Semantic Web Analysis with Flavor of Micro-Services

Farshad Bakhshandegan Moghaddam[1], Carsten Felix Draschner[1] Jens Lehmann[1], and Hajira Jabeen[2]

[1] University of Bonn, Bonn, Germany
{firstname.lastname}@uni-bonn.de,
[2] University of Cologne, Cologne, Germany
hajira.jabeen@uni-koeln.de

Abstract. The last decades witnessed a significant evolution in terms of data generation, management, and maintenance, especially in the RDF format. Moreover, in the energy domain, semantic data is finding its way and can be used for various data analytics tasks. However, since data set sizes are increasing and can now be enormous, technologies are evolving to scale with the increasing data set sizes. In this regard, tools and frameworks such as SANSA have been emerged to facilitate the analytic over semantic data. SANSA is using big data technologies such as Apache Spark (as an analytics engine for large-scale data processing) and Apache Hadoop (as a distributed file system) in its backbone to be able to perform analytics in a distributed manner over a cluster of nodes. However, to be able to use SANSA, one should set up a cluster of nodes with enabled Spark and Hadoop. This requires extensive knowledge and expertise in computer systems, networking, distributed computing and etc. Moreover, in case of having sufficient technical knowledge, setting up such a cluster consumes huge manpower and is labor-intensive. To tackle the aforementioned issues, in this paper we introduce a micro-service architecture that easily brings the power of SANSA and distributed semantic data analysis in the end-user ecosystem, without having technical knowledge in the mentioned areas. The introduced architecture is based on Docker technologies and can be installed on-premise or in the cloud systems.

Keywords: Micro Services, SANSA, Docker, Distributed Computing, RDF Data, Analytics Pipelines

1 Introduction

With the rapidly growing amount of data available on the Internet, it becomes necessary to have a set of tools to extract meaningful and hidden information from the online data. The Semantic Web is able to form a structural view of the

existing data on the web and provides machine-readable formats [1]. Currently, many companies in the fields of science, engineering, and energy publish their data in the form of RDF[3]. Due to the complex graph nature of RDF data, applying standard machine learning algorithms to this data is cumbersome. Moreover, the challenges in the current big data era (limited computational resources) cause analytical approaches to mostly fail to operate on large-scale data. To tackle the mentioned issues, Scalable Semantic Analytics Stack (SANSA) [6] has been emerged. SANSA addresses the need of having a scalable and distributed computational engine to work with semantic data. It benefits from an in-memory analytics framework, Apache Spark[4] and provides fault-tolerant, highly available, and scalable approaches to efficiently process RDF data with the support of semantic technology standards. SANSA provides various layers of functionalities for semantic data representation, querying, inference, and analytics[5].

To be able to use SANSA effectively, a cluster of Spark nodes with an HDFS file system is required. Even by having such a cluster, one can only interact with SANSA via a terminal. Establishing a cluster of different nodes containing Spark and Hadoop[6] and configuring them is by nature a cumbersome task and needs a lot of knowledge and experiences. Moreover, in the case of having technical knowledge, establishing such a cluster is a time-consuming task. Moreover, end-users prefer to have a user friendlier way to interact with SANSA without having any programming and scripting knowledge. Therefore, in this paper, we introduce a micro-service structure of a SANSA-enabled Spark and Hadoop cluster with 2 user-friendly interactive communication mechanisms aka. REST API and Zeppelin Notebook[7]. Our introduced architecture is based on Docker technologies[8]. Moreover, our sample explanatory tutorial enables non-technical users to easily use SANSA without having any specific knowledge and skills.

1.1 Contributions:

- Introducing a micro-service architecture of Big Data tools such as Apache Spark, Apache Hadoop, Apache Zeppelin, HDFS File Browser, Apache Livy
- Introducing for the first time REST APIs for the SANSA stack
- Introducing an interactive Notebook (i.e. Apache Zeppelin) for interacting with SANSA
- Making the code and the framework open-source and publicly available on Github[9]

[3] https://www.w3.org/RDF/
[4] http://spark.apache.org/
[5] http://sansa-stack.net/
[6] https://hadoop.apache.org/
[7] https://zeppelin.apache.org/
[8] https://www.docker.com/
[9] https://github.com/SANSA-Stack/SANSA-Stack

2 Related Work

In recent years, it has been recognized that creating complex technical environments for big data is a major challenge. Setting up a lot of nodes and connecting them together in a way that all together perform a specific task not only is costly but requires extensive knowledge in many computer science areas such as networking, cluster computing, and big data technologies. Although nowadays, other technologies such as virtualization (e.g. VMbox3, Parallels, VMware) and its successor containerization (e.g. Docker, Swarm, Kubernetes) keep the costs lower but add up other skills to be able to set up a cluster properly.

