Designing a Data Logistics and Model Deployment Service

Jedrzej Rybicki
Juelich Supercomputing Center (JSC)
Juelich, Germany
Email: j.rybicki@fz-juelich.de

***Abstract*—In Big Data applications, it is often required to integrate data from different sources to fuel machine learning models. In this paper, we describe a prototype implementation of the data logistics and model deployment services. Our goal was to create a one stop shop solution to support generic Data Science life cycle. It starts from formalized and repeatable data selection and processing provided by the data logistic service. The data are used for model creation in a typical machine learning fashion. The model is then put into a model repository to enable easy model management, sharing, and deployment. The functionality of the proposed prototype is positively verified with a particular use case from environmental science.**

Keywords–Data Science; Big Data; Machine Learning; Model Repository.

I. INTRODUCTION

Data Science is a way of obtaining novel insights from collected data. The process is propelled by two main forces: large amounts of data and analysis methods subsumed under the term *machine learning*. There are many ways of defining the Data Science process [1], but for the sake of argumentation, we can reduce it to three main phases: data preparation, model creation, and model deployment. Each of the phases poses some unique challenges. The modelling phase probably attracts most of the attention. This part unifies approaches from applied computer science, statistics, artificial intelligence, and many more popular scientific fields. Yet, the phase cannot be conducted efficiently without the data collection phase, and it is not very useful if the created model is not put into production. Therefore, in this paper we focus on the data collection and model deployment and propose a solution, which is sufficiently generic to accommodate different kinds of models.

The quality of the outputs of a Data Science project is mainly resulting from the quality and amounts of the input data used (rather than a sophistication of the used model). Thus, in the process of data preparation, one has to make sure to collect as much relevant data as possible. Just as in the physical world, a factory needs to be timely supplied with all the production means it requires, and the quality of the products depends on the resources used. The problem in physical world is solved by logistics. Along these lines, in this paper we propose a *data logistics service* responsible for timely delivery of the data to the models.

Collection of the data requires access to many sources. Furthermore, the data have to be cleansed to ensure their quality. Also, a higher-level processing is often required, for instance, to transform the data into a different format. In our experience, the process of data preparation takes a lot of time and effort, and yet becomes little acknowledgment because it is regarded as a mundane and less important process than modeling. The challenges posed in the phase of data collection are further reinforced by the fact that in many cases new data becomes available during the process as data collections evolve over time. This is especially important for the forecasting models, as their output might be more dependent on the most up-to-date information rather than the historical ones. The model performance depends on the *data freshness*. It is generally considered a bad practice to perform data collection and processing in a manual way [2]. Rather a formalization of the process in form of programming scripts shall be sought after. Programmatic approach helps in understanding and repeating the process of data collection and can also be crucial for efficient provenance tracking. The availability of programs and scripts for automated data collection and processing does not alone solve the problem of keeping track of data changes. Because of the aforementioned requirement of data freshness, the data collection is not a one-off act but rather a repeatable action. Thus, the programs and scripts have to be executed periodically, and monitored to detect progress, errors, and problems. In this paper, we propose an approach based on Apache Airflow [3] to implement data logistic service to gather and process data in an automatic, repeatable, and user-friendly way.

Second phase we would like to focus our paper on, is the model deployment phase. It follows the phase of model selection, tuning, and training. This is often done, at least partly, in an interactive way with tools like Jupyter Notebook [4], or Zeppelin [5]. Such tools are second to none in terms of user friendliness and quick turn over times (at least for small models). As soon as a promising model is found and its basic parameters are set, a more laborious phase of model training follows. Roughly speaking, this process sets up the model internal parameters to try to fit the collected empirical data as good as possible. Depending on the size of the data and complexity of the model the training can take substantial amounts of time. The trained model should be then put into production to accomplish the work it was intended to do. The production can be a support of an interactive web application where the model does the predictions, classifications, visualizations, etc. Given the dynamic nature of the data used in most Data Science projects, the model may require a retraining to account for the newly collected information. Sometimes also an adjustment of the parameters, or even change of the model class is required. In this paper, we show how the training of a model can be incorporated in the proposed data logistics service, and also, how the trained models can be put in MLflow [6] model repository to enable easy model sharing, review, and deployment. Our goal is to provide a one-stop shop solution to support complete Data Science life cycle.

Our high-level motivation is based on two observations. Models created in scientific endeavours should be verifiable

 ISBN: 978-1-61208-775-7

by other researchers. Such a verification can be conducted also by applying given model to a new set of data. We believe that our approach can be helpful here. Secondly, we observe increasing asymmetry between resource usage of model training and prediction. Complex models, e.g., neural networks driven by large amounts of data often require large amounts of special kinds of hardware for efficient training. Yet a prediction with such models are pretty quick even with simple hardware. Also, for these purposes, a model repository with model deployment functionality can be beneficial. It allows for large research organizations to share their (often expensive) specialized hardware and results it produces.

