Institut für Informationssysteme
Distributed Systems Group
VU Advanced Internet Computing (184.269)

Project Topic 4 – Docker-based Service Composition

Introduction

Your task in this project is to develop a simple Docker-based service composition. The composition performs a sentiment analysis for tweets. Sentiment analysis is, broadly, the process of finding out (typically automatically) what the general feeling ("sentiment") of one or more Web communities (for instance, the blogosphere or the Twitter community) about a company or product is [2]. This sort of analysis has become an increasingly relevant marketing tool in recent years. In short, a service composition is a sequence (or more complex pattern [3]) of services, together forming a complex application [1].

The composition needs to be modeled as a BPMN process[1], consisting out of three steps, as sketched in Figure 1:

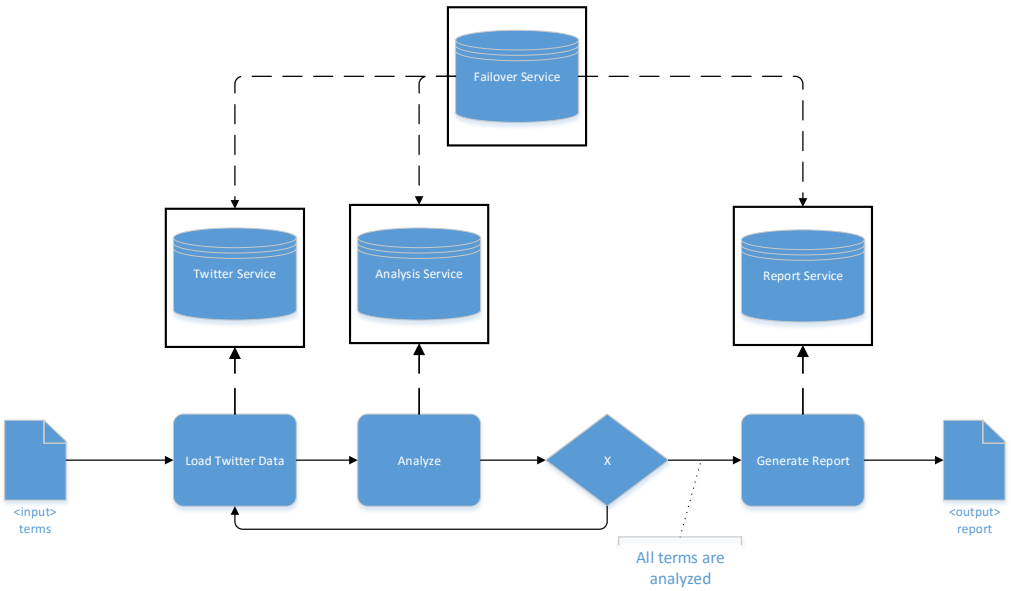

Figure 1: Sentiment Analysis Process Sketch

Essentially, the service composition should expect a list of terms as input, e.g. <Microsoft, Apple, Audi>. Then, for each of those terms, a set of tweets should be loaded from the *Twitter Service*. These tweets are forwarded to an *Analysis Service* for sentiment processing. Once the individual values for each term are available, those terms and sentiment values are passed to the *Report Service*, which generates a PDF report. Each of these services needs to be deployed in a Docker Container. The modelling and execution/enactment of the process should be done using a BPMN 2.0 engine like Camunda[2] or jBPM[3].

[1] `http://www.bpmn.org/`
[2] `http://www.camunda.com/`
[3] `http://www.jbpm.org/`

Outcomes

The expected outcomes of this project are two-fold: (1) the actual project solution, (2) two presentations of the results.

Project Solution

The project has to be hosted on a Git repository provided by the Distributed Systems Group – you will get instructions how to apply for such a repository at the lab kickoff meeting. Every member of your team is assigned a separate Git account. *We will check who has contributed to the source code*, so please make sure you use your own account when submitting code to the repository. Furthermore, it is required to provide an easy-to-follow README that details how to deploy, start and test the solution.

For the submission (see below), you also have to create a virtual machine (Linux or Windows as a vmdk file), which hosts your implemented solution. You can find detailed instructions on how to configure this virtual machine on TUWEL. Alternatively you are also allowed to submit your solution as Docker Containers stored on Docker Hub [4]. In case you submit your solution as Docker Containers please also provide your Dockerfiles in your repository.

Presentations

There are two presentations. The first presentation is during the mid-term meetings, and covers at least the tasks of Stage 1 (see below), the second presentation is during the final meetings and contains all your results. The actual dates are announced on TISS.