Besides this, there are numerous centralized machine learning frameworks and algorithms for RDF data. For example, TensorLog [3] and ProPPR [9] are recent frameworks for efficient probabilistic inference in first-order logic. AMIE [5] and AMIE+ [10] learn association rules from RDF data. DL Learner [2] is a framework for inductive learning for the Semantic Web. [7] provides a review of statistical relational learning techniques for knowledge graphs. However, SANSA leveraged Apache Spark and Apache Jena[10] to provide an open-source, distributed, scalable, and end-to-end framework for data analytic pipelines for large-scale RDF Knowledge Graphs[4, 11, 13].

In order to use SANSA effectively, having a distributed environment is inevitable. Although, some enterprise companies such as Databricks[11] [12] develops a web-based platform for working with Spark, that provides automated cluster management and IPython-style notebooks, however, they are costly as the user needs to provide AWS[12], Microsoft Azure[13], or Google Cloud[14] account. Moreover, its free community edition provides limited functionality and services. However, our approach differs from the mentioned methods as it provides a versatile, flexible, and free-to-use framework via Docker technologies which can be set up on a single machine (e.g. a laptop) or a cluster of machines in a cloud environment.

3 Architecture

In this section, we present the micro-service system architecture for using SANSA. Worth mentioning that the framework is open-source and hosted on GitHub[15]. The main goal of the framework is to bring a simple and effortless approach to set up a Spark cluster with all the requirements without having extensive computer science knowledge.

[10] https://jena.apache.org
[11] https://databricks.com/
[12] https://aws.amazon.com/
[13] https://azure.microsoft.com
[14] https://cloud.google.com/
[15] https://github.com/SANSA-Stack/SANSA-Stack

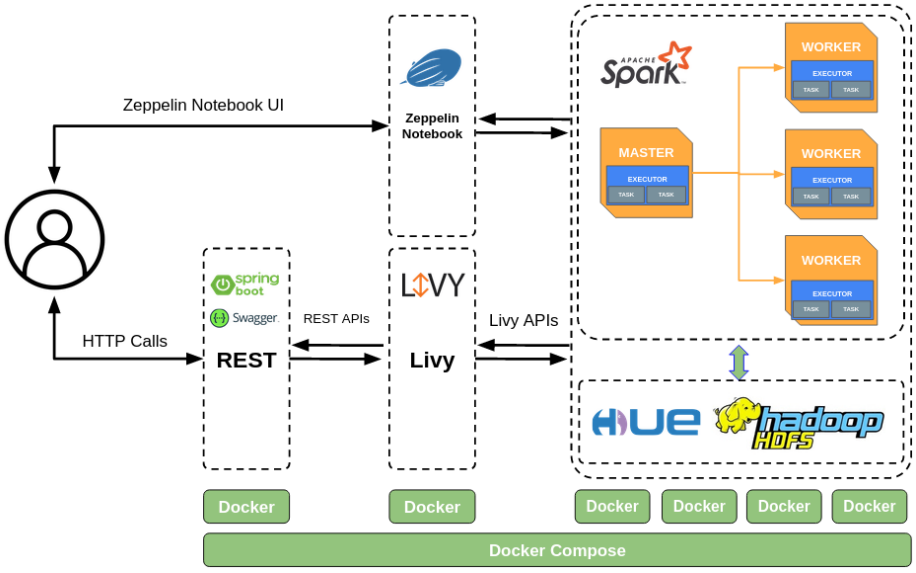

Fig. 1. High-level system overview

3.1 Components

We provided two interaction mechanisms for the end-users. *a)* Zeppelin Notebooks *b)* REST APIs. Depends on the scenario users may select one of the mentioned mechanisms. These mechanisms cover the full spectrum from simplicity to flexibility. By using the REST APIs, users will be able to call predefined functionalities from SANSA without any effort. However, in case a user is interested in the new functionalities, they can write code and stack their code via Zeppelin Notebook and submit their task to the Spark cluster. Figure 1 depicts the high-level system overview. The architecture contains 4 main components i.e. *a)* Java-based REST APIs *b)* Apache Zeppelin Notebook *c)* Apache Livy *d)* Spark-Hadoop Cluster, which are all utilized via Docker-Compose.

Java-based REST API This layer provides functionality for the end-user to interact with SANSA via REST APIs. It is Java-based and is powered by Spring Boot[16] technology and contains a Swagger[17] UI which enables users to easily call any provided functions via a browser. Figure 2 shows the SwaggerUI and provided APIs. A sample scenario of how to use these APIs is provided in Section 4.