The rest of the paper is structured as follows. We firstly, summarize the use case that motivated our work in Section II. We then proceed with the description of the system design in Section III, where we describe both the data logistic and model deployment services and their interplay. The created solution is evaluated in Section IV. Subsequently, we shortly discuss related works in Section V, before summarizing the paper in Section VI.

II. USE CASE DESCRIPTION

In this paper, we propose a system to support typical tasks in a Data Science project. In particular, we cover the data preparation and model deployment phases. These phases occur in many standardized Data Science life cycles (even if under different names) like Cross Industry Standard Process for Data Mining [7] or Team Data Science Process [8]. To better understand how the proposed solution can facilitate efficient Data Science endeavors, let us describe a use case that motivated our implementation.

Firstly, our goal was to put the relevant data in a target database. Subsequently, the database was used for training a machine learning model, which was then put into production to conduct forecasting. Our data source was the OpenAQ Platform [9]. It collects measurement of following pollutant types PM10, PM2.5, sulfur dioxide (SO2), carbon monoxide (CO), nitrogen dioxide (NO2), ozone (O3), or black carbon (BC). OpenAQ stores raw data from measuring stations operated by government entities or international organizations across the world. The data are accessible through an API and also put in a public storage based on Amazon Simple Storage Service (S3) [10]. OpenAQ publishes data in different formats and with different time resolutions, we were interested in the most current ones, i.e., the real-time version published every 10 minutes to S3 [11].

Our target database was Tropospheric Ozone Assessment Report (TOAR), which is a relational database of global surface ozone observations emerging from a cooperation among many data centers and individual researchers worldwide. It combines data from over 10 000 measuring stations, allowing for sophisticated analysis of ozone concentrations in troposphere. Ozone is relevant for both human health and environment [12]. Access to the collected data is granted through Jülich Open Web Interface for accessing TOAR surface ozone data [13]. OpenAQ shall become one more of many sources of data integrated into the TOAR database.

Two main challenges with respect to data management were to keep them up-to-date and transform data from OpenAQ into a new TOAR format. Roughly speaking, the TOAR database is built around the notion of measurement series stored in a relational database, whereas OpenAQ collects single measurements stored in compressed NDJSON [14] format. Such discrepancies are typical in real life and have to be often addressed in the data collection phase.

The target database was used to retrieve relevant measurement series, which in turn were used to train a model for predicting air quality in a given area. The model was deployed and served as an analytic backend for a web application. We intentionally omit some details regarding the actual model and its usage, this part belongs to a different Data Science life cycle phase, which lays outside of the scope of this paper. The presented use case comes from a scientific field of environmental science, but we believe that the principles apply in other scientific fields and also outside of the academia, where Data Science approaches become more and more popular. The data flow in our use cases is schematically depicted on Figure 1.

III. SYSTEM DESIGN

In this section, we describe the design and implementation of the proposed solution. Its two main parts are data logistics service and model repository with deployment function. Although, as we pointed out, the parts support distinct phases of Data Science life cycle, there is also an overlap between them. For instance, a model can only be deployed when it passed the training phase fuelled by the delivered data.

A. Data logistics

Our solution for data logistic is based on Apache Airflow [3]. It is a platform to programmatically author, schedule, and monitor workflows. Workflows are defined as Directed Acyclic graphs (DAGs), which comprise of *Operators* and additional metadata defining, e.g., execution frequency. A unique feature of Airflow, when comparing to well-known workflow systems like Taverna [15] or Kepler [16], is that it does not use a product-specific language for defining workflows, but rather uses standard Python programming language. This allows for more flexibility in terms of task and dependencies and also lowers the entry barrier for the new users.

The way the Airflow workflows are executed differs from the aforementioned workflow systems. DAG's Operators are instantiated to become Tasks, which are then passed through a messaging queue to the Worker nodes for execution. The number of workers in the system can be changed depending on the workload. Operators abstract different kind of tasks and constitute extension points in system. There are three kind of Operators in Airflow: actions, data transfers, and sensor. Sensors wait and detect a particular event, e.g., publication of new data. Data transfer operators move data to and from particular system (like database, object store, etc.). Finally, the action operators execute particular action in remote environment, for instance `DockerOpetator`,`SparkOperator`, `BashOperator`, `SSHOperator`.

