Using DODAS as deployment manager for smart caching of CMS data management system

M. **Tracolli[1],** ***M.*** **Antonacci[2],** ***T.*** **Boccali[3], D. Bonacorsi[6],** ***D*** **Ciangottini[1],** ***G.*** **Donvito[2],** ***C.*** **Duma[4],** ***L.*** **Gaido[5],** ***D.*** **Salomoni[4],** ***D.*****Spiga[1], V. Kuznetsov[7]**

[1] Istituto Nazionale di Fisica Nucleare, 06123 Perugia, Italy
[2] Istituto Nazionale di Fisica Nucleare, 70126 Bari, Italy
[3] Istituto Nazionale di Fisica Nucleare, 56127 Pisa, Italy
[4] Istituto Nazionale di Fisica Nucleare CNAF, 40127 Bologna, Italy
[5] Istituto Nazionale di Fisica Nucleare, 10125 Torino, Italy
[6] University of Bologna, Italy
[7] Cornell University (US)

Mirco.Tracolli@pg.infn.it

Abstract. DODAS stands for Dynamic On Demand Analysis Service and is a Platform as a Service toolkit built around several EOSC-hub services designed to instantiate and configure on-demand container-based clusters over public or private Cloud resources. It automates the whole workflow from service provisioning to the configuration and setup of software applications. Therefore, such a solution allows using “any cloud provider”, with almost zero effort. In this paper, we demonstrate how DODAS can be adopted as a deployment manager to set up and manage the compute resources and services required to develop an AI solution for smart data caching. The smart caching layer may reduce the operational cost and increase flexibility with respect to regular centrally managed storage of the current CMS computing model. The cache space should be dynamically populated with the most requested data. In addition, clustering such caching systems will allow to operate them as a Content Delivery System between data providers and end-users. Moreover, a geographically distributed caching layer will be functional also to a data-lake based model, where many satellite computing centers might appear and disappear dynamically. In this context, our strategy is to develop a flexible and automated AI environment for smart management of the content of such clustered cache system. In this contribution, we will describe the identified computational phases required for the AI environment implementation, as well as the related DODAS integration. Therefore we will start with the overview of the architecture for the pre-processing step, based on Spark, which has the role to prepare data for a Machine Learning technique. A focus will be given on the automation implemented through DODAS. Then, we will show how to train an AI-based smart cache and how we implemented a training facility managed through DODAS. Finally, we provide an overview of the inference system, based on the CMS-TensorFlow as a Service and also deployed as a DODAS service.

1. Introduction

In the current computing model of the CMS experiment at LHC recorded and simulated data are distributed and replicated via a central management system across a worldwide network of custodial storage sites. A huge increase of both storage and computing requirements are foreseen for the HL-LHC era (between 1 and 2 orders of magnitude) and new kind of resource is covering a crescent amount of needs (e.g. private or public cloud and HPC facilities). Thus CMS experiment has been pushed towards evaluating an evolution for the optimization of the amount of space that is managed centrally and the CPU efficiency of the jobs that run on “storage-less” resources. The goal would be a geographically distributed cache storage based on unmanaged resources, with a consequent reduction of the operational efforts for maintaining managed custodial storages and increasing the flexibility of the system. The cache system will appear as a distributed and shared file system populated with the most requested data; in case of missing information data access will fallback to the remote access. Moreover, in a possible future scenario where a data-lake model will be implemented, a protection layer against centrally managed storages might be a key factor along with the control on data access latency. The cache storages used for such a layer will be, by definition, "non-custodial", thus reducing the overall operational costs. In order to optimize the cache management, orchestration and usage we decided to develop an AI-based strategy. A prerequisite to attain such an AI-based approach is to enable a computation platform that exploits the whole toolchain. The workflow can be treated as a standard Machine Learning (ML) task, as such there are three main tasks that need to be addressed: data pre-processing, model training and model Inferencing. In this paper, we describe how we implemented the computational stack, named “smart decision services” using Dynamic On Demand Analysis Service (DODAS)[7] as enabling technology. We describe initially the reasons why we adopted DODAS (section 2), then we detail the implementation used to address the CMS data cache problem, describing the whole toolchain and the details about the infrastructure. We will point out the feasibility of the environment and the usability improvement with this configuration. In particular, section 3 will present the complete workflow of the smart decision service, explaining the various phases, from the environment to the model creation. Section 4 is dedicated to the integration with a cache system. Finally, in section 5, we expose our conclusions and future plans.

