Sure! Here is the recommended project structure for your Spring Boot application, including Docker and Kubernetes configuration files.

**Project Structure**

css

Copy code

lambda-s3-documentum/

├── src/

│ ├── main/

│ │ ├── java/

│ │ │ └── com/

│ │ │ └── example/

│ │ │ └── lambdas3documentum/

│ │ │ ├── controller/

│ │ │ │ └── S3DocumentumController.java

│ │ │ ├── service/

│ │ │ │ └── DocumentumService.java

│ │ │ └── LambdaS3DocumentumApplication.java

│ │ └── resources/

│ │ ├── application.properties

│ │ └── logback-spring.xml

│ └── test/

│ └── java/

│ └── com/

│ └── example/

│ └── lambdas3documentum/

│ └── LambdaS3DocumentumApplicationTests.java

├── .dockerignore

├── .gitignore

├── Dockerfile

├── README.md

├── deployment.yaml

├── service.yaml

├── secret.yaml

└── pom.xml

**Project Files**

**1. LambdaS3DocumentumApplication.java**

java

Copy code

package com.example.lambdas3documentum;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LambdaS3DocumentumApplication {

public static void main(String[] args) {

SpringApplication.run(LambdaS3DocumentumApplication.class, args);

}

}

**2. S3DocumentumController.java**

java

Copy code

package com.example.lambdas3documentum.controller;

import com.example.lambdas3documentum.service.DocumentumService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api")

public class S3DocumentumController {

@Autowired

private DocumentumService documentumService;

@PostMapping("/process-files")

public String processFiles(@RequestParam String bucketName, @RequestParam String folder) {

try {

List<String> keys = documentumService.listFiles(bucketName, folder);

documentumService.streamFilesToDocumentum(keys, bucketName);

return "Files processed successfully";

} catch (Exception e) {

e.printStackTrace();

return "Error processing files: " + e.getMessage();

}

}

}

**3. DocumentumService.java**

java

Copy code

package com.example.lambdas3documentum.service;

import com.amazonaws.services.s3.AmazonS3;

import com.amazonaws.services.s3.model.S3Object;

import com.documentum.fc.client.DfClient;

import com.documentum.fc.client.IDfClient;

import com.documentum.fc.client.IDfSession;

import com.documentum.fc.client.IDfSessionManager;

import com.documentum.fc.common.DfLoginInfo;

import com.documentum.fc.common.IDfLoginInfo;

import com.documentum.fc.common.IDfSysObject;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import reactor.core.publisher.Flux;

import reactor.core.publisher.Mono;

import reactor.core.scheduler.Schedulers;

import java.io.ByteArrayInputStream;

import java.io.ByteArrayOutputStream;

import java.io.InputStream;

import java.util.List;

import java.util.stream.Collectors;

@Service

public class DocumentumService {

@Autowired

private AmazonS3 amazonS3;

public List<String> listFiles(String bucketName, String folder) {

return amazonS3.listObjects(bucketName, folder)

.getObjectSummaries()

.stream()

.map(s -> s.getKey())

.collect(Collectors.toList());

}

public void streamFilesToDocumentum(List<String> keys, String bucketName) {

try {

IDfClient client = DfClient.getLocalClient();

IDfSessionManager sessionManager = client.newSessionManager();

IDfLoginInfo loginInfo = new DfLoginInfo();

loginInfo.setUser("username");

loginInfo.setPassword("password");

sessionManager.setIdentity("repo", loginInfo);

Flux.fromIterable(keys)

.flatMap(key -> processFile(sessionManager, bucketName, key))

.doOnError(Throwable::printStackTrace)

.blockLast(); // Wait for all files to be processed

} catch (Exception e) {

e.printStackTrace();

}

}

private Mono<Void> processFile(IDfSessionManager sessionManager, String bucketName, String key) {

return Mono.fromCallable(() -> {

IDfSession session = sessionManager.getSession("repo");

S3Object s3Object = amazonS3.getObject(bucketName, key);

InputStream inputStream = s3Object.getObjectContent();

// Convert InputStream to byte array

byte[] content = toByteArray(inputStream);

saveFileToDocumentum(session, content, key);

sessionManager.release(session);

return Mono.empty();

}).subscribeOn(Schedulers.boundedElastic()).then();

