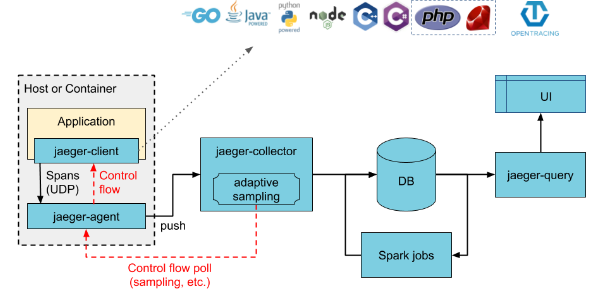
Distributed tracing system [Jaeger](https://www.jaegertracing.io/) and how it helps in troubleshooting microservices-based distributed systems.

When it comes to developing applications within the microservice architecture, the number of microservices can grow quickly. Managing microservices becomes harder with each new or updated microservice. When an application experiences a slowdown and its "data flow" goes through several different microservices, pinpointing the exact location of a slowdown may be difficult for a developer.

Let us take the example of a service like renting a movie on iTunes (or other movie rental service). A service like this requires many other microservices to check that the movie is available, proper payment credentials are received, and enough space exists on the viewer’s device for download. If either one of those microservice fail, then the entire transaction fails. In such a case, having logs just for the main rental service wouldn’t be very useful for debugging. However, if you were able to analyze each service you wouldn’t have to scratch your head to troubleshoot which microservice failed and what made it fail.

In real life, applications are even more complex and with the increasing complexity of applications, monitoring the applications has been a tedious task. Opentracing helps us to easily monitor:

**Architecture**



### Agent

The Jaeger **agent** is a network daemon that listens for spans sent over UDP, which it batches and sends to the collector. It is designed to be deployed to all hosts as an infrastructure component. The agent abstracts the routing and discovery of the collectors away from the client.

### Collector

The Jaeger **collector** receives traces from Jaeger [**agents**](https://www.jaegertracing.io/docs/1.14/architecture#agent) and runs them through a processing pipeline. Currently our pipeline validates traces, indexes them, performs any transformations, and finally stores them.

Jaeger’s storage is a pluggable component which currently supports [**Cassandra**](https://www.jaegertracing.io/docs/1.14/deployment#cassandra), [**Elasticsearch**](https://www.jaegertracing.io/docs/1.14/deployment#elasticsearch) and [**Kafka**](https://www.jaegertracing.io/docs/1.14/deployment#kafka).

### Query

**Query** is a service that retrieves traces from storage and hosts a UI to display them

===========================================================

**All in One**

All-in-one is an executable designed for quick local testing, launches the Jaeger UI, collector, query, and agent, with an in memory storage component.

docker run -d --name jaeger \

-e COLLECTOR\_ZIPKIN\_HTTP\_PORT=9411 \

-p 5775:5775/udp \

-p 6831:6831/udp \

-p 6832:6832/udp \

-p 5778:5778 \

-p 80:16686 \

-p 14268:14268 \

-p 9411:9411 \

jaegertracing/all-in-one:1.9

The Jaeger GUI is now accessible on <http://localhost:16686>. Here I given 80 port

The container exposes the following ports:

| **Port** | **Protocol** | **Component** | **Function** |
| --- | --- | --- | --- |
| 5775 | UDP | agent | accept zipkin.thrift over compact thrift protocol (deprecated, used by legacy clients only) |
| 6831 | UDP | agent | accept jaeger.thrift over compact thrift protocol |
| 6832 | UDP | agent | accept jaeger.thrift over binary thrift protocol |
| 5778 | HTTP | agent | serve configs |
| 16686 | HTTP | query | serve frontend |
| 14268 | HTTP | collector | accept jaeger.thrift directly from clients |
| 14250 | HTTP | collector | accept model.proto |
| 9411 | HTTP | collector | Zipkin compatible endpoint (optional) |

