Using DevOps Principles to Continuously Monitor RDF Data Quality

Roy Meissner
Institute for Applied Informatics
Hainstrasse 11
Leipzig, Germany
meissner@informatik.uni-leipzig.de

Kurt Junghanns
Institute for Applied Informatics
Hainstrasse 11
Leipzig, Germany
kjunghanns@informatik.uni-leipzig.de

ABSTRACT

One approach to continuously achieve a certain data quality level is to use an integration pipeline that continuously checks and monitors the quality of a data set according to defined metrics. This approach is inspired by Continuous Integration pipelines, that have been introduced in the area of software development and DevOps to perform continuous source code checks. By investigating in possible tools to use and discussing the specific requirements for RDF data sets, an integration pipeline is derived that joins current approaches of the areas of software-development and semantic-web as well as reuses existing tools. As these tools have not been built explicitly for CI usage, we evaluate their usability and propose possible workarounds and improvements. Furthermore, a real-world usage scenario is discussed, outlining the benefit of the usage of such a pipeline.

CCS Concepts

•**Information systems** → **Information integration; Resource Description Framework (RDF);** *Data cleaning;* •**Software and its engineering** → *Software maintenance tools;*

Keywords

DevOps; Continuous Integration; RDF; Data Quality; Quality Monitoring; Data Integration; Instant Feedback;

1. INTRODUCTION

Recent research results of Arndt et al. [1] uncovered an approach towards storing an RDF data stores data inside a Git repository. A natural continuation of this idea will lead to uploaded data stores on repository hosting services (RHS), like GitHub, in order to backup, share and publish the data. Besides this, some ontologies and data sets are already developed and published on RHSs, like MODS RDF[1], Public NPM Domain Ontology[2] and many more.

One of the enhancements of the DevOps movement, that gained traction within the last years as of the area of software development, is the development workflow of Continuous Integration (CI), that has been implemented by several providers for popular RHSs. CI refers to a development workflow where a development team frequently commits their work. Each commit is tested and integrated by an automated build tool detecting errors [6]. Current CI approaches focus on software development (source code repositories), rather than data centered repositories.

As a CI pipeline helps to maintain an integrable quality of the content of an RHSs hosted repository [5], we define in this paper a CI pipeline for repositories containing RDF data. Our focus is to reuse existing software solutions to check and analyse the repositories content and, in the end, to indicate the data quality and integrability of the RDF. We have implemented the presented pipeline exemplarily on the RHS provider GitHub[3] and the CI provider Travis-CI[4] to showcase its functionality and validate the usability of the available RDF quality assessment software for the CI use case.

2. RELATED WORK

Sandro Cirulli presents a CI pipeline that integrates XML data and converts them to RDF [3]. He describes that his department uses the pipeline for quality assessment, quality improvement and to execute integration tasks. As of the quality assessment part, the used software solutions (RDFUnit for RDF based data) provide JUnit result sets that integrate quite well with their remaining technology stack (Jira). Considering the (for this paper interesting) part of the CI usage for quality assessment, he just states that RDFUnit is used. In addition, he outlines benefits of the usage of the CI pipeline, that are only repetitions of general benefits of the usage of a CI pipeline (like in [5, 6]) and thus not use case specific.

VoCol, described by Peterson et al. [11], is a tool which aims to improve the whole process of vocabulary creation and maintenance. Their motivation is that common vocab-

SEMANTiCS 2016 September 12-15, 2016, Leipzig, Germany

ACM ISBN 978-1-4503-4752-5/16/09.

DOI: http://dx.doi.org/10.1145/2993318.2993351

[1]MODS RDF GitHub repository: https://github.com/blunalucero/MODS-RDF

[2]Public NPM Domain Ontology GitHub repository: https://github.com/nature/public-npg-domain-ontology

[3]GitHub (RHS) Homepage: http://github.com

[4]Travis-CI (continuous integration tool) Homepage: https://travis-ci.org

ularies are mostly simple because the creation process is too difficult. In order to improve the situation they discussed necessary steps and possible solutions of the process, which resulted in the implementation of an collaborative environment for ontology engineering. VoCol covers quality assessment topics like validation in terms of syntax, semantic error detection and reporting in order to make users aware of possible errors. Therefore it uses some of the tools that are described as of section 3, as users work at vocabularies.

3. OVERVIEW OF RDF QUALITY ASSESSMENT TOOLS

The Raptor Utils[5] are a set of command-line tools to parse and convert several RDF serialization formats and check parts of their syntax and semantics. A similar attempt is trailed by Apache Jena RIOT[6], that is part of the Apache Jena project.