2. **Using DODAS as enabling technology**

DODAS is an open source project aimed to create and configure on-demand clusters. It is one of the so-called Thematic Services of the EOSC-hub project[3]. DODAS is a Platform as a Service toolkit that helps to customize a cluster service on any cloud infrastructure with almost no effort and has been designed to work also on hybrid clouds implementing a high level of automation. Its approach is a Zero Ops model to get the user a tailored environment ready to go. As shown in Fig. 1, DODAS has a highly modular architecture and its workflows are highly customizable. For this reason, it is very extensible, spanning from software dependencies up to the integration of external services, including also user-tailored code management. There are four main pillars in the DODAS architecture which can be summarized as:

- abstraction: both in terms of software application and dependency description, and of underlying IaaS level cloud infrastructures;
- automation: it refers to software and application setup in order to manage resources and orchestrate software applications;
- multi-cloud support: to deal with multiple heterogeneous Cloud infrastructures;
- flexible Authentication and Authorization Infrastructure (AAI): authentication, authorization, delegation, and credential translation primitives providing a secure composition of the various services participating in the DODAS workflow. Authentication and authorization implementation is based on INDIGO-IAM[4][5] that can adapt to whatever needs.

DODAS uses container paradigm to provide user services, and resource management relies on Apache Mesos or Kubernetes. Mesos is also distributed with Apache Marathon to manage long-running services. Both configurations use Docker as container technology. The end users can access directly to the services built for them; cluster managers can easily customize and control the cluster.

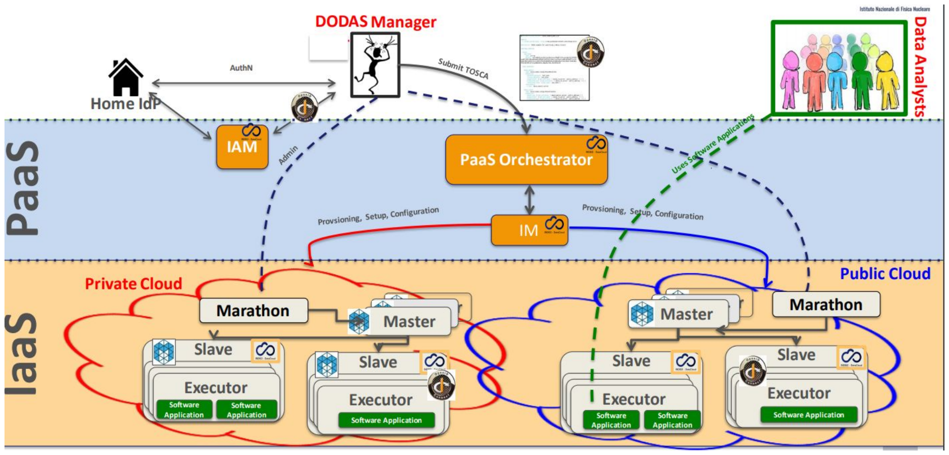


Fig. 1. A high-level schema of the DODAS architecture. Three different colors identify User Domain (white), DODAS PaaS core service layer (light blue), and resource layer (dark yellow).

DODAS is a deployment manager which provides a highly customizable platform that guarantees flexibility and extensibility in order to cope with highly specific use cases. These are the main motivations for us to adopt such a solution as enabling technology to develop the smart decision service.
Moreover, DODAS already provides generic recipes for deploying Big Data software and libraries such as Apache Hadoop and Apache Spark as shown in Fig 2. Finally, a key motivation for us to adopt DODAS is that it provides recipes for a custom ML environment which in turn allows extension and customization in order to easily integrate PyTorch[8] respect TensorFlow[9] or add Parquet or Hive support to Spark, plus other that might come in the near future.

3. **Implemented workflow**

The implemented workflow comply with a standard ML workflow with its well-known tasks: Data Pre-processing, Model Training and Model Inferencing. Internally each of these tasks can consist of sub-tasks that contribute to the generation of the final output for that phase. Furthermore, these phases could be iterated several times: with a new input, we can call a new pre-processing, train another model and serve it in a sort of continuous training approach.
From a logical perspective, there are two phases: the first phase prepares the raw data to be used into an ML framework; the second phase provides the trained model accessible through a normal public service. This is where the cache infrastructure will interact with the smart decision service e.g. via the HTTP protocol.