}

private byte[] toByteArray(InputStream input) throws Exception {

ByteArrayOutputStream buffer = new ByteArrayOutputStream();

int nRead;

byte[] data = new byte[16384];

while ((nRead = input.read(data, 0, data.length)) != -1) {

buffer.write(data, 0, nRead);

}

buffer.flush();

return buffer.toByteArray();

}

private void saveFileToDocumentum(IDfSession session, byte[] content, String key) {

try {

// Assuming you have a method to create an object in Documentum

// Replace this with your actual Documentum file saving logic

// Example: Create a new document and set its content

IDfSysObject newDoc = (IDfSysObject) session.newObject("dm\_document");

newDoc.setObjectName(key);

newDoc.setContentType("application/octet-stream");

newDoc.setContent(new ByteArrayInputStream(content));

newDoc.save();

System.out.println("Saved file to Documentum: " + key);

} catch (Exception e) {

System.err.println("Error saving file to Documentum: " + key);

e.printStackTrace();

}

}

}

**4. application.properties**

properties

Copy code

# AWS S3 Configuration

cloud.aws.credentials.accessKey=${AWS\_ACCESS\_KEY\_ID}

cloud.aws.credentials.secretKey=${AWS\_SECRET\_ACCESS\_KEY}

cloud.aws.region.static=your-aws-region

# Documentum Configuration

documentum.username=${DOCUMENTUM\_USERNAME}

documentum.password=${DOCUMENTUM\_PASSWORD}

documentum.repo=${DOCUMENTUM\_REPO}

**5. Dockerfile**

Dockerfile

Copy code

# Use an official OpenJDK runtime as a parent image

FROM openjdk:11-jre-slim

# Set the working directory

WORKDIR /app

# Copy the jar file

COPY target/lambda-s3-documentum-1.0-SNAPSHOT.jar app.jar

# Run the jar file

ENTRYPOINT ["java", "-jar", "app.jar"]

**6. Kubernetes Configuration**

**deployment.yaml:**

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: s3-documentum-deployment

labels:

app: s3-documentum

spec:

replicas: 2

selector:

matchLabels:

app: s3-documentum

template:

metadata:

labels:

app: s3-documentum

spec:

containers:

- name: s3-documentum

image: your-docker-repo/s3-documentum:latest

ports:

- containerPort: 8080

env:

- name: AWS\_ACCESS\_KEY\_ID

valueFrom:

secretKeyRef:

name: aws-credentials

key: accesskey

- name: AWS\_SECRET\_ACCESS\_KEY

valueFrom:

secretKeyRef:

name: aws-credentials

key: secretkey

- name: DOCUMENTUM\_USERNAME

value: "your-documentum-username"

- name: DOCUMENTUM\_PASSWORD

value: "your-documentum-password"

- name: DOCUMENTUM\_REPO

value: "your-documentum-repo"

**service.yaml:**

yaml

Copy code

apiVersion: v1

kind: Service

metadata:

name: s3-documentum-service

spec:

selector:

app: s3-documentum

ports:

- protocol: TCP

port: 80

targetPort: 8080

type: LoadBalancer

**secret.yaml:**

yaml

Copy code

apiVersion: v1

kind: Secret

metadata:

name: aws-credentials

type: Opaque

data:

accesskey: base64-encoded-access-key

secretkey: base64-encoded-secret-key

To create the base64 encoded values for your AWS credentials:

sh

Copy code

echo -n 'your-access-key' | base64

echo -n 'your-secret-key' | base64

**7. pom.xml**

Ensure your pom.xml includes the necessary dependencies:

xml

Copy code

<dependencies>

<!-- Spring Boot Starter Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

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

</dependency>

<!-- AWS SDK for S3 -->

<dependency>

<groupId>com.amazonaws</groupId>

<artifactId>aws-java-sdk-s3</artifactId>

</dependency>

<!-- Documentum SDK (replace with actual groupId and artifactId) -->

<dependency>

<groupId>com.documentum</groupId>

<artifactId>documentum-sdk</artifactId>

<version>your-version</version>

</dependency>

<!-- Other necessary dependencies -->

<dependency>

<groupId>org.springframework.boot</groupId>

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

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

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

<scope>test</scope>

</dependency>

</dependencies>

**8. Build and Deploy**

Follow the steps to build your application, create a Docker image, push it to a repository, and deploy it to AWS EKS as described previously.

This structure and the associated files will help you transition your application from AWS Lambda to a Spring Boot application running on AWS EKS.