Airflow has a very unique yet powerful approach to the repeatable tasks. To enable reproducibility of the workflows their dependency on time is reduced. Each DAG has to have a `start_date` and `schedule_interval`. The `end_date` is optional and if no value is provided a date in the future is used to facilitate perpetual repeating workflows. The interval is divided into smaller parts, each of `schedule_interval` length. For each of the parts, one DAG run is created and executed. The time variable is injected into the tasks of the workflow. The tasks must be implemented in such a way that they should rely on the execution time provided by

 ISBN: 978-1-61208-775-7

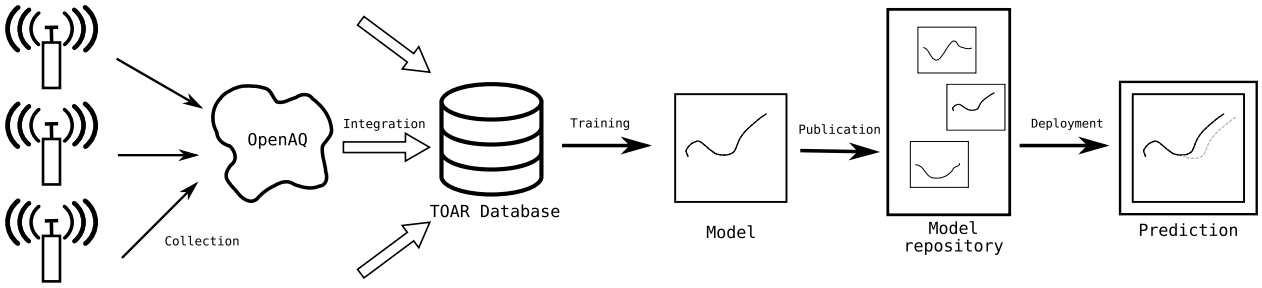

Figure 1. Data and computation flow in the modeled use case.

the workflow system (rather than other means like operating system date or time values). Thus, upon failure of a workflow it is possible to restart it at later date. Also, changes in the workflow or single tasks can be easily implemented and then rerun in an efficient way.

The workflow for the retrieval of the OpenAQ data and upload it TOAR database is composed of three main tasks. Firstly, a list of objects in S3 is created and filtered so that only objects from the injected time interval of ten minutes are considered in the next steps. Subsequently, the identified compressed NDJSON objects are downloaded and temporarily stored in a distributed file system. Lastly, the files are analyzed and uploaded to the target TOAR's Postgres database. During the last task, single measurements from OpenAQ are analyzed. If they refer to a measuring station, which is present in the target database, the measurement is added to the existing measurement series, otherwise a new series is created. Unfortunately, the stations in OpenAQ do not have a unique identifiers. Their names are given by the station operators and can even change over time. Therefore, we decided to use coordinates to identify the stations. This worked for most of the cases, stations with no coordinates were discarded.

Since new measurements are published every 10 minutes to S3 buckets, the workflow has to be rerun periodically. This part is taken care of by Airflow scheduler. To speed-up the processing, some of the tasks are implemented in a parallel fashion: NDJSON objects are split into chunks, which are processed in parallel.

B. Model creation

The availability of the data marks a starting point, at which training of a model for air quality forecast becomes possible. The process is manifold. Firstly, the relevant data are selected from the database. We are interested only in measurements regarding particular station. Secondly, a simple *RandomForestRegressor* model from Python `Scikit-learn` [17] package is trained. It is worth stressing that we use this very simple model only to show case how our solution works. In reality, the scientists doing the analysis would be using much more data and much more sophisticated models. The input data for the model is pulled from a temporary data table. The reason for this is the flexibility to use the same train code with different data. The progress of the training process is traced with help of a `MLflow` server [6]. With such a server, it is possible to store model parameters, metrics like mean square error, and model artifacts, e.g., serialized model.

We implemented the model creation as an Airflow DAG. One of the challenges of such an approach is the dependency management. Although, we use a popular Python library (`Scikit-learn`), it might not be available at the Ariflow workers executing the model training task. The `MLflow` offers help here. It is possible to create `MLflow` Projects that comprise not only of the code for the model creation but also metadata to define its dependencies. For Python, `conda` can be used, which is a well-established and mature package, dependency and environment management solution [18]. It will, upon project execution, take care of downloading and installing all the required libraries. This solution also works with other programming languages and libraries.

It is worth stressing that our approach is pretty flexible. The created Airflow DAG is capable of running different `MLflow` projects, which train the model. Such projects can be stored in GitHub [19] repositories from which they are retrieved for execution. The only constrain that we put on the projects is the convention for data retrieval. In our case, it is assumed that the data will be in placed in a temporary database table. The address of the database is injected to the `MLflow` project through environment variables. At the same time, the `MLflow` Projects can be executed outside of our data logistics service, e.g., on a local machine in a early phase of model selection.

