

Migrate the CMS Production Workload Management system from CouchDB 1.6.x to CouchDB 3.1.x

Max Uetrecht¹

supervised by Alan Malta Rodrigues² & Todor Trendafilov Ivanov²

¹CERN Summer Student

`max.uetrecht@{cern.ch, protonmail.com}`

²CERN WMCORE Team

`{alan.malta, todor.trendafilov.ivanov}@cern.ch`

Period: July 05 - September 03, 2021

Abstract

The CMS Production Workload Management system drives all the Monte Carlo and data processing activities in the Experiment: through it millions of jobs are submitted to the World Wide Computing Grid every month, making it a crucial component in the CMS Computing Infrastructure. Another crucial technology in this ecosystem is the NoSQL database Apache CouchDB, which is largely used in the Workload Management System. This report summarizes the summer student work of the adoption of the latest CouchDB version 3.1.x in CMS Computing, ensuring that database replications and new features are explored at their best, as well as benchmarking and applying any possible optimizations to the system. At the end, a brief outlook on unfinished milestones and a potential CouchDB 4 migration is given.

Keywords: CMS, CMS Workload Management System, CouchDB, WMCORE, WMAgent

Contents

1	Introduction	2
1.1	RequestManager2	3
1.2	WorkQueue	3
1.3	WMStats	3
1.4	WMAgent	3
2	Milestones	4
2.1	Evaluate CouchDB features currently used by WMCORE	4
2.2	Document all the important changes between CouchDB 1.6.x and 3.1.x	4
2.3	Evaluate the cross-version compatibility of CouchDB's replication protocol	4
2.4	Creating a CouchDB 2.x Docker Image	4
2.5	Define a plan to migrate CouchDB databases from 1.x to 2.x	4
2.6	Get familiar with a Docker Image for CouchDB 3.1.x	5
2.7	Create new SPEC files for CMSDIST	5
2.8	Update existing Erlang patches for the CMS authentication layer	5
2.9	Create a development Docker Image with CouchDB 3.x	6
2.10	Identify and apply relevant changes to WMCORE	6
2.11	Build, deploy and test final setup	6
3	Conclusion	7
4	Future Work	7

1 Introduction

The goal of this project is to migrate the currently used CouchDB 1.6.1 to 3.1.x in the *CMS Workload Management System* (WMS) [7], which is part of the CMS Computing Model. To understand the importance of WMS and the interactions of the components involved, a closer look on CouchDB, the CMS Computing Model and relevant components of WMS will be taken.

CouchDB is an open-source *NoSQL*¹ database written in *Erlang*, which uses HTTP as its main programming interface and JSON for data storage. Its key features are seamless replication, scalability and the absence of read locks by using Multi-Version Concurrency Control (MVCC). With regards to the *CAP theorem* [12], CouchDB focuses on Availability and Partition tolerance; it achieves consistency through a mechanism called *Eventual consistency* [5].

