Using OpenTelemetry API

OpenTelemetry supports both services and send traces directly to Jaeger without a Collector, you'll need to make several changes to both the FutureXCourseApp and FutureXCourseCatalog services. Here's a step-by-step guide on what you need to do:

1. Update dependencies in both services' pom.xml files:

Updated pom.xml for both services, according to https://opentelemetry.io/docs/zero-code/java/spring-boot-starter/getting-started/

OpenTelemetry BOMs using Maven:

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>io.opentelemetry.instrumentation</groupId>
            <artifactId>opentelemetry-instrumentation-bom</artifactId>
            <version>2.6.0</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

The OpenTelemetry starter uses OpenTelemetry Spring Boot:

```
<dependency>
    <groupId>io.opentelemetry.instrumentation</groupId>
    <artifactId>opentelemetry-spring-boot-starter</artifactId>
</dependency>
```

2. Create a configuration class in both services to set up OpenTelemetry:

OpenTelemetryConfig.java for both services

```
package com.futurex.services.FutureXCourseApp;

import io.opentelemetry.api.OpenTelemetry;
import io.opentelemetry.api.common.Attributes;
import io.opentelemetry.api.trace.Tracer;
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")  // Updated to use the OTLP gRPC port
                .build();

        SdkTracerProvider sdkTracerProvider = SdkTracerProvider.builder()
                .addSpanProcessor(BatchSpanProcessor.builder(spanExporter).build())
                .setResource(resource)
                .build();

        return OpenTelemetrySdk.builder()
                .setTracerProvider(sdkTracerProvider)
                .setPropagators(ContextPropagators.noop())
                .buildAndRegisterGlobal();
    }

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

3. Update the main application classes in both services to use OpenTelemetry:

For FutureXCourseApp:

Updated FutureXCourseAppApplication.java

```
import io.opentelemetry.api.OpenTelemetry;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
public class FutureXCourseAppApplication {

    public static void main(String[] args) {

        SpringApplication.run(FutureXCourseAppApplication.class, args);
    }

    @Bean
    public RestTemplate restTemplate(OpenTelemetry openTelemetry) {
        return new RestTemplate();
    }
}
```

For FutureXCourseCatalog:

Updated FutureXCourseCatalogApplication.java

```
import io.opentelemetry.api.OpenTelemetry;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
public class FutureXCourseCatalogApplication {

    public static void main(String[] args) {
        SpringApplication.run(FutureXCourseCatalogApplication.class, args);
    }

    @Bean
    public RestTemplate restTemplate(OpenTelemetry openTelemetry) {
        return new RestTemplate();
    }
}
```

4. Update the controllers in both services to use OpenTelemetry for tracing:

For CourseController in FutureXCourseApp:

Updated CourseController.java

```
import io.opentelemetry.api.trace.Span;
import io.opentelemetry.api.trace.Tracer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.math.BigInteger;
import java.util.List;

@RestController
public class CourseController {

    @Autowired
    private CourseRepository courseRepository;

    @Autowired
    private Tracer tracer;

    @RequestMapping("/")
    public String getCourseAppHome() {
        Span span = tracer.spanBuilder("getCourseAppHome").startSpan();
        try {
            return "Course App Home";
        } finally {
            span.end();
        }
    }

    @RequestMapping("/courses")
    public List<Course> getCourses() {
        Span span = tracer.spanBuilder("getCourses").startSpan();
        try {
            return courseRepository.findAll();
        } finally {
            span.end();
```

```
        }
    }

    @RequestMapping("/{id}")
    public Course getSpecificCourse(@PathVariable("id") BigInteger id ) {
        return courseRepository.getOne(id);
    }

    @RequestMapping(method = RequestMethod.POST, value="/courses")
    public void saveCourse(@RequestBody Course course) {
        courseRepository.save(course);
    }

    @RequestMapping(method = RequestMethod.DELETE,value = "{id}")
    public void deleteCourse(@PathVariable BigInteger id) {
        courseRepository.deleteById(id);
    }


}
```

For CatalogController in FutureXCourseCatalog:

Updated CatalogController.java

```
import io.opentelemetry.api.trace.Span;
import io.opentelemetry.api.trace.Tracer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
public class CatalogController {

    @Value("${course.service.url}")
    private String courseServiceUrl;

    private final RestTemplate restTemplate;

    @Autowired
    private Tracer tracer;

    public CatalogController(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    @RequestMapping("/")
    public String getCatalogHome() {
        Span span = tracer.spanBuilder("getCatalogHome").startSpan();
        try {
            String courseAppMessage = restTemplate.getForObject(courseServiceUrl, String.class);
            return "Welcome to FutureX Course Catalog " + courseAppMessage;
        } finally {
            span.end();
        }
    }

    @RequestMapping("/catalog")
    public String getCatalog() {
        Span span = tracer.spanBuilder("getCatalog").startSpan();
        try {
            String courses = restTemplate.getForObject(courseServiceUrl + "/courses",
String.class);
            return "Our courses are " + courses;
        } finally {
            span.end();
        }
```

```
    }

    @RequestMapping("/firstcourse")
    public String getSpecificCourse() {
        Span span = tracer.spanBuilder("getSpecificCourse").startSpan();
        try {
            Course course = restTemplate.getForObject(courseServiceUrl + "/1", Course.class);
            return "Our first course is " + course.getCoursename();
        } finally {
            span.end();
        }
    }
}
```

5. To install Jaeger using the configuration you provided, create a docker-compose.yml file in the root of your project:

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
```

To run Jaeger, use the command: docker-compose up -d

After making these changes, both services will use OpenTelemetry to send traces directly to Jaeger. You can view the traces in the Jaeger UI by navigating to http://localhost:16686 in your web browser.

Remember to restart both services after making these changes. The traces will be visible in Jaeger once you start making requests to your services.