

Data Bookkeeping Service 3 - A new event data catalog for CMS

This article has been downloaded from IOPscience. Please scroll down to see the full text article.

2012 J. Phys.: Conf. Ser. 396 052036

(<http://iopscience.iop.org/1742-6596/396/5/052036>)

View [the table of contents for this issue](#), or go to the [journal homepage](#) for more

Download details:

IP Address: 131.225.23.169

The article was downloaded on 01/08/2013 at 20:45

Please note that [terms and conditions apply](#).

Data Bookkeeping Service 3 - A new event data catalog for CMS

M Giffels¹, Y Guo² and D Riley³

¹ PH-CMG-CO. CERN, CH-1211 Genève 23, Switzerland

² Fermi National Accelerator Laboratory, Batavia, IL 60510, U.S.A.

³ Cornell University, Ithaca, New York, U.S.A.

E-mail: Manuel.Giffels@cern.ch, yuyi@fnal.gov, Daniel.Riley@cornell.edu

Abstract. The Data Bookkeeping Service (DBS) provides an event data catalog for Monte Carlo and recorded data of the Compact Muon Solenoid (CMS) Experiment at the Large Hadron Collider (LHC) at CERN, Geneva. It contains all the necessary information used for tracking datasets, like their processing history and associations between runs, files and datasets, on a large scale of about 10^5 datasets and more than 10^7 files. The DBS is widely used within CMS, since all kind of data-processing like Monte Carlo production, processing of recorded event data as well as physics analysis done by the user, are relying on the information stored in DBS.

1. Introduction

The CMS Data Bookkeeping Service (DBS) comprises databases and services used to store and access metadata related to CMS physics event data. In addition to recording what data exist, DBS also stores process-oriented provenance information, including parentage relationships of files and datasets, configurations of processing steps, and the associations with run number and luminosity section necessary to find a particular set of events within a dataset.

The CMS DBS is a federated system of services and databases, with several different DBS instances used for specific purposes. The Global-DBS records all official CMS data, real or simulated, that are available for physics analysis. The CMS Tier0 DBS instances records information for data from the detector as it is processed by the Tier0 facility. The CAF instance records data from the prompt reconstruction stream used for detector calibrations and diagnostics. Finally, physics analysis DBS instances let individual physicists or physics groups record and manage the results of their analysis steps. Physics analysis data determined to be of general use for a physics group can be repackaged and migrated to the Global-DBS via the StoreResults service [1] and the DBS Migration Service. This paper describes the motivation and design of the third major version of the system, DBS 3.

The second version of DBS [2], DBS 2, was designed and implemented in 2006-2007, prior to full operation of the LHC. At the time, CMS did not have a standardized services architecture, so the DBS 2 team chose to implement it using Java servlets in an Apache Tomcat container. Without a standard model for deploying new services, many requests were made for additional data to be stored in DBS 2 that were not entirely consistent with its original purpose. DBS 2 used XML RPC for client-server communications, and in some cases had very “thick” client APIs, which eventually lead to numerous problems with API versioning.

Since the time when DBS 2 was designed, the CMS data processing model has evolved in ways that were not anticipated by the DBS 2 design, and the CMS DataManagement/WorkflowManagement (DMWM) project had developed a standard architecture and deployment model for CMS web services. A 2009 project status review led to the decision to start work on DBS 3, to better match CMS requirements and integrate with other DMWM projects. Design goals for DBS 3 included: aligning the data model with the evolved CMS processing model and use cases; reset to the original project scope by spinning-off any services that were outside the project scope; simplified APIs and optimized database schemas based on observed DBS 2 usage, “logical” provenance, collapsing out the split and merge steps of the processing history that have no effect on the data content. An initial set of high and low level use cases, corresponding APIs, and an initial schema and architecture were developed over Summer 2010, followed by an intensive design review in September 2010.

After the design review, we built prototypes of various possible architectural and technology choices, eventually resolving on the current DBS 3 design and implementation.

