app in your local host port 8080. To open the app, go to http://localhost:8080.

•	•••		<	> ,	2)	$\mathbb{O}_{\mathbb{C}}$	localhost	Ċ	٩	₫	+	Ō
	Your w	eb client's	ld: 39	89								
							Hot R.O.D.					



In the sample app, Hot R.O.D., select any one of the buttons to generate data. Now you can view trace data in Dashboards.

# **Step 3: View trace data in OpenSearch Dashboards**

After you generate Jaeger trace data you can go to OpenSearch Dashboards to view your trace data.

Go to Dashboards Trace analytics at http://localhost:5601/app/observability-dashboards#/trace\_analytics/home.

## Use trace analytics in OpenSearch Dashboards

To analyze the Jaeger trace data in Dashboards, first set up the trace analytics functionality. To get started, see Get started with trace analytics.

### **Data sources**

You can specify either Data Prepper or Jaeger as the data source when you perform trace analytics. From Dashboards, go to **Observability > Trace analytics** and select Jaeger.

Some of the second seco	ashboards	
	ervability Trace analytics	Dashboard
Observability Application analytics Trace analytics	Dashboa <sub>Jaeger</sub> ~	ard
Services Event analytics	CHOOSE DATA TYPE	oup name, service name
Metrics analytics Operational panels Notebooks	Q     Filter options       ✓     Jaeger       Data Prepper	rer time

## **Dashboard view**

After you select Jaeger for the data source, you can view all of the indexed data in Dashboard view, including Error rate and Throughput.

#### **Error rate**

You can view the trace error count over time in the **Dashboard** view and also see the top five combinations of services and operations that have a non-zero error rate.

ashboard <sup>ger</sup> ~						
Trace ID, trace group name, service	name	<b>.</b> ~	Last 5 years		Show dates	C Refres
+ Add filter						
ace error rate over time					Error rate T	hroughput
		Nov	: 51.15%			
60%						
20%						
2014						
20%						
0%2021	2021		2021	2021	2021	
op 5 Service and Operation Errors	(5)					
Service and Operation Name ©		Average	latency (ms) <sup>©</sup>		Error rate	Traces ©
rontend0,HTTP GET /config0			0.04		100%	89,774
rontend0,HTTP GET /config2			0.04		100%	88,449
rontend0,HTTP GET /config4			0.04		100%	89,586
rontendo, HTTP GET /conlig4			0.04		100%	89,349
rontend0,HTTP GET /config6			0.04			00,040

### Throughput

With **Throughput** selected, you can see the throughput of traces on Jaeger indexes that are coming in over time.

You can select an individual trace from Top 5 Service and Operation Latency list and view the detailed trace data.

Dashboard					
∩ Trace ID, trace group name, se	ervice name	🛗 🗸 Last 5 year	s	Show dates	C Refrest
) + Add filter					
Traces over time				Error rate Thr	oughput
		Now: 1,306,425			
(°) 1M IN 0.5M					
0	2021	2021	2021	2021	
Top 5 Service and Operation	Latency (5)		< 95 percentile	■ >= 95 percentile	
Service and Operation Name $^{\odot}$	Latency variance (ms 0 0.2 0.4 0.6	) <sup>⊙</sup> 0.8 1 <sup>↓</sup>	Average latency (ms) $^{\odot}$	Error rate ©	Traces ©
frontend0,HTTP GET /config0	1		0.04	100%	89,774
frontend0,HTTP GET /config1	1		0.04	0%	91,358

frontend0,HTTP GET /config1	0.04	0%	91,358

You can also see the combinations of services and operations that have the highest latency.

If you select one of the entries for Service and Operation Name and go to the Traces column to select a trace, it will automatically add the service and operation as filters.

# **Traces**

In **Traces**, you can see the latency and errors for the filtered service and operation for each individual Trace ID in the list.

Traces				
laeger 🗸				
Q Trace ID, trace group nam	e, service name	🛗 🗸 Last 5 years	Show dates	C Refres
process.serviceName: fronter	d0 × operationName: HTTP GET /config0 ×	+ Add filter		
Trace ID 1		Latency (ms)		ast updated
Trace ID ↑ 00de6a9aaf045bd400	6	Latency (ms) 0.04	Ver <sup>1</sup>	ast updated 2/16/2022 0:08:12
	<b>a</b>		Yes 1	2/16/2022

If you select an individual Trace ID, you can see more detailed information about the trace, such as time spent by the service and each span for the service and operation. You can also view the payload that you get from the index in JSON format.

Image: Construction of the second	6a9aaf045bd400	0
00de6a9aaf045bd400		
Overview		
Trace ID 00de6a9aaf045bd400 👔	Latency Errors 0.04 Yes Last updated 12/16/2022 10:08:12	
Time spent by service	Spans (3) Timeline Sp	pan list
• frontend0 100%	0 ms       0.01 ms       0.02 ms       0.03 ms       0.04 ms         HTTP GET /config0       A Error         frontendo       A Error         frontendo       A Error         frontendo       A Error         A Error       A Error         frontendo       A Error         A Error       A Error         A Error       A Error	
<pre>Payload</pre>		

### **Services**

You can also look at individual error rates and latency for each individual service. Go to **Observability > Trace analytics > Services**. In **Services**, you can see the average latency, error rate, throughput and trace for each service in the list.

Services				
Ω Trace ID, trace group name, service	📋 🗸 Last 5 years	Show da	tes C	Refres
+ Add filter				
Services (7)				
Name 个	Average latency (ms)	Error rate	Throughput	Traces
frontend0	0.04	50%	746,657	685,7 32
frontend1	0.04	50%	52,000	39,96 9
frontend10	0.04	50%	52,000	39,73 8
frontend11	0.04	50%	52,000	39,55 6
frontend12	0.04	50%	52,000	40,00 4
frontend13	0.04	50%	52,000	40,38 1
frontend14	0.04	50%	52,000	39,88

### **Get Involved**

Code of Conduct Forums Github **Community Projects** 

Resources FAQ Testimonials Brand Guidelines Trademark Usage Policy **OpenSearch Disambiguation** 

# Connect

Connect



© 2023 OpenSearch contributors. OpenSearch is a registered trademark of Amazon Web Services.

© 2005-2021 Django Software Foundation and individual contributors. Django is a registered trademark of the Django Software

Foundation. This website was forked from the BSD-licensed <u>djangoproject.com</u> originally designed by <u>Threespot</u> & <u>andrevv</u>. We  $\heartsuit$  Django and the Django community. If you need a <u>high-level Python framework</u>, check it out.