Using OpenTelemetry Collector

Let's modify our setup to use an OpenTelemetry Collector instead of sending traces directly to Jaeger. This approach provides more flexibility and allows for additional processing of telemetry data. Here's a step-by-step guide to implement this:

1. Set up the OpenTelemetry Collector:

First, we need to configure and run the OpenTelemetry Collector. Create a file named `otel-collector-config.yaml`:

otel-collector-config.yaml

```
receivers:
  otlp:
    protocols:
      grpc:
      http:

processors:
  batch:

exporters:
  otlp:
    endpoint: "jaeger:4317"
    tls:
      insecure: true

service:
  pipelines:
    traces:
      receivers: [otlp]
      processors: [batch]
      exporters: [otlp]
```

2. Update the `docker-compose.yml` file to include the OpenTelemetry Collector:

Updated docker-compose.yml

```
version: '3'
services:
  jaeger:
    image: jaegertracing/all-in-one:latest
    ports:
      - "16686:16686"   # Web UI
      - "14250:14250"   # gRPC for Jaeger-to-Jaeger communication
      - "4317:4317"     # OTLP gRPC receiver
      - "4318:4318"     # OTLP HTTP receiver
    environment:
      - COLLECTOR_OTLP_ENABLED=true

  otel-collector:
    image: otel/opentelemetry-collector:0.88.0
    command: ["--config=/etc/otel-collector-config.yaml"]
    volumes:
      - ./otel-collector-config.yaml:/etc/otel-collector-config.yaml
    ports:
      - "4319:4317"    # OTLP gRPC receiver
      - "4320:4318"    # OTLP HTTP receiver
    depends_on:
      - jaeger
```

3. Update the `application.properties` file in your Spring Boot application:

Updated application.properties

```
# Existing properties
spring.application.name=fx-catalog-service
server.port=8002
course.service.url=http://localhost:8001
spring.main.allow-bean-definition-overriding=true

# Updated OpenTelemetry configuration
otel.exporter.otlp.endpoint=http://localhost:4317
otel.exporter.otlp.protocol=grpc
otel.traces.exporter=otlp
otel.metrics.exporter=none
otel.logs.exporter=none
```

4. Ensure your `pom.xml` has the necessary OpenTelemetry dependencies from the previous exercise:

5. Update your `OpenTelemetryConfig` class:

Updated OpenTelemetryConfig.java

```
import io.opentelemetry.api.OpenTelemetry;
import io.opentelemetry.api.common.Attributes;
import io.opentelemetry.api.trace.Tracer;
import io.opentelemetry.api.trace.propagation.W3CTraceContextPropagator;
import io.opentelemetry.context.propagation.ContextPropagators;
import io.opentelemetry.exporter.otlp.trace.OtlpGrpcSpanExporter;
import io.opentelemetry.sdk.OpenTelemetrySdk;
import io.opentelemetry.sdk.resources.Resource;
import io.opentelemetry.sdk.trace.SdkTracerProvider;
import io.opentelemetry.sdk.trace.export.BatchSpanProcessor;
import io.opentelemetry.semconv.ResourceAttributes;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class OpenTelemetryConfig {

    @Value("${spring.application.name}")
    private String applicationName;

    @Bean
    public OpenTelemetry openTelemetry() {
        Resource resource = Resource.getDefault()
                .merge(Resource.create(Attributes.of(ResourceAttributes.SERVICE_NAME,
applicationName)));

        OtlpGrpcSpanExporter spanExporter = OtlpGrpcSpanExporter.builder()
                .setEndpoint("http://localhost:4317")
                .build();
        SdkTracerProvider sdkTracerProvider = SdkTracerProvider.builder()
                .addSpanProcessor(BatchSpanProcessor.builder(spanExporter).build())
                .setResource(resource)
                .build();

        return OpenTelemetrySdk.builder()
                .setTracerProvider(sdkTracerProvider)

.setPropagators(ContextPropagators.create(W3CTraceContextPropagator.getInstance()))
                .buildAndRegisterGlobal();
    }

    @Bean
    public Tracer tracer(OpenTelemetry openTelemetry) {
        return openTelemetry.getTracer(applicationName);
    }
}
```

6. Start the Docker containers:

```
docker-compose up -d
```

7. Generate some traffic by accessing your application's endpoints.
8. View the traces in Jaeger UI (http://localhost:16686).

By following these steps, your Spring Boot application will now send traces to the OpenTelemetry Collector, which will then forward them to Jaeger. This setup allows for more flexibility in processing and routing telemetry data.

Some benefits of using the Collector:

1. It can receive data in multiple formats and export to various backends.
2. It can perform data processing, such as batching and filtering.
3. It provides a buffer between your application and the backend, improving reliability.