2. DBS 3 - A RESTful Web service

For the current generation of web services, the CMS DMWM project has standardized on the Representational State Transfer (REST) architecture [3]. Choosing REST simplifies the task of integrating the different components of the DMWM system into a common architecture. REST also imposes a discipline of thin client interfaces. We largely standardized on the Java Script Object Notation (JSON) data-format for exchanging information, as a simpler alternative to the XML used in the previous generation. The DMWM project also standardized on Python as the implementation language, using the SQLAlchemy and CherryPy toolkits, and a standard “WMCORE” framework. Despite some initial skepticism, our prototyping convinced us that the chosen technologies could be competitive with the performance and reliability of the Java servlets used for DBS 2, while the advantages of using a common language and toolkit across all DMWM projects are obvious. By adopting the common DMWM architecture, DBS 3 transparently becomes a service provider for the CMS Data Aggregation Service (DAS) [4], which makes it possible to de-aggregate DBS into several services with narrower scope without significant loss of query functionality. DBS 3 also benefits from the work already done within the DMWM project on scalability, testing, and deployment.

Accordingly, DBS 3 has been completely re-designed and re-implemented in Python using the CherryPy and SQLAlchemy toolkits within the CMS standard DMWM framework. It uses RESTful interfaces (see Figure 1), where the API used is chosen by the path in the URI and the operation is chosen by the HTTP method. DBS 3 supports GET, POST and PUT operations. The deletion of data inside the catalog is not provided to ensure perpetual traceability. The client-server communication is stateless, which enhances the scalability of the service. An Oracle Database backend provided for CMS [5] is utilised as persistent storage in DBS 3. A standard layered architecture is used, with Data Access Objects (DAO) [6] at the lowest layer.

The database schema for DBS 3 was reexamined *de novo*, based on our use cases, usage statistics for DBS 2, and feedback from the design review. The DBS 3 schema particularly benefitted from the narrower and more precisely defined project scope compared to DBS 2. The narrower scope was largely made feasible by the integration of different services through the common DMWM architecture.

3. The deployment and packaging of DBS 3

DBS 3 is deployed on the CMS core system for web services (CMSWEB)[7] (see also Figure 1), which is developed, managed and operated by the HTTP-Group within the DMWM project. All hosted projects can profit from a common infrastructure providing sophisticated tools and documentation for the deployment and management of their projects on CMSWEB. To ensure

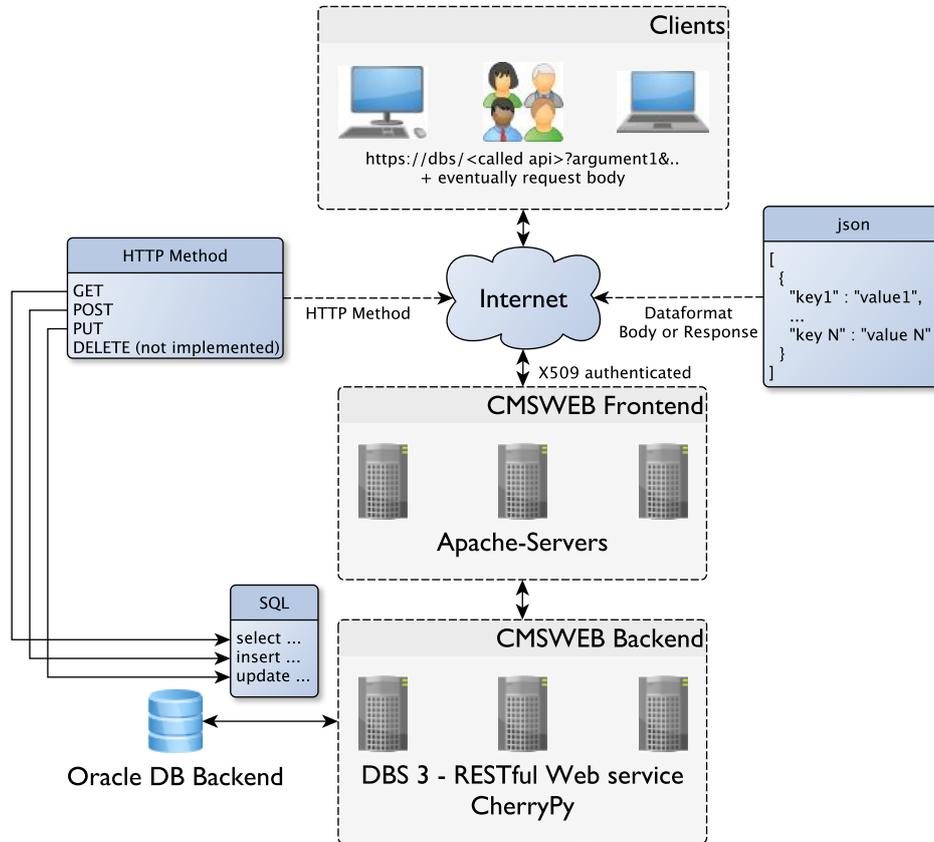


Figure 1. Function principle of the DBS 3 RESTful web service deployed on the CMSWEB cluster.

a secure operation, each project that will be deployed on CMSWEB, have to pass a full code-audit. Additionally, deployment and management scripts based on the common infrastructure are validated by the HTTP group. Beside the required tests during the deployment procedure, DBS 3 is already intensively tested on our development virtual machines using sophisticated test suites and procedures before the deployment is requested. Altogether, this procedure ensures that only well-functioning versions of DBS 3 are deployed.

