Reproducibility package for JSON Schema Discovery Project

Yash Dalal
University of Passau
Reproducibility Engineering
dalal02@ads.uni-passau.de

ABSTRACT

There is a growing debate about the reproduction of scientific results. Many of the published scientific results cannot be obtained again even by using the same setup. With this project, I would like to reproduce a project on JSON Schema Extraction and verify the results and claims made by the authors. The reproducibility package with all the necessary dependecnies is fully automated using Docker.

KEYWORDS

reproducibility package, JSON schema extraction, docker

ACM Reference Format:
Yash Dalal. 2023. Reproducibility package for JSON Schema Discovery Project. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Currently, researchers are worried about the reproducibility crisis in the scientific community. This occurs when the results of the original work cannot be obtained again using the same experimental setup. This makes the scientific community sceptic about the methods and processes described by the authors to obtain the results.

Through this project I aim to reproduce a JSON Schema Extraction project by Frozza, et. al. [1]. I have used the same experimental setup as used by the authors, hence it is a reproduction package. If the results after building this package successfully match that of the authors then the conclusion of full reproducibility and genuineness can be established. The paper is divided into Literature review 2, Implementation and Building the package 3, Challenges 4, Working of the project 5 and lastly Discussions 6.

2 LITERATURE REVIEW

The authors in the paper [1] have built their own approach of JSON document schema extraction. JSON schema defines the structure of particular JSON document. Their JSON schema extraction process includes 4 different steps.

Firstly, they convert the JSON documents into raw schema. The raw schema is structurally similar to the original JSON document with respect to its attributes, objects and nested arrays. Secondly, they perform aggregation of the raw schemas. This operation outputs a unique collection of JSON objects. Thirdly, this unique collection is then converted into a hierarchical tree based data structure called Raw Schema Unified Structure (RSUS). It has five attributes as - FieldType, PrimitiveType, ExtendedType, ObjectType and ArrayType. Lastly they transform this RSUS tree into final JSON schema as key-value pairs.

2.1 Datasets

The authors have used 5 simple and concise JSON documents with minor differences to check for proper functioning of their algorithm. After evaluation with simple and concise documents, the authors test their algorithm on Foursquare datasets of venues, checkins and tweets.

2.2 Results

The authors found that 99% of the time was consumed obtaining raw schemas from JSON documents. Lastly, they also try to compare their results with results of Wang, et. al. [2] using their datasets on drugs, companies and movies. Their results show that their algorithm performs similar to Wang, et. al's.

2.3 Code Availability

The authors have made their code written in Typescript along with the dependencies available in their Github repository. **MIT license** has been chosen by the authors. Nevertheless, I received a green signal from the authors to reproduce their work after specifying the purpose of the project.

3 IMPLEMENTATION AND BUILDING THE DOCKER RECIPE

The most time consuming and important step of this project is to implement and build the Docker recipe. For this, firstly I went through their paper and repository thoroughly. Luckily, the authors have provided a *package.json* file containing the dependencies and their exact version. The *README* file also contains the commands to start and build their project on the command line. The following contents will throw a light on the various aspects of implementation.

3.1 Dockerfile

The Linux operating system of *Ubuntu 22.04* is chosen. The other important aspects of the Dockerfile include the installation of dependencies, Node Version Manager installation, git cloning of the repository, angular, typescript and npm installation with the command to run the web application.

Conference'17, July 2017, Washington, DC, USA
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

3.2 Docker Compose

Docker-compose file acts as a link between the web application API and the mongodb database. The main contents of this file are the mongodb image and the web application API based on the Dockerfile. The package.json file has also been added as it was updated by the authors in the recent changes.

3.3 Design Decisions

Building a successful reproduction package requires lot of design decisions. Initially, I tried to create a mongodb image inside the Dockerfile itself. This method did not work as it could not create a mongodb image. Hence, a docker-compose file was created which would help in interaction between the web application and the mongodb database.