Apache Livy REST APIs As the Spark tasks may be long-running (up to a few days) and also there is a chance that the node which is running the task

[16] https://spring.io/projects/spring-boot
[17] https://swagger.io/

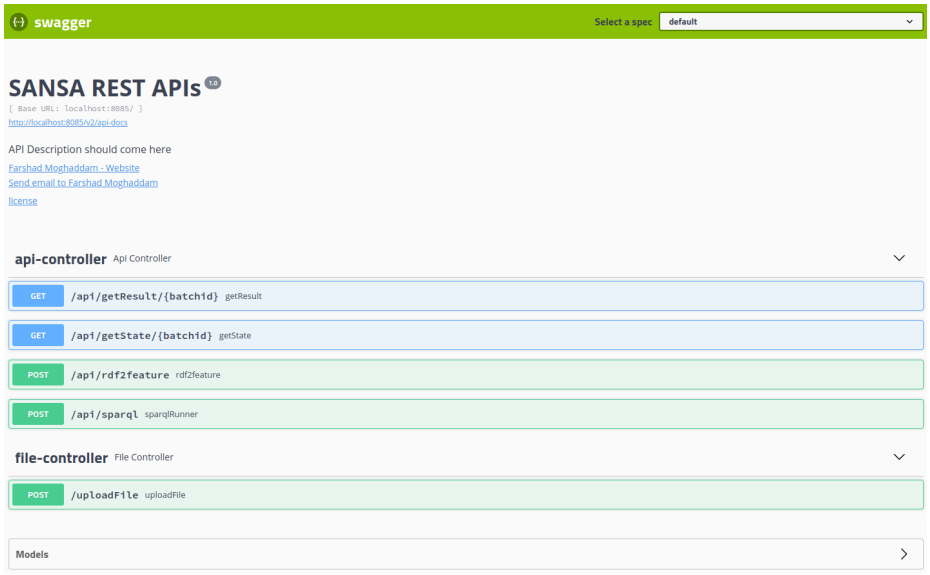

Fig. 2. REST Swagger UI

crashes and loses the calculations, therefore, directly connecting the REST APIs to the Spark cluster is not feasible due to asynchronous nature of such computations. To tackle this, another layer has been added by using Apache Livy[18], which is able to keep tracks of Spark sessions and calculation states. Livy enables programmatic, fault-tolerant, multi-tenant submission of Spark jobs from web/-mobile apps. So, multiple users can interact with the Spark cluster concurrently and reliably. Although the Livy interface is available for the user, this layer works as a background process and the user does not need to interact with it directly, because all the functionalities will be handled by the REST API layer.

Spark-Hadoop Cluster To be able to run a SANSA functionality, having a Spark cluster with a Hadoop file system is inevitable. To do so we containerized Spark, Hadoop Namenode, Hadoop Datanode, and Hue HDFS file browser[19], in docker images which are publicly available. Moreover, we configured the containers to interact with each other seamlessly via Docker-Compose. Of course, all the other layers have been containerized and exposed in the same docker-compose file.

[18] https://livy.apache.org/
[19] https://gethue.com/

4 Usage

To be able to run the cluster, the user needs to clone the SANSA Stack from Github and navigate to *sansa-rest* sub-folder. The following codes in the terminal will bring up the cluster.

```
$ git clone https://github.com/SANSA-Stack/SANSA-Stack.git
$ cd SANSA-Stack/sansa-rest
$ make
$ make up
```

To stop the cluster, the user only needs to run the following command.

```
$ make down
```

Table 1 lists all the available endpoints and their functionalities.

Table 1. Available endpoints and their functionalities

endpoint	**functionality**
http://localhost	Zeppelin Notebook
http://localhost:8085	REST Swagger UI
http://localhost:8998	Livy UI
http://localhost:8080	Spark Master UI
http://localhost:8088/home	Hue file browser UI

As already mentioned, users will have two interaction mechanisms to connect to SANSA, either using Zeppelin Notebook or REST APIs. Using Zeppelin Notebook is easy and straightforward same as all the other notebook technologies such as Jupyter Notebook[20]. Therefore, without losing the generality and due to the space issue, we ignore the explanation in this paper. However, in the following, we explain a sample scenario which shows how to use the REST APIs. Besides its many functionalities, SANSA provides a distributed SPARQL engine (i.e. Sparklify [8]) which is able to execute a SPARQL query in a distributed fashion. As a scenario, imagine the user has an RDF file (any format) and would like to run a SPARQL query over it. To be able to use a REST API for this purpose, the user first needs to upload her file into the HDFS because SANSA needs to access the file in a distributed manner. For this reason, we have provided an API in Swagger which enables the user to upload files to the HDFS. The result of the API call will be the address which the file will be stored in the HDFS. The user will need to provide this address for any subsequent API call. To call the SPARQL engine API (i.e. `/api/sparql`), the user simply needs to provide the SPARQL query and the address of the file which he retrieved from the file upload API. Keep in your mind that the result of any API call will be a *livy batch id*. This *id* is a unique number which identifies a Livy session which