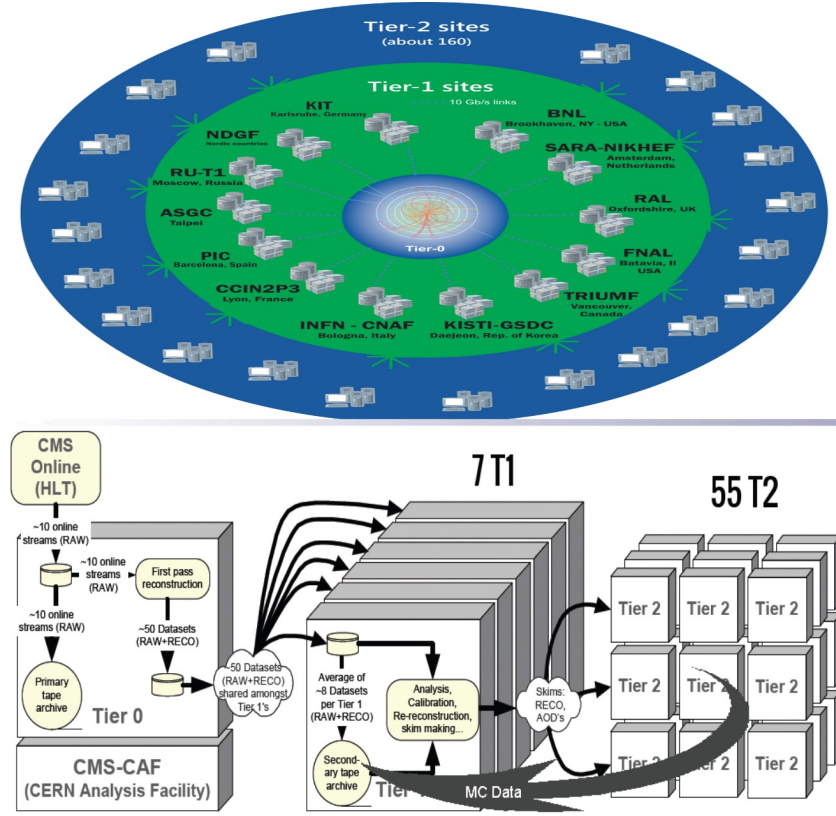


Figure 1: Overview of the three Tiers of the CMS Computing Model [11].

The *CMS Computing Model* can roughly be divided in three Tier Levels (see Figure 1):

- **Tier-0:** Accepts RAW data from the CMS Online Data & repacks, archives and distributes it to Tier-1 data centers
- **Tier-1:** Provides tape archive system, on top of Disk and CPU resources for data reprocessing, monte carlo production and data analysis. It's in charge of RAW data archival, as well as other event content types such as Analysis Object Data (AOD)² and their mini and nano variants
- **Tier-2:** Used for Grid-based analysis, Monte Carlo simulation and data reprocessing

A crucial part of the Tier-2 infrastructure is the *CMS Workload Management System* (WMS), which is driving all the Monte Carlo and data-processing related activities of the CMS experiment and is storing its results in Tier-1 data centers.

¹A NoSQL database provides a mechanism to store and retrieve data without enforcing tabulat relations, used in relational database like MySQL

²AOD is a format that contains all informations needed for a CMS analysis. It contains reconstructed 'physical objects' such as electrons, photons and muons [1].

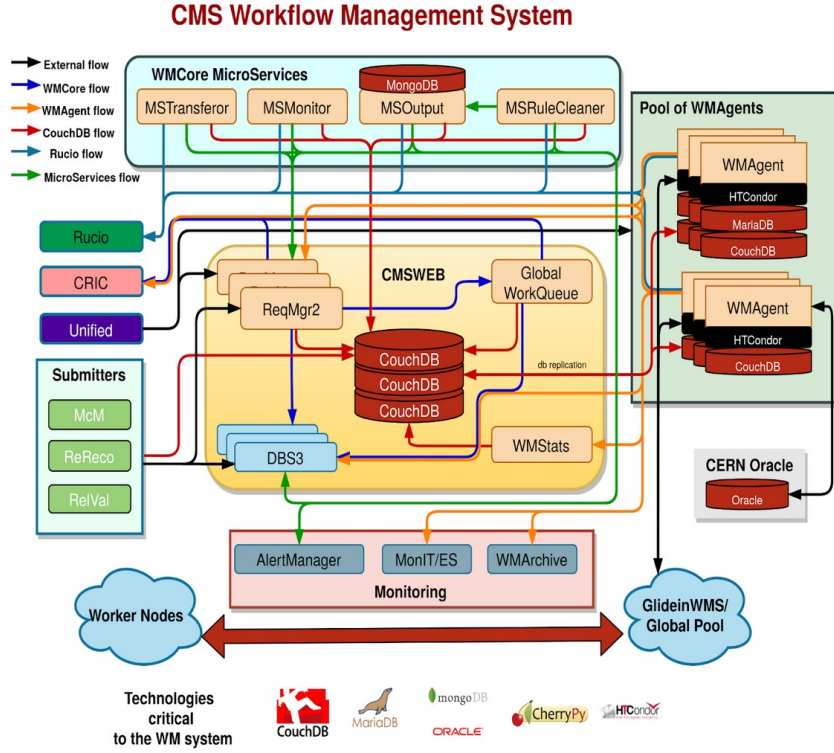


Figure 2: Schematic overview of the CMS Workload Management System using CouchDB at its core [11].