The Ontology Pitfall Scanner! (OOPS!) is an RDF authoring tool for detecting common problems when creating ontologies. It is described by M. Poveda-Villalón et al. [12], where they explicitly state that they have implemented a web-service that checks an issued ontology against a pattern library for the most common pitfalls. They have developed their tool for ontology validation only. Nevertheless, some of the described pitfalls in [12] can be treated as general pitfalls and will also apply for plain RDF data sets.

Beek et al. introduce the Linked Open data (LOD) Washing Machine [2], that is a part of the LOD Laundromat. The LOD Washing Machine is a software that scans common RDF document serializations for patterned pitfalls that are explicitly LOD focused. These pitfalls will be automatically fixed or deleted. Thus the final data set is pitfall free regarding their defined pitfall library. In addition, they record occurring pitfalls for further usage scenarios, like surveys.

Kontokostas et. al describe the tool Databugger [8] (that has been renamed to RDFUnit) that focuses on LOD data quality assessment. It is a tool that automatically computes test cases based on RDF data sets or ontologies and can also use manually created ones. These test cases are subsequently executed as SPARQL queries on top of an issued RDF data set to discover and assess data quality problems. The result of a test execution is reported in a human readable format and the RDF data is left unaltered, thus it has to be revised manually in order to fix issues.

Debattista et al. present the Luzzu Assessment Quality Framework [4], which is a web-service that executes metrics on linked data and stores a history of executions with results in order to keep or increase their quality. The tool can be extended with custom metrics and focuses on scalability to improve the assessment performance. Metrics have to use Luzzu specific vocabularies. In order to improve the metric creation process, they invented the Luzzu quality metric language. Each execution of Luzzu performs five steps which are defined as the data quality lifecycle. The result set of an execution is a queryable quality report. They state that Luzzu is scalable and can be used for any kind of RDF data.

Zaveri et al. present among other topics several tools for RDF quality assessment [13]. According to our investigations, all quality assessment related tools, that have not been mentioned until now, are not available or executable any more. Therefore we do not include them.

[5]Raptor RDF Syntax Library (Homepage): http://librdf.org/raptor

[6]Apache Jena RIOT Documentation: https://jena.apache.org/documentation/io/

4. INTEGRATION PIPELINE

The following sections outline the approach and state of the art implementation of CI at CI providers, preliminaries to the proposed integration pipeline, the integration pipeline itself and a usage scenario.

4.1 Continuous Integration

According to [5] (that cites [6] the overall goal of CI is to continuously check a repository for possible errors, outline warnings and to keep or improve the integrability of the repositories content. Integrability is therefore used as of two different aspects: to check the quality, as well as the integrability in terms of integrability with other data/software of the repositories content. As the last aspect heavily depends on the use case of the content, we will mainly focus at the first aspect as of the following sections.

In [10], Vladimir Pecanac presents a comparison of the most discussed CI providers (according to his Google Trends research). By digging through their corresponding documentation, the following state of the art CI provider approach has been revealed:

Current CI software providers offer (virtual) linux machines, which will execute predefined scripts (the pipeline) with every push to a corresponding repository that is hosted at some RHS. Or to be more precise: the CI software registers an after-push hook that triggers the corresponding CI pipeline. Thus, the used CI software realizes the continuous execution of the pipeline.

CI providers allow two exit codes that indicate the healthiness of an execution: succeeded and failed. The healthiness of a whole pipeline execution (in contrast to an execution of a particular command) is visualized by a badge, that is, for open-source software, often included into the RHS repository itself (like into the file README.md at a GitHub repository). A CI user is able to view each pipeline run that has been executed and inspect the concrete outputs of each command in order to fix issues or improve the repositories content.

4.2 Preliminaries

With the explanations from section 4.1 in mind, tools that check the data contained in a repository have to be executable as command-line tools or requestable as web-services. Command-line tools are preferred because of their a priori better integration into the CI environment, that is command-line based. As of the range of applicable tools, a CI pipeline will have to use tools that support failed executions rather than just print warnings but succeed the execution. Otherwise the CI provider is not able to indicate to the user if a pipeline run failed.

As of the fact that many tools exist for special aspects of RDF data quality on the WWW, like we have outlined in sections 2 and 3 as well as the listed tools in [13], an RDF integration pipeline (IP) does not have to be reinvented. Instead it can reuse existing tools. These tools will have to check the syntax of a data set, as well as for common pitfalls, like outlined in [7, 13] and the publications of the mentioned tools in section 3.

4.3 Integration Pipeline

According to the available tools from sections 2 and 3, we classified them according to their main focus. Therefore we defined three classes: syntax and basic semantic accuracy, vocabulary/ontology and plain data. We classified the tools as follows:

- Syntax and basic semantic accuracy
  - Raptor Utils
  - Apache Jena RIOT
- Vocabulary/Ontology
  - OOPS!
  - VoCol
  - Luzzu
  - RDFUnit
  - LOD Washing Machine
- Plain data
  - RDFUnit
  - Luzzu
  - LOD Washing Machine

This classification in mind, we use the Raptor Utils in each and every case as a precondition to check for syntax errors and a basic level of semantic accuracy. We chose Raptor Utils because of their availability in most modern Linux distributions software repositories and thus easier setup process at the CI provider.