Docker image building after every major step is the best debugging technique. It helps the programmer to identify the errors and its reason. The error logs helped a lot in finding the problem and web search helped to find the solution to it.

Node Version Manager was necessary as the project requires specific version to be installed. This version was curled using the link - *https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh.* A seperate directory was created in the docker container to install the same.

During the course of project development, the authors have updated the contents in their Github repository. Cloning the repository with latest developments would not build the Dockerfile. Hence, the commit made by the authors before I started with the project was chosen as a threshold. This was done using *git checkout* command.

3.4 Building the Docker Image

Docker-compose acts as the main file that creates a mongodb image and then builds the Dockerfile with the contents required for the web application to work. The user needs to just download the Dockerfile and docker-compose and then run the command **docker-compose build** to build the project. The command **docker-compose up** runs the docker image and starts the web application and **docker-compose down** stops the docker image.

The building of the Dockerfile consumes upto 2 hours. Most of the time is taken by the dependencies installation step. To be specific, the *r-cran* packages significantly increase the build time. After the image is built and started, it takes approximately 5 more minutes to establish secure connection and get all the things in place for the web application to work.

4 CHALLENGES

Reproduction of a project is a challenge in itself. Especially the version of the packages used by the authors. As this paper was published in 2018, some of the versions were outdated and hence had to be explicitly defined in the Dockerfile such as *node-gyp* having version of *3.8.0.* The npm would install the latest version and then docker build would not be successful.

As described earlier, Node Version Manager (NVM) was used to install *node v6.11.2.* This was one of the most time consuming task involving extensive research on installation of older node versions.

Another notable challenge was solving the segfault error. This error was mitigated using trial and error method. The installation of r-cran packages solved the error but also increased the build time significantly.

The datasets used by the authors was not easily available. The authors sent the datasets used by them via email after I requested them to share.

After the image is created another issue was to create the mongodb collections to store the datasets. The collections could not be created at the time of writing the report. Tried out various methods, like creating a Python file and shell script for creating collections, mongo-seed but all of them resulted in failure. Tried out atleast 20 methods given on the web to solve this problem but there is no success yet.

5 WORKING OF THE PACKAGE

The web application runs on *localhost:3000* which happens to be the default port for node applications. Mongodb image runs on *localhost:27017* which is the default port for mongodb. The web application requires you to specify the name, port, database name and collection name of the mongodb database. The web application is in Portuguese language but it can be easily translated into language of choice using the translate option on the web browser. The best part is that it is very simple to use.

6 DISCUSSION

6.1 Limitations

The creation of the mongodb database collections was unsucessful hence the results of the authors could not be verified. I believe that if this issue is solved then it is very easy to verify the results.

6.2 Conclusion

Reproducibility of any previous projects is not an easy task especially in this dynamic world where the packages and their versions are updated frequently. The biggest challenge according to me is to find the right version, connect the components sucessfully and then build the application. Although, the creation of collections in the mongodb database did not go well, the authors have provided lot of materials to make their project reproducible and check the results. The only claim by the authors that could not be verified is the results they obtained. As future work, I would try to create a collection and test their datasets for the results.

7 ACKNOWLEDGEMENTS

I would like to thank Prof. Stefanie Scherzinger to give me a chance to work on this reproducibility package and making this wonderful course. I thank the authors of this project for making the Github repository available with all the necessary dependencies to reproduce the package.

REFERENCES

[1] Angelo Augusto Frozza, Ronaldo dos Santos Mello, and Felipe de Souza da Costa. 2018. An Approach for Schema Extraction of JSON and Extended JSON Document Collections. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*. 356–363. https://doi.org/10.1109/IRI.2018.00060

[2] Lanjun Wang, Shuo Zhang, Juwei Shi, Limei Jiao, Oktie Hassanzadeh, Jia Zou, and Chen Wangz. 2015. Schema Management for Document Stores. *Proc. VLDB Endow.* 8, 9 (may 2015), 922–933. https://doi.org/10.14778/2777598.2777601

[1] [2]