By looking at the schematic overview of WMS in Figure 2, the following components can be identified:

1.1 RequestManager2

RequestManager2 (*ReqMgr2*) is part of the CMSWEB central services cluster and provides the tools for the creation and management of requests via a Web GUI and a REST API. It furthermore manages the request life cycle (e.g. update status, priority) and stores the request data in *CouchDB*.

1.2 WorkQueue

WorkQueue is a multi-level queue, whose *WorkQueueElements* carry key information such as input data, statistics of the amount of work measured in several jobs as well as state and other data.

At its core, *WorkQueue* processes work (requests) from *ReqMgr2* and splits them into small pieces to make it available for the local queues to acquire and posteriorly inject it into WMS.

With regards to data storage, *CouchDB* also plays a key role in *WorkQueue*: Being the central data storage, *CouchDB* is being written to by *WorkQueue* to update locations and state of the *WorkQueueElements* periodically and hence serves as the 'memory' of *WorkQueue*.

1.3 WMStats

The *WMStats* component is responsible for keeping a snapshot of the current status of requests and collective jobs. It furthermore provides a Web GUI and REST API for finding requests and uses *CouchDB* as its main data storage.

1.4 WMAgent

The *WMAgent* software is a distributed component of the production system, which splits *WorkQueue* elements into smaller units of processing (jobs), is creating jobs and tracking its status.

Furthermore, *WMAgent* is responsible for data bookkeeping and is ensuring the proper enforcement of

task dependencies.

To put it in a nutshell, CouchDB plays a crucial role in WMS and therefore requires constant optimization and software update to keep it secure and fast.

2 Milestones

Because of the complexity of WMS and the various services interacting with CouchDB, an upgrade path with different milestones had to be defined.

The main goals are analyzing the existing infrastructure, migrating CouchDB data to the new data format (introduced in CouchDB 2.x) and outlining a procedure for the actual upgrade in the production environment.

In particular, the following milestones had been set:

2.1 Evaluate CouchDB features currently used by WMCore

To properly test CouchDB, local instances using cmsweb-testbed data had been deployed in *OpenStack* VMs.

After that, all *couchapps* of WMCore, located under <https://github.com/dmwm/WMCore/tree/master/src/couchapps>, had been deployed to both CouchDB 1.6.x and 3.1.x to investigate breaking changes.

WMCore serves as a meta-project providing multiple software for WMS, such as WMAgent, ReqMgr2, Global WorkQueue, WMStats and ReqMgr2 MicroServices.

It furthermore contains core libraries to other Data Management and Workload Management (DMWM) [7] CMS services, like CRABServer, DBS3, Tier0 and Unified [10].

Once set up, this deployment allows testing of all CouchDB queries like views, filters and update functions with dummy data.

2.2 Document all the important changes between CouchDB 1.6.x and 3.1.x

To complete this milestone, the CouchDB Upgrade Notes [6] had to be studied.

Consulting the relevant CouchDB documentation shed some light on the most important changes between CouchDB 1.x and 3.x.

Next to the deprecation of the Futon Web GUI in favor of Fauxton, support for native clustering and the removal of temporary views turned out to be the most crucial changes.

2.3 Evaluate the cross-version compatibility of CouchDB's replication protocol

Researching in the CouchDB documentation and GitHub Issue Tracker led to the conclusion that the replication protocol is interoperable between CouchDB 2 and 3. Still untested is the cross-version compatibility between CouchDB 1.x and 3.x.

2.4 Creating a CouchDB 2.x Docker Image

Because CouchDB changed its database format in 2.x, a Docker Image is needed in order to migrate the existing data. To fulfill this task, no WMS-specific features are needed. Hence the bitnami couchdb:2 Docker Image had been used.

