SEARCH PLUGINS

REPLICATION PLUGIN

OBSERVABILITY PLUGIN

Event analytics

Notebooks

Trace analytics

Get Started

OpenSearch

data

Log analytics

Dashboards plugin

Analyze Jaeger trace

Application analytics

ML COMMONS PLUGIN

NEURAL SEARCH PLUGIN

MONITORING PLUGINS

NOTIFICATIONS PLUGIN

Operational panels

About Observability

Observability security

INDEX MANAGEMENT PLUGIN

Trace analytics / Analyze Jaeger trace data **Analyze Jaeger trace data** INTRODUCED 2.5

(OTEL) formatted 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

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

Q Search...

and look into the span details of a trace to pinpoint any service issues.

Download About Community Documentation

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. endpoint data.

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

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 otel-v1-

apm-span-* Or otel-v1-apm-service-map*. Jaeger indexes follow the naming conventions jaeger-span-* or jaeger-service-*.

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

How to set up OpenSearch to use Jaeger data

trace data.

version: '3' services: opensearch-node1: # This is also the hostname of the container within the Docker netv image: opensearchproject/opensearch:latest # Specifying the latest available image container_name: opensearch-node1

ES_TAGS_AS_FIELDS_ALL environment variable set to true to enable errors to be added to

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

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 wher - cluster.initial_cluster_manager_nodes=opensearch-node1,opensearch-node2 # Nodes

bootstrap.memory_lock=true # Disable JVM heap memory swapping - "OPENSEARCH_JAVA_OPTS=-Xms512m -Xmx512m" # Set min and max JVM heap sizes to at 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 at hard: 65536 volumes: - opensearch-data1:/usr/share/opensearch/data # Creates volume called opensearchports: - "9200:9200" - "9600:9600" networks: - opensearch-net # All of the containers will join the same Docker bridge network opensearch-node2: image: opensearchproject/opensearch:latest # This should be the same image used for 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 or 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-node2:926 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"] - "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

five combinations of services and operations that have a non-zero error rate. **Dashboard** Jaeger ~

Trace ID, trace group name, service name

2021

Step 2: View trace data in OpenSearch Dashboards

Use trace analytics in OpenSearch Dashboards

Trace analytics

Dashboard

ے

er time

After you select Jaeger for the data source, you can view all of the indexed data in

You can view the trace error count over time in the **Dashboard** view and also see the top

Now: 51.15%

2021

♂ Refresh

Throughput

Traces ©

89,774

88,449

89,586

89,349

4,016

♂ Refresh

Throughput

Traces ©

89,774

91,358

■ >= 95 percentile

100%

0%

10:08:12

10:08:12

Timeline Span list

12/16/2022

0

0.04

0.04

2021

Error rate ©

100%

100%

100%

100%

100%

Show dates

Error rate

2021

Jaeger v

CHOOSE DATA TYPE

Q Filter options

Jaeger

Data Prepper

Dashboard view, including Error rate and Throughput.

2021

To get started, see Get started with trace analytics.

trace data.

Data sources

Observability

Trace analytics

Traces

Notebooks

Error rate

+ Add filter

Trace error rate over time

Services

Event analytics

Metrics analytics

Operational panels

Dashboard view

Application analytics

OpenSearch Dashboards

Observability

Go to Dashboards **Trace analytics** at

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

To analyze the Jaeger trace data in Dashboards, first set up the trace analytics functionality.

You can specify either Data Prepper or Jaeger as the data source when you perform trace

Dashboard

oup name, service name

analytics. From Dashboards, go to **Observability > Trace analytics** and select Jaeger.

http://localhost:5601/app/observability-dashboards#/trace_analytics/home.

Dashboard

C Trace ID, trace group name, service name C Refresh Show dates process.serviceName: frontend0 × operationName: HTTP GET /config0 × + Add filter Traces (3000) Trace ID ↑ Latency (ms) Errors Last updated 12/16/2022 00de6a9aaf045bd400 a 0.04 10:08:12 12/16/2022

"_id": "_gaRlIU8yJaFH6S_Qt1g" "_score": 12.914443, "traceID": "00de6a9aaf045bd400" "spanID": "00de6a9aaf045bd400" "flags": 1 "operationName": "HTTP GET /config0",
"references": [],
"startTime": 1671214892597974, "startTimeMillis": 1671214892597 "tags": [], "http@status_code": 200 "http@url": "/config?nonse=0.8809634976926217"
"internal@span@format": "proto", 'sampler@type': "const 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,

52,000 0.04 50% 50% 0.04 52,000 50% 0.04 52,000 0.04 50% 52,000 39,88 0.04 50% Connect Resources

Top 5 Service and Operation Errors (5) Service and Operation Name © Average latency (ms) © 0.04 frontend0,HTTP GET /config0 frontend0,HTTP GET /config2 0.04 frontend0,HTTP GET /config4 0.04 0.04 frontend0,HTTP GET /config6 0.04 frontend0,HTTP GET /config8 **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. Jaeger v Trace ID, trace group name, service name + Add filter Traces over time Now: 1,306,425 Top 5 Service and Operation Latency (5) < 95 percentile</p> Service and Operation Name

O Latency variance (ms) 0 0.2 0.4 0.6 0.8 frontend0,HTTP GET /config0 frontend0,HTTP GET /config1 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** Jaeger v Œ 00de6a9aaf045bd4010 0.04 00de6a9aaf045bd40100 a 0.04 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. ☐ Observability Trace analytics Traces 00de6a9aaf045bd400 00de6a9aaf045bd400 Overview Trace ID Latency Errors 00de6a9aaf045bd400 📑 0.04 Last updated 12/16/2022 10:08:12 Time spent by service Spans (3) HTTP GET /config Payload **Services** error rate, throughput and trace for each service in the list. Services Jaeger ~ Show dates + Add filter Services (7) Name 1 Average latency (ms) frontend0 0.04 50% 50% frontend1 0.04

> Connect **Testimonials Brand Guidelines** Trademark Usage Policy

Get Involved

Forums

Github

Code of Conduct

Community Projects

Foundation.

FAQ

OpenSearch Disambiguation

© 2023 OpenSearch contributors. OpenSearch is a <u>registered trademark</u> of <u>Amazon Web Services</u>. © 2005-2021 <u>Django Software Foundation</u> and individual contributors. Django is a <u>registered trademark</u> of the Django Software 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.

C Refresh Error rate Throughput Traces 685,7 746,657 52,000 39,96 frontend10 52,000 6 39,55 frontend11 frontend12 frontend13

frontend14