To check plain RDF data sets, we propose to use the tools RDFUnit and Luzzu. We dockerized RDFUnit to be able to use it regardless of the underlying Linux (virtual) machine of the CI provider. As of the LOD Washing Machine we could not ascertain on how to use it as a standalone tool, or put another way without the remaining tools of the LOD Laundromat. Thus it is currently not usable for CI. The proposed pipeline for plain RDF data sets is depicted in figure 1.

To check vocabularies/ontologies for common pitfalls, we propose to use the tools OOPS!, VoCol, Luzzu and RDFUnit. OOPS! is already hosted as a web-service and thus we had not dockerized it. Luzzu can be used likewise. VoCol, in contrast, isn't usable as of the command-line tool or as a standalone web-service. It seems to be a precondition to use the VoCol editing environment in order to use VoCols functionality. Therefore, the proposed pipeline for RDF vocabularies/ontologies is basically the same like in figure 1. The only difference is that a new block with the tool OOPS! is added.

4.4 Usage Scenario

We like to outline as a concrete usage example a continuation of the ideas introduced by Arndt et al. [1]. They describe a triple store that is built upon the source code versioning tool git. Whenever a store shall be publicly available or used in a collaborative manner, it can be uploaded to some RHS, like GitHub. As soon as the stores data gets modified, the changes will be pushed to the corresponding RHS and subsequent, the CI pipeline gets triggered by the RHS. The CI pipeline executes several tests that check the contained datasets on the one side for data quality aspects and on the other side for integrability aspects (that have to be implemented for the concrete use case). In the end, the corresponding commit at the RHS will be highlighted according to the data climate and the owner of the repository will be, if necessary, notified in order to improve the dataset.

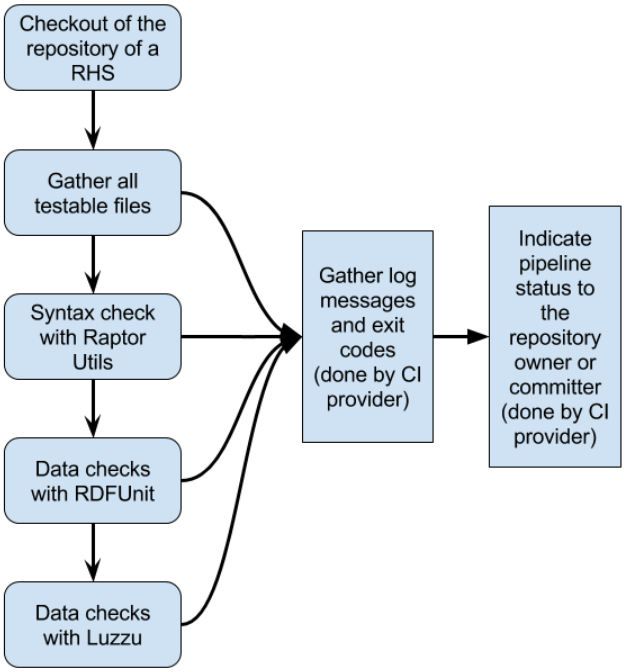

Figure 1: CI pipeline for plain RDF data sets

5. REALIZATION AND VALIDATION

To showcase the functionality of the proposed CI pipeline, we have implemented it on top of a Git repository that is hosted at GitHub. The pipeline itself is executed at the CI provider Travis-CI and is triggered by a push event. The used data set and Travis-CI configuration is accessible at the mentioned Git repository at GitHub at https://github.com/AKSW/amsl-on-ci. We use a fork of the AMSL vocabulary, that contains 1711 triples and has been introduced in [9], to showcase a pipeline for vocabularies. The data set has been edited manually to introduce custom errors in order to show the pipelines functionality across different commits.

We observed that the mentioned tool OOPS! and Luzzu of section 3 are only usable as web-services. As CI providers deal with exit codes, the response of a web request has to be parsed and analysed in order to produce proper exit codes. Thus, extensions or wrappers are needed in order to properly use these tools on CI providers. In contrast, RDFUnit and Raptor Utils are usable as command-line tools which produce for tainted data malicious exit codes (different from zero). Apart from that, RDFUnit exits for warnings and complex semantic errors with the status code zero.