The packaging of DBS 3 is done using common tools provided by CMS computing. The RPM Package Manager (RPM) is used together with the Advanced Packaging Tool (APT) as package management system. Any developer can request a private repository for building RPMs used in development and testing. RPM building in the private repository is semi-automated using the CMS package tools. Together with the developed DBS 3 setup scripts, each developer is able to create a new development environment within a short time. RPMs in the official repositories used for the deployment on CMSWEB are build using a fully automated tool called BuilderAgent developed within the DMWM project.

4. Performance and integration testing of DBS 3 using PhEDEx LifeCycleAgent

Most of the functionality of DBS 3 is already tested during development and deployment. However, that is not sufficient to ensure a well interoperability of all services relying on DBS 3. Therefore, additional performance and integration tests are required. During the process of

evaluation, it turned out that the PhEDEx LifeCycleAgent [8] is well suited to perform those tests.

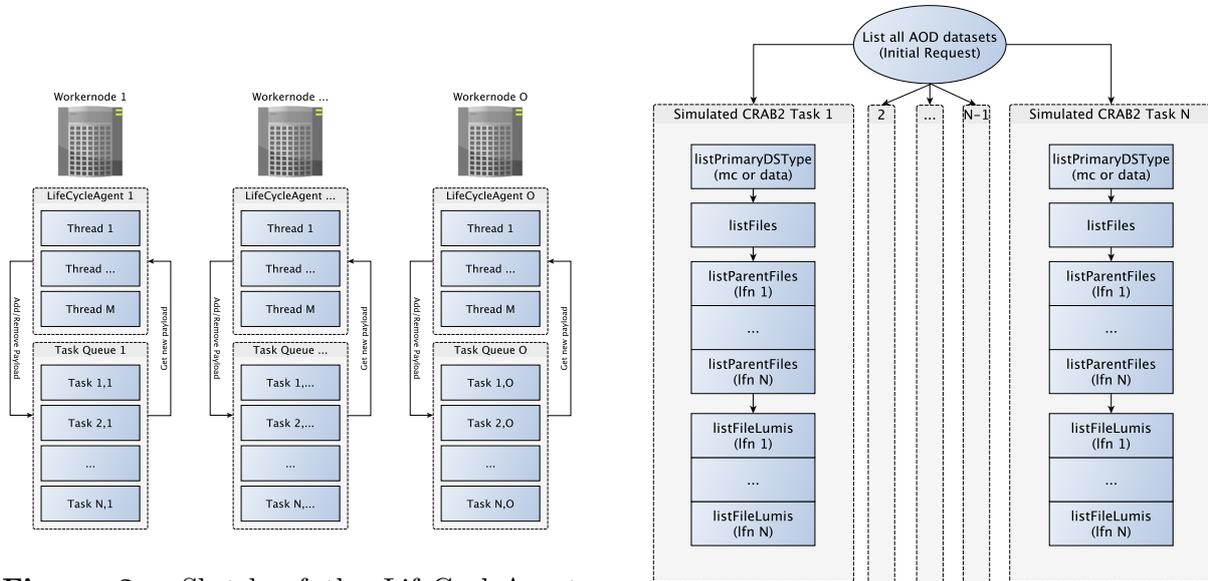


Figure 2. Sketch of the LifeCycleAgent working principle.

Figure 3. Realistic example of an analysis workflow defined in a LifeCycleAgent workflow.

The LifeCycleAgent was originally developed by the PhEDEx team to drive a realistic simulation of their data transfer system [9]. The LifeCycleAgent can execute functions in perl modules as well as external scripts, which makes it useable apart from PhEDEx, too.

The functioning of the LifeCycleAgent is depicted in Figure 2. The LifeCycleAgent uses an configurable number of threads to process its tasks. The individual tasks are fetched from a task queue. Each task is associated with a so called payload in JSON format. This payload can be modified within the task depending on its result. The modified payloads are then passed from one task to another, which enables the possibility to dynamically add tasks to the queue.