[20] https://jupyter.org/

is responsible for the task computation. We provided two APIs which receive this *id* and provide more information about the execution process and about the result of the executions. The result of the `/api/getState` API will be either `running`, `success`, or `dead`. Only in case of `success`, the user can use `/api/getResult` API to see the result of the call. The other two states either show the process is ongoing, or the process is unexpectedly stopped.

5 Conclusion

In this paper, we introduced a framework which is able to set up a Big Data-enabled cluster with all its requirement from Spark and Hadoop to Zeppelin Notebook to be able to use SANSA framework without any effort. The proposed approach is based on Docker Compose technology and can be installed on-premise or on any cloud environment. Moreover, we implemented a set of REST APIs via Swagger which enables all the non-technical users to interact with SANSA in a very straightforward manner.

Acknowledgement

This work was partly supported by the EU Horizon 2020 project PLATOON (Grant agreement ID: 872592). We would also like to thank the SANSA development team for their helpful support.

References

1. Berners-Lee, T. A roadmap to the Semantic Web. (http://www.w3.org/DesignIssues/Semantic.html,1998)
2. Bühmann, L., Lehmann, J. & Westphal, P. DL-Learner - A framework for inductive learning on the Semantic Web.. *J. Web Semant.*. 39 pp. 15-24 (2016)
3. Cohen, W. TensorLog: A Differentiable Deductive Database. *CoRR*. abs/1605.06523 (2016)
4. Draschner, C., Lehmann, J. & Jabeen, H. DistSim-Scalable Distributed in-Memory Semantic Similarity Estimation for RDF Knowledge Graphs. *2021 IEEE 15th International Conference On Semantic Computing (ICSC)*. pp. 333-336 (2021)
5. Galárraga, L., Teflioudi, C., Hose, K. & Suchanek, F. Fast rule mining in ontological knowledge bases with AMIE+.. *VLDB J.*. 24, 707-730 (2015)
6. Lehmann, J., Sejdiu, G., Bühmann, L., Westphal, P., Stadler, C., Ermilov, I., Bin, S., Chakraborty, N., Saleem, M., Ngonga, A. & Jabeen, H. Distributed Semantic Analytics using the SANSA Stack. *Proceedings Of 16th International Semantic Web Conference - Resources Track (ISWC'2017)*. pp. 147-155 (2017)
7. Nickel, M., Murphy, K., Tresp, V. & Gabrilovich, E. A Review of Relational Machine Learning for Knowledge Graphs. *Proceedings Of The IEEE*. 104, 11-33 (2016), https://doi.org/10.1109/JPROC.2015.2483592
8. Stadler, C., Sejdiu, G., Graux, D. & Lehmann, J. Sparklify: A Scalable Software Component for Efficient Evaluation of SPARQL Queries over Distributed RDF Datasets. *The Semantic Web - ISWC 2019 - 18th International Semantic Web*

Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II. 11779 pp. 293-308 (2019), https://doi.org/10.1007/978-3-030-30796-7

9. Wang, W., Mazaitis, K. & Cohen, W. Structure Learning via Parameter Learning.. *CIKM*. pp. 1199-1208 (2014)
10. Galárraga, L., Teflioudi, C., Hose, K. & Suchanek, F. Fast rule mining in ontological knowledge bases with AMIE+. *VLDB J.*. 24, 707-730 (2015)
11. C. F. Draschner, C. Stadler, F. B. Moghaddam, J. Lehmann, and H. Jabeen, "DistRDF2ML-Scalable distributed in-memory machine learning pipelines for rdf knowledge graphs" in *2021 ACM International Conference on Information and Knowledge Management (CIKM)*.
12. C. F. Draschner, F. B. Moghaddam, J. Lehmann, and H. Jabeen, "Semantic Analytics in the Palm of Your Browser" in *Big Data Analytics 3rd Summer School* (2021).
13. F. B. Moghaddam, C. F. Draschner, J. Lehmann, and H. Jabeen, "Literal2Feature: an automatic scalable rdf graph feature extractor" in *Proceedings of the 17th International Conference on Semantic Systems, SEMANTICS 2021, Amsterdam, The Netherlands, September 6-9, 2021*. SEMANTICS (2021)