Crawling and text categorizing system project.

The System consists of the main services: Text Crawling and Categories.

The UrlTextCrawling service

performs asynchronous a Web URL page and separation of text of page, links, paragraphs.

The downloaded text is mapped to url while being output to a communication channel (in this case- file System), which can be listened by another services.

In this case Camel FS is used with observed directory /vol_interract/text_mail

This services exposed the following test API with common server name:

/crawling

Get Single text
GET crawling/url query parameter “ur”=[your URL]”

Processing batch of urls

POST /crawling/batch HEADER Content-Type=application/json”, Request body: [“url1”,”url2”]
Returns; application/json

Calling any of the rest api methods leads to computation text for URLS and placement of the url to text mapping in file within directory, way of Camel's producer
Categorization service will react on this changes and recompute categories.

Internally, the services works fully asynchronously, having limited number of processing threads and executors.

On the bottom level the services uses the JSOUP library for url text extraction

The CrawlingControler controller class, which handles the described bellow methods, returns Mono(Reactive Java)

Categorization service (CategorizationService).

The service relates a text of an URL to predefined categories, where a category defined by name and array of keywords.

The services reads directories definition from /vol_interact/cathegories directory,

or, from internal resources, when absent.

The services listens camel entry point, which, in this case observes the vol_interact/text_mail directory and reacts on file records changes continuously by updating url text by category mapping.

The CategorizationServer exposed REST SPI, which is mapped by Spring mechanism by CategoryController controller.

It exposes the REST API:

GET /cathegories/entries/{category}" which returns json body, which is map of urls, which belong to this category with matching power.

Services common architecture.

The services collaboration is Event Driven, uses the Event Source pattern and is built on choreography principles.
When the CategorizationService is called by API and extracts an URL text, Camel producer of the service sources an event (way of Camel), which is handled by the CathegorizationService's Camel observer, The CathegorizationService reacts by processing text of URL.

The Services implemented as Spring Boot Rest Application with dynamic configuration, perticularly, of Camel URL forming, which can be adapted to any Camel supported messaging.

Modular structure.

The project has modular structure of the three modules:

1. commons.infra module- common infrastructure of Services, which includes main Spring configurations, as Camel utilities, data marshaling, asynchronous operations, properties

load

2. UrlTextCrawler – uses the common.infra module, implements asynchronous text download from URL, sources Camel FS events
3. cathegorization.cathegorized – uses the common.infra module, implements text categorization by predefined categories, upon reacting on new text and URI, which are produced by Camel

Containerization.

The TextCrawlerSDervice and CathegorizationService should be packed into docker images.
The starting docker line should contain -v /interact_volume:/interact_volume.

Kubernetes.

If Kubernetes used, then there should be done the following things.

1. PVC must be defined at list for number of Gigas (as) we use them for Camel communication.If PV are not definded by administration, then they are to be defined with related selectors (see the volume.yaml as template)
2. Delivery yamls must be defined WITH PVM references from the 1. name of volume must be /vol_interact (under the containers definition in template. This must be done for both of service with the same volumes (that is critical).

Discussion.

Performance issues.

1. Categorization requires application of special algorithms and caching. The used solution is primitive and simply searches for keywords in text. This way, we have Cat_number* Word_per_cat* max word len order of complexity.
2. The categorization algorithm can be improved by the following way at least:
   2.1. Replace text's not word characters by " " as text=text.replaceAll("[^a-zA-Z0-9]", "") ; and then split text text=text.split("\\ "), replace text as Map<String,List<Integer>> where keys are words, and values- indexes of their occurrences. Identification of categories by keywords now is simply to find a keyword word in the map, and, if the word is composite, then see are the indexes of their occurrences sequential.
   2.2 But this way requires additional memory, and, besides, it will spend time for text conversion 2 map (will be dependent on number of words in the text). Besides the procedure will be repeated when reacting on new URL and text. Alternative way- obligate crawling services to produce a conversion of text to map(or set), but this would create a coupling between services

Service interaction.

1. Now Camel FS interaction is used as a simplest solution way with persistence, which can be realized without addition frameworks, but inside Docker, Swarm, Kubernetes concepts
2. An Alternative way is to use a messaging framework, but NOT Kafka (huge message size for it), No Rabbit (same cause). However I would see AWS Kinezis as a good fit for this goal.
   The existing mechanism of Camel's messaging integration is quite suitable, when com.txt_mining.common.config.camel.impl.fs.FsCAmelOpeartions should be replaced by relate Kinezis oriented implementation of IcamelUriInherent interface (camel.uri.builder property).

3.