C. Model deployment

A nice side effect of using `MLflow` [6] to store the models is an ability to instantiate such models in an easy way. For this, a single command is required:

```
mlflow models serve
  -m runs:/98ec38b6b846/model
  -p 8081
```

Each model registered with the `MLflow` has its unique run id, which can be used to instantiate it as in the command above. Models are decorated with a REST interface, which is accessible at given port (`-p 8081`). To this endpoint, a request with data in JSON format can be sent. The data will be passed over to the model for predictions and the results are sent back to the client.

IV. Evaluation

Despite this paper being a work in progress record, we decided to include some preliminary evaluation of the system performance. To this end, we used data from the OpenAQ data repository and analyzed 10 subsequent days starting from 1. January 2014. For the execution of a test system, we override Airflow configuration to reduce the task parallelism to 1 to eliminate the possibility of DAGs interleaving. Figure 2 shows the Task run times for the most important tasks in the DAG:

 ISBN: 978-1-61208-775-7

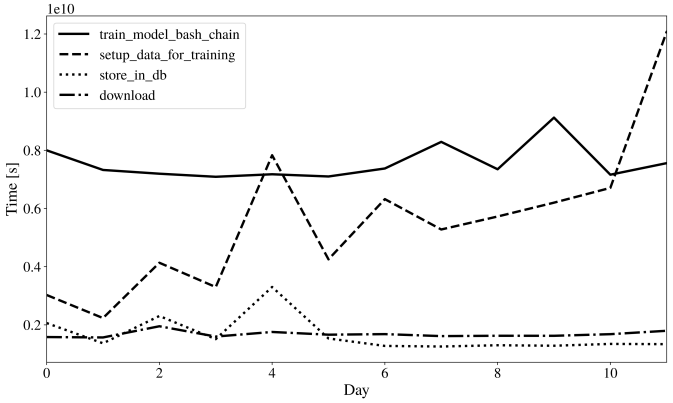

Figure 2. Duration of workflow tasks.

download of the data, data conversion and database upload, data selection for training, model training and publication. Apparently, the data download and conversion take constant amount of time, similarly to training and model upload. This is true, despite the increasing amounts of data used for the training (36 measurements on first day, comparing to 345 on the last one). The only tasks that displays some worrisome scalability characteristics, is the data preparation step in which all measurements from one selected measuring station are retrieved from a database and made available for training. Perhaps the negative scalability trend can be overcame by some database optimization and shall be a subject of following works.

V. Related Work

Our work is partly motivated by the publication of Chen et al [20] who proposed Data-as-a-Service (DaaS) and Analytics-as-a-Service (AaaS). Our data logistic service could be understood as a type of DaaS, and some functionalities of the model repository can be used to offer Analytics services.

We already pointed out the differences between used solution for data logistics and typical workflow systems like Taverna [15] or Kepler [16] (see Section III). Airflow addresses different kind of use cases, focuses mostly on efficient data movement and integration. We think, that the proposed solution is orthogonal to the classical workflow systems, the later ones can be executed by Airflow, e.g., to facilitate model training where computation performance is of the primary interest.

The problem of model repository and efficient model deployment seems to be gaining attention lately. There is a patent describing an idea of model repository [21]. It provides, however, no implementation details. FBLearner Flow by Facebook [22], Google TensorFlow Extended [23], or Kubeflow [24] are infrastructure and framework specific solutions for model deployment and management. In our work, we were striving for a generic solution and also wanted to promote the idea of model sharing in academia. The Open Science movement was successful in promoting the idea of data repository. A truly open science requires publishing and sharing of the created models. An interesting work by Behrouz [25] focuses on continuous model deployment. At first glance the approach is complementary to the solution presented herein. The author provided a means for efficient model retraining, whereas we cover the remaining phases of collecting data, changing and redeploying of the model. We intend to verify if the proposed solution can alleviate the problem of limited scalability of data selection tasks observed in our evaluation.

VI. Conclusion and Future Work

In this paper, we proposed a prototypical implementation of the data logistics and model deployment services. The services were used to implement a real use case from environmental science. We described and explained our design decisions. The use case was implemented successfully and put into production.

In our future work, we plan to address some of the shortcomings of our current solution. More improvement is required in terms of efficient sharing of Airflow DAGs. This will help in implementing other use cases. Also, more sophisticated deployments (e.g., with monitoring and dynamic resource management) are planned. To this end, migration to Docker-based model execution might be a good idea.