3.1. *Computational Environment*

In order to build the computational environment, we selected Apache Mesos[10] to manage the resources. It is used especially as a resource manager for Apache Spark, that is available in client mode in the cluster. The main configuration provides a Jupyter notebook service that is used as a terminal for the various frameworks in the cluster. With this terminal, the end users can write their code and use the infrastructure to process the data.

The implementation can interact with an Apache Hadoop system that is available for the training process but can also be accessed in the further steps. Additionally, TensorFlow framework with Keras[11] has been made available: these are used to create and save the model for the inference in a TensorFlow as a Service (TFaaS) component, as shown in Fig. 2. In particular, the TFaaS uses TensorFlow as engine and it is written in Go and exposes an HTTP API interface to interact with. Moreover, there are several modules written in Python that can help the users to interact with the toolchain previously illustrated.

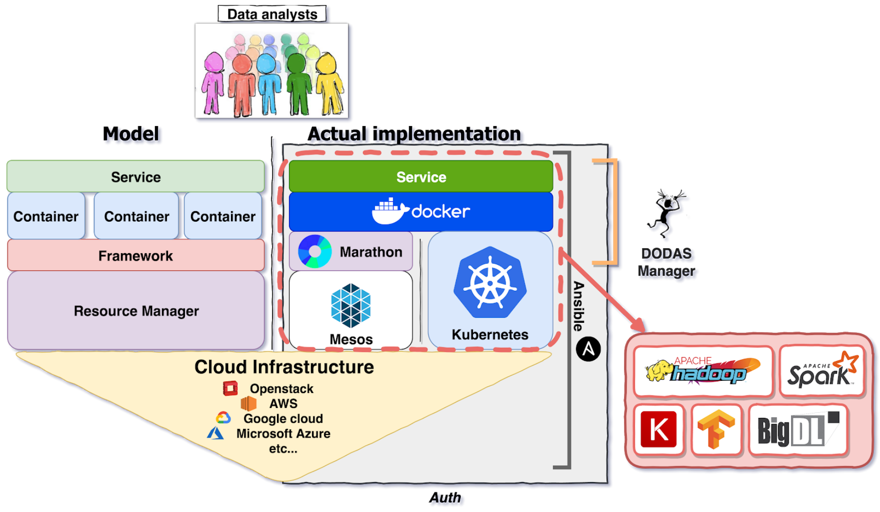


Fig. 2. Schematic view of the abstraction layers implemented by DODAS on top of a generic cloud provider.

3.2. *Data flow and pre-processing*

The very first step is to process data in order to get them ready for the ML phase. This is the step where we filter and extract the proper features for the training. There are two types of data source: historical data or real-time data. An utility named Data Manager, helps the user to interact with these sources; it is completely customizable and can be extended to support different sources. A schematic summary of the overall data flow is represented in Fig. 3.

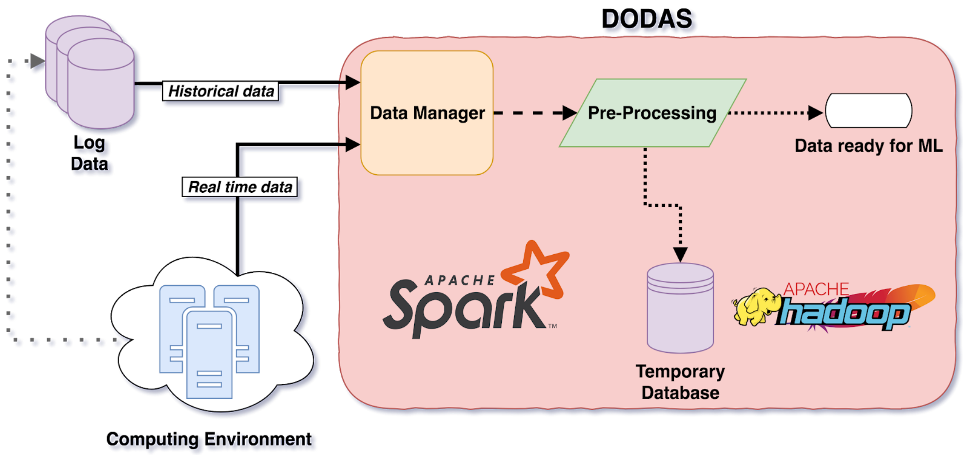
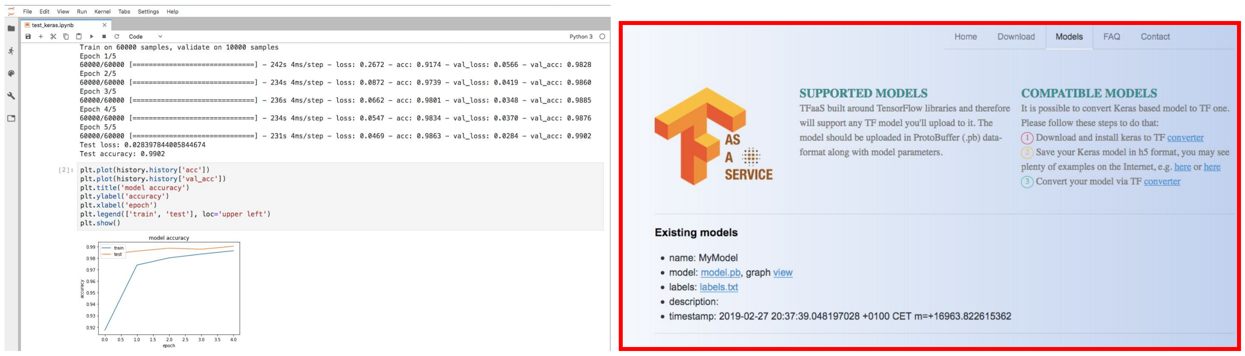


