Quarkus - Using Apache Tika

This guide explains how your Quarkus application can use Apache Tika to parse the documents.

Apache Tika is a content analysis toolkit which is used to parse the documents in PDF, Open Document, Excel and many other well known binary and text formats using a simple uniform API. Both the document text and properties (metadata) are available once the document has been parsed.

> If you are planning to run the application as a native executable and parse documents that may have been created with charsets different than the standard ones supported in Java such as UTF-8 then you should configure Quarkus Maven Plugin to get the native image generator include all the charsets available to the JVM:

```xml
<plugin>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-maven-plugin</artifactId>
    <executions>
        <execution>
            <id>native-image</id>
            <goals>
                <goal>native-image</goal>
            </goals>
            <configuration>
                <addAllCharsets>true</addAllCharsets>
                ...
            </configuration>
        </execution>
    </executions>
</plugin>
```

Prerequisites

To complete this guide, you need:

- less than 20 minutes

- an IDE

- JDK 1.8+ installed with JAVA_HOME configured appropriately

- Apache Maven 3.5.3+

- Docker

- GraalVM installed if you want to run in native mode

Solution

We recommend that you follow the instructions in the next sections and create the application step by step. However, you can go right to the completed example.

Clone the Git repository: `git clone https://github.com/quarkusio/quarkus-quickstarts.git`, or download an archive.

The solution is located in the `tika-quickstart` directory.

> ℹ️ The provided solution contains a few additional elements such as tests and testing infrastructure.

Creating the Maven Project

First, we need a new project. Create a new project with the following command:

```
mvn io.quarkus:quarkus-maven-plugin:1.3.0.Alpha2:create \
    -DprojectGroupId=org.acme.example \
    -DprojectArtifactId=tika-quickstart \
    -DclassName="org.acme.quickstart.tika.TikaParserResource" \
    -Dpath="/parse" \
    -Dextensions="tika,resteasy"
cd tika-quickstart
```

This command generates a Maven project, importing the `tika` and `resteasy` extensions.

If you already have your Quarkus project configured you can add the `tika` and `resteasy` extensions to your project by running the following command in your project base directory.

```
./mvnw quarkus:add-extension -Dextensions="tika,resteasy"
```

This will add the following to your pom.xml:

```xml
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-tika</artifactId>
</dependency>
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-resteasy</artifactId>
</dependency>
```

Examine the generated JAX-RS resource

Open the `src/main/java/org/acme/quickstart/tika/TikaParserResource.java` file and see the following content:

```java
package org.acme.quickstart.tika;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/parse")
public class TikaParserResource {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return "hello";
    }
}
```

Update the JAX-RS resource

Next update `TikaParserResource` to accept and parse PDF and OpenDocument format documents:

```java
package org.acme.quickstart.tika;

import java.io.InputStream;
import java.time.Duration;
import java.time.Instant;

import javax.inject.Inject;
import javax.ws.rs.Consumes;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

import io.quarkus.tika.TikaParser;
import org.jboss.logging.Logger;

@Path("/parse")
public class TikaParserResource {
    private static final Logger log = Logger.getLogger
(TikaParserResource.class);

    @Inject
    TikaParser parser;

    @POST
    @Path("/text")
    @Consumes({"application/pdf",
"application/vnd.oasis.opendocument.text"})
    @Produces(MediaType.TEXT_PLAIN)
    public String extractText(InputStream stream) {
        Instant start = Instant.now();

        String text = parser.getText(stream);

        Instant finish = Instant.now();

        log.info(Duration.between(start, finish).toMillis() + " mls
have passed");

        return text;
    }
}
```

As you can see the JAX-RS resource method was renamed to `extractText`, `@GET` annotation was replaced with `POST` and `@Path(/text)` annotation was added, and `@Consumes` annotation shows that PDF and OpenDocument media type formats can now be accepted. An injected `TikaParser` is used to parse the documents and report the extracted text. It also measures how long does it take to parse a given document.