References

[1] J. Rybicki, “Best practices in structuring data science projects,” in Proceedings of the International Conference on Information Systems Architecture and Technology. Springer, 2018, pp. 348–357, ISBN: 978-3-319-99992-0. [Online]. Available: https://doi.org/10.1007/978-3-319-99993-7_31

[2] G. Wilson et al., “Good enough practices in scientific computing,” PLOS Computational Biology, vol. 13, no. 6, Jun. 2017, pp. 1–20. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1005510

[3] Apache Airflow. [Online]. Available: https://airflow.apache.org/ [retrieved: 2020. 01 .01]

[4] Project Jupyter. [Online]. Available: https://jupyter.org/ [retrieved: 2020. 01 .01]

[5] Apache Zeppelin. [Online]. Available: https://zeppelin.apache.org/ [retrieved: 2020. 01 .01]

[6] MLflow. [Online]. Available: https://mlflow.org [retrieved: 2020. 01 .01]

[7] P. Chapman et al. CRISP-DM 1.0: Step-by-step data mining guide. [Online]. Available: ftp://ftp.software.ibm.com/software/analytics/spss/support/Modeler/Documentation/14/UserManual/CRISP-DM.pdf [retrieved: 2020. 01 .01]

[8] TDSP: Team data science process. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/overview [retrieved: 2020. 01 .01]

[9] OpenAQ platform (openaq.org). [Online]. Available: http://openaq.org/ [retrieved: 2020. 01 .01]

[10] D. Robinson, Amazon Web Services Made Simple: Learn How Amazon EC2, S3, SimpleDB and SQS Web Services Enables You to Reach Business Goals Faster. London, UK, UK: Emereo Pty Ltd, 2008, ISBN: 978-1-92157-306-4.

[11] OpenAQ daily fetches at S3. [Online]. Available: https://openaq-fetches.s3.amazonaws.com/index.html [retrieved: 2020. 01 .01]

[12] M. G. Schultz et al., “Tropospheric ozone assessment report: Database and metrics data of global surface ozone observations,” Elementa: Science of the Anthropocene, vol. 5, 2017, pp. 58–68. [Online]. Available: https://doi.org/10.1525/elementa.244

[13] Jülich Open Web Interface for accessing TOAR surface ozone data. [Online]. Available: https://join.fz-juelich.de/ [retrieved: 2020. 01 .01]

[14] NDJSON - Newline delimited JSON. Specification. [Online]. Available: https://github.com/ndjson/ndjson-spec [retrieved: 2020. 01 .01]

[15] T. Oinn et al., “Taverna: a tool for the composition and enactment of bioinformatics workflows,” Bioinformatics, vol. 20, no. 17, 06 2004, pp. 3045–3054, ISSN: 1367-4803. [Online]. Available: https://doi.org/10.1093/bioinformatics/bth361

[16] B. Ludäscher et al., “Scientific workflow management and the Kepler system,” Concurrency and Computation: Practice and Experience, vol. 18, no. 10, 2006, pp. 1039–1065. [Online]. Available: https://doi.org/10.1002/cpe.994

[17] F. Pedregosa et al., “Scikit-learn: Machine learning in Python,” Journal of Machine Learning Research, vol. 12, 2011, pp. 2825–2830, ISSN: 1533-7928.

[18] Conda. [Online]. Available: https://conda.io/ [retrieved: 2020. 01 .01]

[19] GitHub. [Online]. Available: https://github.com/ [retrieved: 2020. 01 .01]

[20] Y. Chen, J. Kreulen, M. Campbell, and C. Abrams, “Analytics ecosystem transformation: A force for business model innovation,” in Proceedings of the IEEE Annual SRII Global Conference, Mar. 2011, pp. 11–20, ISSN: 2166-0778.

[21] C. R. Chu and S. C. Tideman, “Model repository,” Patent US Patent 6,920,458, 2005.

[22] FBLearner Flow. [Online]. Available: https://engineering.fb.com/core-data/introducing-fblearner-flow-facebook-s-ai-backbone/ [retrieved: 2020. 01 .01]

[23] TensorFlow Extended (TFX). [Online]. Available: https://www.tensorflow.org/tfx/ [retrieved: 2020. 01 .01]

[24] Kubeflow. [Online]. Available: https://www.kubeflow.org/docs/about/kubeflow/ [retrieved: 2020. 01 .01]

[25] B. Derakhshan, “Continuous deployment of machine learning pipelines,” in Proceedings of 22nd International Conference on Extending Database Technology (CEDT’ 19), Mar. 2019, pp. 397–408.

 ISBN: 978-1-61208-775-7