Fig. 3. Schematic view of the pre-processing phase implemented with DODAS.

The pre-processing task completely relies on Spark and produces some intermediate data ready for the ML phase. These data can be stored in Hadoop or in different databases that the Data Manager can interact with. As you can see in the screenshots (Fig. 4), the Spark interaction with Apache Mesos[10] is completely transparent to the users: the Spark workflow does not change.

3.3. *Training models*

Thanks to the integration with Jupyter notebook, it is possible to verify the data with some preliminary steps, before the real training, for example analyzing and visualizing online the records processed in the previous phase. Then, the data created with the pre-processing phase will be used to create a model, that could be loaded directly onto the inference service. An example of such a model could be a Neural Network that works on images. The users can benchmark the created models before making them accessible to external services through the TFaaS and besides, after the upload, users can continue to experiment and test their own models. In conclusion, the results from these steps will have TensorFlow compatible model that we can upload in the final service with the HTTP API as shown in Fig. 4.



```
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 242s 4ms/step - loss: 0.2672 - acc: 0.9174 - val_loss: 0.0566 - val_acc: 0.9828
Epoch 2/5
60000/60000 [==============================] - 234s 4ms/step - loss: 0.0872 - acc: 0.9739 - val_loss: 0.0419 - val_acc: 0.9860
Epoch 3/5
60000/60000 [==============================] - 236s 4ms/step - loss: 0.0662 - acc: 0.9801 - val_loss: 0.0348 - val_acc: 0.9885
Epoch 4/5
60000/60000 [==============================] - 234s 4ms/step - loss: 0.0547 - acc: 0.9834 - val_loss: 0.0370 - val_acc: 0.9876
Epoch 5/5
60000/60000 [==============================] - 231s 4ms/step - loss: 0.0469 - acc: 0.9863 - val_loss: 0.0284 - val_acc: 0.9902
Test loss: 0.028397844005844674
Test accuracy: 0.9902
```

```
[2]: plt.plot(history.history['acc'])
     plt.plot(history.history['val_acc'])
     plt.title('model accuracy')
     plt.ylabel('accuracy')
     plt.xlabel('epoch')
     plt.legend(['train', 'test'], loc='upper left')
     plt.show()
```

SUPPORTED MODELS

TFaaS built around TensorFlow libraries and therefore will support any TF model you'll upload to it. The model should be uploaded in ProtoBuffer (.pb) data-format along with model parameters.

COMPATIBLE MODELS

It is possible to convert Keras based model to TF one. Please follow these steps to do that:

1. Download and install keras to TF converter
2. Save your Keras model in h5 format, you may see plenty of examples on the Internet, e.g. here or here
3. Convert your model via TF converter

Existing models

- name: MyModel
- model: model.pb, graph view
- labels: labels.txt
- description:
- timestamp: 2019-02-27 20:37:39.048197028 +0100 CET m=+16963.822615362

Fig. 4. Screenshots of the Jupyter notebook and the inference service with TensorFlow (CMS TFaaS[2]).

