# **Jaeger:**

Jaeger is a distributed tracing system released as open source by [**Uber Technologies**](http://uber.github.io/). It is used for monitoring and troubleshooting microservices based distributed systems, including:

* Distributed context propagation
* Distributed transaction monitoring
* Root cause analysis
* Service dependency analysis
* Performance / latency optimization

# **Major Components of Jaeger:**

# **Jaeger Client Libraries**— Jaeger clients are language-specific implementations of the [OpenTracing API](http://opentracing.io/). The OpenTracing API provides a standard, vendor-neutral framework for instrumentation. This means that if a developer wants to try out a different distributed tracing system, then instead of repeating the whole instrumentation process for the new distributed tracing system, the developer can simply change the configuration of the Tracer.

**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 and runs them through a processing pipeline. Currently, the pipeline validates traces, indexes them, performs transformations, and finally, stores them. Jaeger’s storage is a pluggable component which currently supports [Cassandra](https://www.jaegertracing.io/docs/1.8/deployment#cassandra), [Elasticsearch](https://www.jaegertracing.io/docs/1.8/deployment" \l "elasticsearch), and [Kafka](https://www.jaegertracing.io/docs/1.8/deployment#kafka).

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

**Ingester** — Ingester is a service that reads from Kafka topic and writes to another storage backend (Cassandra, Elasticsearch).

# **Problem that Jaeger addresses:**

#### Distributed transaction monitoring

#### Performance and latency optimization

#### Root cause analysis

#### Service dependency analysis

#### Distributed context propagation

# **How to setup Jaeger?**

For setup of Jaeger of we need some extra tools like Docker (to get the image of jaeger) and Kubernetes (to manage the container).

Step1: Setup Docker in VM (by exe or bin file).

Step 2: Run this command –

docker run -d --name jaeger -e COLLECTOR\_ZIPKIN\_HOST\_PORT=:9411 -p 5775:5775/udp -p 6831:6831/udp -p 6832:6832/udp -p 5778:5778 -p 16686:16686 -p 14268:14268 -p 14250:14250 -p 9411:9411 jaegertracing/all-in-one:1.9

Here we are configuring the Jaeger (version 1.9 ) with Zipkin images on respectively port 16686 , 9411. These are the by default port.

Step 3 : Add jaeger dependency in pom.xml file of the project.

<dependency>

    <groupId>io.opentracing.contrib</groupId>

    <artifactId>opentracing-spring-jaeger-web-starter</artifactId>

    <version>3.3.1</version>

</dependency>

Dependency should be added in each microservices.

Step 4 : Configure Jaeger into the project.

@Bean

public JaegerTracer jaegerTracer() {

  return new io.jaegertracing.Configuration("jaeger-client")

      .withSampler(new io.jaegertracing.Configuration.SamplerConfiguration().withType(ConstSampler.TYPE)

      .withParam(1))

      .withReporter(new io.jaegertracing.Configuration.ReporterConfiguration().withLogSpans(true))

      .getTracer();

}

Configuration should be done in every microservices which we want to trace.

Step 5 : Up all the services. To trace the detail we can go to Jaeger port (by default 16686) which we define earlier.

You can see this type of UI after setup everyting.