The interrelationship between the tasks is defined in a so called workflow. The workflow defined in Figure 3 comprises two different external python scripts. The first one is called as initial request inside LifeCycleAgent and asks DBS 3 to list all known Analysis Object Data (AOD) datasets. Inside that script an array of payloads is created, which means that the task is forked dynamically based on the result of the initial request, in other words N tasks are added to the task queue. The second external scripts calls several APIs of DBS 3. This workflow is a realistic simulation of N analysis users creating and submitting their analysis jobs to the Grid using the CMS Remote Analysis Builder (CRAB) [10]. This analysis workflow is used as a read-only stress-test of DBS 3.

In addition to the performance tests, integrations tests are needed, too. Recently a campaign between DAS, PhEDEx and DBS 3 has been started. The idea behind that campaign is depicted in Figure 4. A tool called DataProvider developed by Valentin Kuznetsov ensures that a consistent set of fake data are provided to the LifeCycleAgent. Different workflows for PhEDEx and DBS 3 are injecting that data into their systems. A third workflow uses DAS to check the consistency of the injected data between both subsystems. This also includes injections of false-positives. Once this integration tests are established, it is planned to include other projects within the DMWM realm as well.

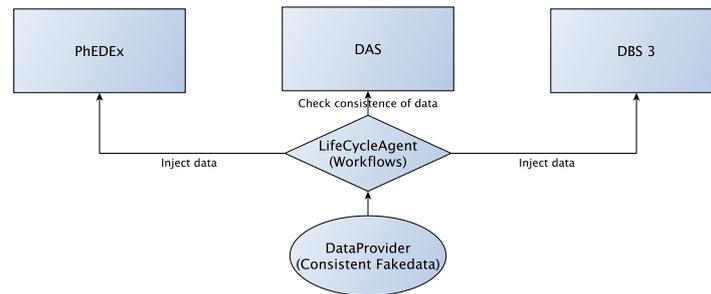


Figure 4. Sketch of the planned interaction tests between DAS, PhEDEx and DBS 3.

5. Conclusion

The development of DBS 3 was driven by the lessons learned from its predecessor DBS 2 and the ongoing revision of the DMWM software. DBS 3 better fits the needs of the evolving data processing model of CMS, since the needs were already settled before the design phase. In addition, DBS 3 is now better integrated into the realm of DMWM. So DBS 3 is taking advantage of the common infrastructure for deployment and packaging. The RESTful design using lightweight APIs and a stateless client-server communication ensures a better scalability of DBS 3 and it is also based on a common ground shared by multiple projects in DMWM. In addition to the development itself, a lot of effort is spent in the development of test suites, performance and integration tests between the interoperating projects, which helps to ensure that the product satisfies the requirements of CMS. DBS 3 is already deployed on CMSWEB, however it is not yet used in production. Developing a reliable concept of introducing DBS 3 to CMS is one of future challenges to address.

References

- [1] Giffels M, 2011, et al, Design and early experience with promoting user-created data in CMS, *J. Phys. Conf. Ser.* **331** 072049
- [2] Afaq A et al, 2008, The CMS Dataset Bookkeeping Service, *J. Phys. Conf. Ser.* **119** 072001
- [3] Fielding R T, 2000, Architectural Styles and the Design of Network-based Software Architectures, Dissertation, *University of California, Irvine*, ISBN: 0-599-87118-0
- [4] Ball G et al, 2011, Data Aggregation System: A system for information retrieval on demand over relational and non-relational distributed data sources, *J. Phys. Conf. Ser.* **331** 042029
- [5] Pfeiffer A et al., CMS experience with online and offline databases, paper [163] at this conference
- [6] Alur D, 2003, Core J2EE Patterns: Best Practices and Design, *Prentice Hall International*, ISBN: 978-0131422469
- [7] Metson S et al, 2008, CMS offline web tools, *J. Phys. Conf. Ser.* **219** 082007
- [8] Wildish T et al, From toolkit to framework - the past and future evolution of PhEDEx, paper [188] at this conference
- [9] Egeland R et al, 2010, PhEDEx data service, *J. Phys. Conf. Ser.* **219** 062010
- [10] Spiga D et al, 2007, The CMS Remote Analysis Builder (CRAB), *Lect. Notes Comput. Sci.* **4873** 580-586