3.4. *Performing inference*

The inference service is implemented using the CMS TFaaS[2] (Fig. 4). It is a Software as a Service based on TensorFlow framework for ML written in Go and exposes an API through the HTTP protocol. There are some basic manager operations, for example: store a model in the inference service, delete a model and ask for the list of the models already present. Also, we have some specific endpoints to trigger the inference with input data. The result is a JSON file with the content of the inference based on the model requested. The execution of the model is completely server side, the end user needs only to put the input data in the request.

4. **Integration with data cache middleware**

The integration with the data cache middleware based on XRootD[6] has also been prototyped and a schematic view of the foreseen interactions is shown in Fig. 5. The integration with the XCache system needs an additional plugin that can access and use the inference service exposed by the DODAS based decision service. The plugin component acts as an intermediary between the client request and the Artificial Intelligence to manipulate the cache decisions. For example, if the AI gets as input the file that the user has requested, it can respond with an action like storing or not that file in the cache system. Furthermore, it will increment the data source and trigger a new processing flow to update the final AI for further requests. The client doesn't know about the AI plugin and interacts with the cache normally.

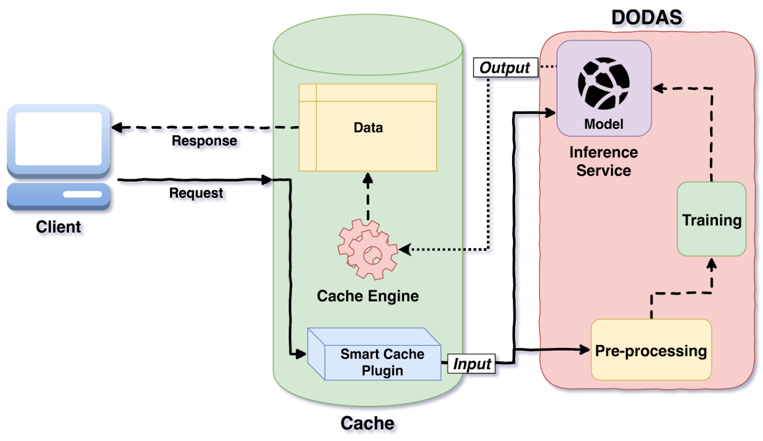


Fig. 5. Data cache integration schema.

5. Conclusion and plan for the future

This proof of concept implementation enables a smart data cache decision. The workflow needed to provide updated models have been described and the underlying technology used to develop the service has been shown. The first integration with CMS data cache solution based on XCache middleware has been presented and first tests of the end to end flow are promising: the current evaluation shows a possible up to 20% improvement. For the future, we want to research and develop a model for the proposed problem as such the main effort will be spent on data analysis and models development. Also, we want to study and compare the performances in order to measure the possible advantages an AI-based solution provides in term of operations costs. A benchmark study based on simulations has been planned to allow a more fine AI model in the final stages. Regarding the infrastructure, we want to improve the DODAS implementation to abstract the underlying interaction with the infrastructure and reach new targets such as better scalability, more automation, and a self-healing feature.

Acknowledgments
The authors would like to thank the European Commission's Horizon 2020 research and innovation programme for financial support, under grant agreement RIA 777536.

References

[1] CMS Collaboration, JINST 3 S08004 (2008)
[2] DOI: Valentin Kuznetsov. (2018, July 9). vkuznet/TFaaS: First public version (Version v01.00.06). Zenodo. http://doi.org/10.5281/zenodo.1308049
[3] EOSC Hub Project: https://eosc-hub.eu/
[4] Salomoni, D. et al., "INDIGO-DataCloud: a Platform to Facilitate Seamless Access to E-Infrastructures", Journal of Grid Computing, 2018, vol 16, num 3, pages 381-408
[5] eduGAIN: https://edugain.org/
[6] XRootd, disk-based, caching proxy for optimization of data access, data placement and data replication, CMS collaboration, J.Phys.Conf.Ser. 513 (2014) 042044
[7] D. Spiga et al. "DODAS: How to effectively exploit heterogeneous clouds for scientific computations", PoS(ISGC 2018 & FCDD)024, DOI: https://doi.org/10.22323/1.327.0024
[8] PyTorch: https://pytorch.org/
[9] Tensorflow: https://www.tensorflow.org/
[10] Apache Mesos: http://mesos.apache.org/
[11] Keras: https://keras.io/