## Sample App: HotROD

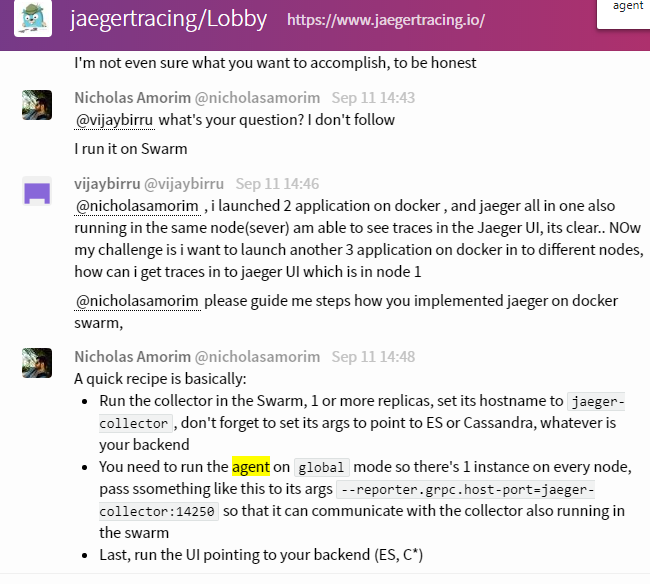
HotROD (Rides on Demand) is a demo application that consists of several microservices

docker run -d --link jaeger --env JAEGER\_AGENT\_HOST=jaeger --env JAEGER\_AGENT\_PORT=6831 -p8080-8083:8080-8083 jaegertracing/example-hotrod:latest all

Then navigate to <http://localhost:8080>

This above setup is only for single node, not applicable to multiple nodes , like docker swarm

The below discussion on <https://gitter.im/jaegertracing/Lobby> ,where i asked question



Now 2 nd method approach as he said

Elasticsearch

1.docker run -d --name elasticsearch --restart=always -p 8080:9200 -p 8090:9300 -e ES\_JAVA\_OPTS="-Xms512m -Xmx512m" elasticsearch:5.6.4

Collector

2.docker run -d --name jaeger-collector -e SPAN\_STORAGE\_TYPE=elasticsearch -e ES\_SERVER\_URLS=http://10.229.161.198:8080 -e ES\_USERNAME=elastic -p 14267:14267 -p 14268:14268 -p 9411:9411 jaegertracing/jaeger-collector

UI

3.docker run -d --name jaeger-query -e SPAN\_STORAGE\_TYPE=elasticsearch -e ES\_SERVER\_URLS=http://10.229.161.198:8080 -p 80:16686/tcp jaegertracing/jaeger-query

Agent must be on all nodes

docker run -d --name jaeger-agent -p 5775:5775/udp -p 6831:6831/udp -p 6832:6832/udp -p 5778:5778/tcp jaegertracing/jaeger-agent:1.14 --collector.host-port=10.229.161.198:14267

The above 1,2,3 steps in node1 and 4th step in all nodes along with the sample application

Now switching to node2

Agent

docker run -d --name jaeger-agent -p 5775:5775/udp -p 6831:6831/udp -p 6832:6832/udp -p 5778:5778/tcp jaegertracing/jaeger-agent:1.14 --collector.host-port=10.229.161.198:14267

Sample Application

docker run -d --link jaeger-agent --env JAEGER\_AGENT\_HOST=jaeger-agent --env JAEGER\_AGENT\_PORT=6831 -p8080-8083:8080-8083 jaegertracing/example-hotrod:latest all

Now am able to see traces in the UI which is in node1

now I want to launch this application in docker swarm

creating service in docker swarm

docker service create --name service1 --replicas 3 --link jaeger-agent --env JAEGER\_AGENT\_HOST=jaeger-agent --env JAEGER\_AGENT\_PORT=6831 -p8080-8083:8080-8083 jaegertracing/example-hotrod:latest all  
unknown flag: --link  
See 'docker service create --help'.

-----docker compose its works in node2---------

This compose file will launch agent along with application

version: '3.7'

services:

jaeger:

image: jaegertracing/jaeger-agent:1.14

command: ["--collector.host-port=10.229.161.198:14267"]

ports:

- "5775:5775/udp"

- "6831:6831/udp"

- "6832:6832/udp"

- "5778:5778"

hotrod:

image: jaegertracing/example-hotrod:latest

ports:

- "8080:8080"

environment:

- JAEGER\_AGENT\_HOST=jaeger

- JAEGER\_AGENT\_PORT=6831

docker stack deploy --compose-file docker-compose.yml servicename