Every member of your team is required to present in either the first *or* the second presentation. Each presentation needs to consist of a slides part and a demo of your implementation. Think carefully about how you are going to demonstrate your implementation, as this will be part of the grading. You have 20 minutes per presentation (strict). We recommend to use only a small part of the presentation for the slides part (e.g., 5 minutes) and to clearly focus on the presentation of your results.

Grading

A maximum of 60 points are awarded in total for the project. Of this, up to 35 points are awarded for the implementation (taking into account both quality and creativity of the solution as well as code quality), 20 points are awarded for the presentations (taking into account content, quality of slides, presentation skills, and discussion), and 5 individual points for questions during the presentations.

A strict policy is applied regarding plagiarism. Plagiarism in the source code will lead to 0 points for the particular student who has implemented this part of the code. If more than one group member plagiarizes, this may lead to further penalties, i.e., 0 points for the implementation.

Deadline

The hard deadline for the project is **January 22rd, 2018**. Please submit the presentations as a single ZIP file via TUWEL. The submission system will close at 12:00 sharp. Late submissions will not be accepted. The only exception is the virtual machine, which needs to be uploaded to a public file hosting service or can be submitted on-site on **January 22rd, 2018, 09:00-11:00**. In case you upload the virtual machine to a public file hosting service, please send a mail to `aic@infosys.tuwien.ac.at`, also until 12:00 on January 22rd.

[4] `https://hub.docker.com/`

Test Cloud Infrastructure

Although the topic implementation can be conducted locally without any cloud resources, we recommend every group to also run their implementations in a cloud environment. Unfortunately it has become harder to provide you with cloud resources for the implementations.

However, both Amazon Web Services[5] and the Google Cloud Engine[6] offer free computational resources for testing purposes. From our experience in the last years, these free resources are by far sufficient for the AIC lab. You need to apply for these resources on your own, which requires a credit card. If you do not have access to any credit card, please contact us at `aic@infosys.tuwien.ac.at` so that we can assign some resources from our teaching grant which only covers a limited subset of all features provided by the Google Cloud Engine.

Stage 1

In the first stage, you are required to implement the three software services, which are together forming the service composition. Each service has to provide a REST interface to be accessible by the BPMN engine. You may use suitable libraries to create these REST-based services, e.g., Spring Boot[7].

The **Twitter Service** queries Twitter to obtain tweets for each of the provided search terms. You may use a library, e.g., twitter4j[8].

The **Analysis Service** derives the sentiment for a given tweet. You should implement a wrapper for an online sentiment analysis service, e.g., indico[9].

The **Report Service** generates a PDF report, based on the sentiment analysis. You may use a library, e.g. PDFBox[10].

To realise the actual composition, you can use any *BPMN engine* of your choice, e.g., Camunda[11] or jBPM[12]. It is required to select an actual BPMN engine, i.e., generic programming or scripting languages, like Java or Ruby, are not sufficient.

Tasks:

1. Implement the required software services.
2. Wrap each software service within a Docker Container. The services need to expose a REST interface to the BPMN engine, whereas each service should listen on a different port.
3. Deploy the Docker Container locally or on a cloud environment.
4. Build your composition with the BPMN engine of your choice and integrate the previously built services. It is advised to work in an iterative manner (one service after the other, and test each service extensively).

[5] `https://aws.amazon.com/grants/`
[6] `https://cloud.google.com/free/`
[7] `http://projects.spring.io/spring-boot/`
[8] `http://twitter4j.org`
[9] `https://www.indico.io`
[10] `http://pdfbox.apache.org`
[11] `http://www.camunda.com/`
[12] `http://www.jbpm.org/`

Stage 2

In the second stage, failover handling will be included in the service composition. We assume that an arbitrary service fails. To be able to compensate this, you need to implement a background service (Failover Service). This service also runs in a Docker Container and constantly checks (e.g., every 10 seconds) the availability of all services and (re)starts the affected Docker Container if it is not available. You may use a library to handle the interaction with Docker, e.g., Docker Client[13].

Tasks:

1. Implement the *Failover Service*. This service is not only able to check the availability of the other services, but it also represents the fault handler, i.e., it (re)starts the Docker Container of an affected service once it recognises that one of the original services is not responsive within a particular timespan.

2. Extend your process model to consider service failures and perform adequate compensation measures to guarantee the desired result.

3. Provide a simple UI which allows an user to enter the terms and to execute the process to obtain the PDF report.

References

[1] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. *International Journal of Web and Grid Services*, 1(1):1–30, 2005.

[2] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1–2):1–135, 2008.

[3] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[13] `https://github.com/spotify/docker-client`