**Steps to Implement Distributed Tracing**

**1. Add OpenTelemetry Tracing in Spring Boot**

1. **Add Dependencies** for OpenTelemetry and Micrometer Tracing in pom.xml:

xml

Copy code

<dependency>

<groupId>io.opentelemetry.instrumentation</groupId>

<artifactId>opentelemetry-spring-boot-starter</artifactId>

<version>1.12.0</version> <!-- Ensure the version is compatible -->

</dependency>

<dependency>

<groupId>io.opentelemetry</groupId>

<artifactId>opentelemetry-exporter-otlp</artifactId>

<version>1.12.0</version>

</dependency>

1. **Configure OpenTelemetry Exporter**:

In your application.properties (or application.yaml), configure the OpenTelemetry exporter to send traces to a tracing backend (like Jaeger or Tempo):

properties

Copy code

# OTLP exporter endpoint - Jaeger, Tempo, or any OpenTelemetry-compatible backend

otel.traces.exporter=otlp

otel.exporter.otlp.endpoint=http://tempo:4317 # Replace with 'jaeger:14250' for Jaeger in OTLP

otel.resource.attributes=service.name=spring-boot-app

Here, http://tempo:4317 is the endpoint for Grafana Tempo if you’re using it in Kubernetes. Adjust for Jaeger if necessary.

1. **Enable Tracing in Kubernetes**:

Deploy the Spring Boot app with otel configurations for tracing. Tracing information will propagate as traceId and spanId headers between services, allowing for distributed trace correlation.

**2. Deploy Jaeger or Grafana Tempo in Kubernetes**

You need a tracing backend in Kubernetes, such as **Jaeger** or **Grafana Tempo**, to collect and visualize the trace data.

**Option A: Deploy Jaeger**

1. **Install Jaeger using Helm**:

bash

Copy code

helm repo add jaegertracing https://jaegertracing.github.io/helm-charts

helm install jaeger jaegertracing/jaeger --namespace tracing

1. **Expose Jaeger UI**:

Expose the Jaeger UI using a LoadBalancer or Ingress to access it from outside the cluster:

yaml

Copy code

apiVersion: v1

kind: Service

metadata:

name: jaeger-query

namespace: tracing

spec:

type: LoadBalancer

ports:

- port: 80

targetPort: 16686

selector:

app.kubernetes.io/name: jaeger

app.kubernetes.io/component: query

This will make the Jaeger UI accessible at http://<external-ip> for visualizing traces.

**Option B: Deploy Grafana Tempo**

1. **Install Tempo using Helm**:

bash

Copy code

helm repo add grafana https://grafana.github.io/helm-charts

helm install tempo grafana/tempo --namespace tracing

1. **Configure Grafana with Tempo**:
   * In Grafana, add Tempo as a **data source**.
   * Set the URL to http://tempo:3100 (or the internal service URL for Tempo in Kubernetes).

**3. Propagate Trace Context Across Services**

To enable tracing across different services in the Kubernetes cluster, configure your Spring Boot applications to propagate trace context using Spring Cloud Sleuth and OpenTelemetry.

1. **Add Spring Cloud Sleuth Dependencies** (Optional):

Sleuth automatically propagates tracing headers like traceId and spanId across services.

xml

Copy code

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-sleuth</artifactId>

<version>3.0.4</version> <!-- Match Spring Boot compatibility -->

</dependency>

1. **Enable Context Propagation**:

By default, Sleuth propagates trace context with OpenTelemetry. Ensure all microservices have Sleuth and OpenTelemetry enabled, allowing traces to flow seamlessly through the services.

**4. Visualize Traces in Grafana**

1. **Set Up Grafana for Trace Visualization**:
   * In Grafana, go to **Configuration > Data Sources > Add data source** and select **Tempo** or **Jaeger** based on your setup.
   * Configure the data source to point to the correct service URL (e.g., http://tempo:3100 for Tempo or http://jaeger-query:16686 for Jaeger).
2. **Create Trace Dashboards**:
   * Grafana supports trace dashboards to visualize latency, spans, and trace trees.
   * You can use built-in templates or create custom dashboards to track request flow across microservices.
3. **Enable Logs Correlation (Optional)**:
   * Configure Grafana to correlate logs with traces by enabling trace context in log statements.
   * In application.properties, add trace ID to your logging pattern for better trace-log correlation:

properties

Copy code

logging.pattern.level=%traceId - %spanId - %message

**Summary**

1. **OpenTelemetry in Spring Boot**: Use OpenTelemetry SDK to generate and export traces.
2. **Jaeger or Tempo in Kubernetes**: Deploy Jaeger or Tempo as the trace collection backend.
3. **Trace Propagation with Sleuth**: Use Sleuth to propagate trace context across services.
4. **Trace Visualization in Grafana**: Configure Grafana to visualize traces from Jaeger or Tempo and correlate them with logs.

This setup enables full traceability across microservices, allowing for real-time monitoring, performance debugging, and bottleneck analysis in your Spring Boot application on Kubernetes.