Run the application

Now we are ready to run our application. Use:

```
./mvnw compile quarkus:dev
```

and you should see output similar to:

quarkus:dev Output

```
$ ./mvnw clean compile quarkus:dev
[INFO] Scanning for projects...
[INFO]
INFO] --------------------< org.acme.example:apache-tika
>--------------------
[INFO] Building apache-tika 1.0-SNAPSHOT
[INFO] ------------------------------[ jar
]------------------------------
...
Listening for transport dt_socket at address: 5005
2019-10-15 14:23:26,442 INFO  [io.qua.dep.QuarkusAugmentor] (main)
Beginning quarkus augmentation
2019-10-15 14:23:26,960 INFO  [io.qua.resteasy] (build-15) Resteasy
running without servlet container.
2019-10-15 14:23:26,960 INFO  [io.qua.resteasy] (build-15) - Add
quarkus-undertow to run Resteasy within a servlet container
2019-10-15 14:23:26,991 INFO  [io.qua.dep.QuarkusAugmentor] (main)
Quarkus augmentation completed in 549ms
2019-10-15 14:23:27,637 INFO  [io.quarkus] (main) Quarkus 999-
SNAPSHOT started in 1.361s. Listening on: http://0.0.0.0:8080
2019-10-15 14:23:27,638 INFO  [io.quarkus] (main) Profile dev
activated. Live Coding activated.
2019-10-15 14:23:27,639 INFO  [io.quarkus] (main) Installed
features: [cdi, resteasy, tika]
```

Now that the REST endpoint is running, we can get it to parse PDF and OpenDocument documents using a command line tool like curl:

```
$ curl -X POST -H "Content-type: application/pdf" --data-binary
@target/classes/quarkus.pdf http://localhost:8080/parse/text
Hello Quarkus
```

and

```
$ curl -X POST -H "Content-type: Content-type:
application/vnd.oasis.opendocument.text" --data-binary
@target/classes/quarkus.odt http://localhost:8080/parse/text
Hello Quarkus
```

Building a native executable

You can build a native executable with the usual command `./mvnw package -Pnative`. Running it is as simple as executing `./target/tika-quickstart-1.0-SNAPSHOT-runner`.

Configuration Reference

📦 Configuration property fixed at build time - ⚙️ Configuration property overridable at runtime

Configuration property	Type	Default
📦 quarkus.tika.tika-config-path The resource path within the application artifact to the `tika-config.xml` file.	string	
📦 quarkus.tika.parsers Comma separated list of the parsers which must be supported. Most of the document formats recognized by Apache Tika are supported by default but it affects the application memory and native executable sizes. One can list only the required parsers in `tika-config.xml` to minimize a number of parsers loaded into the memory, but using this property is recommended to achieve both optimizations. Either the abbreviated or full parser class names can be used. Only PDF and OpenDocument format parsers can be listed using the reserved 'pdf' and 'odf' abbreviations. Custom class name abbreviations have to be used for all other parsers. For example: // Only PDF parser is required: quarkus.tika.parsers = pdf // Only PDF and OpenDocument parsers are required: quarkus.tika.parsers = pdf,odf This property will have no effect if the `tikaConfigPath' property has been set.	string	
📦 quarkus.tika.append-embedded-content Controls how the content of the embedded documents is parsed. By default it is appended to the master document content. Setting this property to false makes the content of each of the embedded documents available separately.	boolean	true
📦 quarkus.tika.parser-options Configuration of the individual parsers. For example: quarkus.tika.parsers = pdf,odf quarkus.tika.parser-options.pdf.sort-by-position = true	Map<String,Map<String,String>>	required ❗

quarkus.tika.parser		
Full parser class name for a given parser abbreviation. For example: quarkus.tika.parsers = classparser quarkus.tika.parser.classparser = org.apache.tika.parser.asm.ClassParser	`Map<String,String>`	required ❗