We have further noticed that all mentioned tools are using different report formats. E.g. RDFUnit is able to produce a JUnit report, thus they reuse an existing format. Luzzu, in contrast, introduces a rather new format that can be treated as an attempt towards a new standard. As of the current

tool implementations, report converters are needed in order to process the results of a CI pipeline execution either automatically by a program, or manually by an user in order to improve the analysed data. This process can be improved by our suggestions in section 6.

As our proof of concept implementation assumes that an RHS and a CI provider is used, the implementation may emerge as inefficient for large or highly dynamic data sets due to throughput issues. Public RHS providers have implemented repository and file size restrictions (e.g. 1GB repository size limit at Github[7]). Additionally, the mentioned CI providers create a new virtual machine per CI pipeline execution. Thus, the latest repository content has to be cloned onto this virtual machine every time it is created, resulting in network and disk traffic. This may result in delays that depend on hardware capabilities of the used RHS and CI providers. As an expedient, we propose to use the differing fine tuning mechanisms of each provider. A way to further improve the throughput is to set up the provider software at self controlled hardware or to use specialized CI software that reuses created virtual machines.

6. CONCLUSION & PERSPECTIVE

We presented two Continuous Integration pipelines to check RDF data sets as well as vocabularies/ontologies for their data quality and in order to improve their integrability. Therefore we evaluated available RDF quality assessment tools and dockerized some of these in order to make them easily usable for integration pipelines. Some of the presented tools are difficult to use at CI because of their lack of proper command-line interfaces (CLI). The remaining tools need custom extensions in order to improve their CI integrability. The proposed pipelines use CLI tools only and can therefore easily be reproduced and used on other projects.

Our suggestions to improve the area is to settle on a standardized report format, like Luzzu introduces, as well as integrate proper CLI into all tools in order to improve their integrability for the current CI approach. Another improvement, that depends on CLI availability, is to introduce more then two exit codes by classifying the tools report data. Adding these improvements will lead to a much easier to set up and use CI pipeline and so integrate well with a Git triple store, like introduced by Arndt. et al. [1].

7. ACKNOWLEDGEMENT

This work was partly supported by the European Union's Horizon 2020 research and innovation programme for the SlideWiki Project under grant agreement No 688095.

8. REFERENCES

[1] N. Arndt, N. Radtke, and M. Martin. Distributed collaboration on rdf datasets using git: Towards the quit store. In *12th International Conference on Semantic Systems Proceedings*, Sept. 2016.

[2] W. Beek, L. Rietveld, H. Bazoobandi, J. Wielemaker, and S. Schlobach. Lod laundromat: A uniform way of publishing other people's dirty data. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2014.

[3] S. Cirulli. Continuous integration for xml and rdf data. *XML LONDON*, pages 52–60, 2015.

[4] J. Debattista, S. Londono, C. Lange, and S. Auer. Luzzu - a framework for linked data quality assessment. Dec. 2015.

[5] P. M. Duvall, S. Matyas, and A. Glover. *Continuous Integration*. Addison-Wesley, Feb. 2008.

[6] M. Fowler. Continuous integration. http://martinfowler.com/articles/continuousIntegration.html, May 2006. Access date: 2016-06-30.

[7] A. Hogan, A. Harth, A. Passant, S. Decker, and A. Polleres. Weaving the pedantic web. *3rd International Workshop on Linked Data on the Web (LDOW2010) in conjunction with 19th International World Wide Web Conference*, 2010.

[8] D. Kontokostas, P. Westphal, S. Auer, S. Hellmann, J. Lehmann, and R. Cornelissen. Databugger: A test-driven framework for debugging the web of data. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion*, Apr. 2014.

[9] A. Nareike, N. Arndt, N. Radtke, S. Nuck, L. Seige, and T. Riechert. AMSL: Managing electronic resources for libraries based on semantic web. In *Proceedings of the INFORMATIK 2014: Big Data – Komplexität meistern*, volume P-232 of *GI-Edition—Lecture Notes in Informatics*, pages 1017–1026. Gesellschaft für Informatik e.V., Sept. 2014. © 2014 Gesellschaft für Informatik.

[10] V. Pecanac. Top 8 continuous integration tools. http://www.code-maze.com/top-8-continuous-integration-tools/, Feb. 2016. Access date: 2016-07-01.

[11] N. Petersen, L. Halilaj, C. Lange, and S. Auer. Vocol: An agile methodology and environment for collaborative vocabulary development. Feb. 2015.

[12] M. Poveda-Villalón, A. Gómez-Pérez, and M. C. Suárez-Figueroa. Oops!(ontology pitfall scanner!): An on-line tool for ontology evaluation. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 10(2):7–34, 2014.

[13] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, and S. Auer. Quality assessment for linked data:a survey. *Semantic Web Journal*, 7(1):63–93, Mar. 2015.

[7]Size limits at Github: https://help.github.com/articles/what-is-my-disk-quota/