2.5 Define a plan to migrate CouchDB databases from 1.x to 2.x

In order to use CouchDB 1.x data in 3.x, all databases had to be migrated to the new format, introduced in 2.x. To make this migration possible, CouchDB 2.x contains couchup, a python-based command-line database migration tool.

In order to run *couchup* in the CouchDB 2.x Docker Container, python3 had to be installed. To do this, a new Docker Container based on the *bitnami* Image had been created by the following *Dockerfile*:

```
// Dockerfile
FROM bitnami/couchdb:2
LABEL maintainer "Bitnami <containers@bitnami.com>"

## Install 'python3'
USER 0
RUN apt-get update
RUN apt-get install -y python3 python3-pip
RUN pip3 install requests
# Switch back to couchdb user
USER 1001
```

After that, the CouchDB 1.x data can be mounted into the container's data directory and be migrated by the *couchup* tool:

```
$ couchup list
$ couchup replicate -a
$ couchup rebuild -a
$ couchup delete -a
```

2.6 Get familiar with a Docker Image for CouchDB 3.1.x

Like before, the *bitnami* Docker Image serves as a sufficient base layer for further changes.

In particular, the following tests had to be performed with the Docker Container:

- Test database replication localhost and remotely
- Test the current authentication method based on X509 grid certificates
- Test and explore token-based authentication
- Perform some scalability/performance tests to evaluate whether a containerization model would meet the current demands (approx. 10 million requests a day, but to be increased in the coming years)

Because of changes in CouchDB and Erlang, the currently used custom authentication module, which enabled X509 grid certificate-based authentication for the CouchDB API and GUI, had to be updated as well (see subsection 2.8).

2.7 Create new SPEC files for CMSDIST

In order to deploy CouchDB 3 in CERN's CC7³, the SPEC files for building CouchDB and Erlang RPM images had to be updated. A SPEC file can be thought of as a 'recipe' that the *rpmbuild* utility uses to actually build an RPM, the baseline package format of Linux:

It tells the build system what to do by defining instructions in a series of sections [9].

The current state is stored in the CMSDIST Git repository [2], which serves as the CMS Offline Software build configuration and contains e.g. SPEC files for building RPM images to run on CC7.

2.8 Update existing Erlang patches for the CMS authentication layer

To authenticate requests against CouchDB's REST API and GUI, a custom authentication module had been compiled into CouchDB.

This custom module allows authentication with a X509 based grid certificate.

In order to use it with CouchDB 3.1.x, the module had to be updated, compiled into the CouchDB RPM and tested with valid grid certificates.

³CC7 is a custom image of CentOS 7 used by CERN

2.9 Create a development Docker Image with CouchDB 3.x

Based on the previously explored Docker Image (see subsection 2.6), a custom image to run WMCore had to be created, which mimics the features and applied patches of the custom RPM image (see subsection 2.7).

The main purpose of this development image is to run all WMCore unit tests⁴ in order to identify errors caused by the CouchDB 3.x migration.

2.10 Identify and apply relevant changes to WMCore

After the CouchDB RPM images are in place and the custom Docker Image for CouchDB 3 is working, breaking changes between CouchDB 1.x and 3.x have to be identified and fixed.

To achieve this, the development Docker Image (see subsection 2.9) is used to run all WMCore unit tests.

Because all major components of WMCore are covered by unit tests, this strategy gives a systematic approach to identify and fix bugs and other errors.

The steps necessary to run unit tests in the Docker container are documented in the WMCore wiki [11]. Because the documentation has currently not been updated to CouchDB 3.x, the container has to be started as follows

```
$ docker run -v ~/LOCAL_PATH_TO_GRID_CERT:/home/dmwm/certs \
-v ~/LOCAL_PATH_TO_WMCORE:/home/dmwm/wmcore_unittest/WMCore \
--name wmcorepy3_tests -it \
--entrypoint /bin/bash \
gitlab-registry.cern.ch/amaltaro/docker:wmcorepy3_tests
```

After that, the environment variables can be sourced, CouchDB and MariaDB started and unit tests using *python3* run:

```
$ source env_unittest_py3.sh
$ manage start-services
$ cd $TEST_DIR/WMCore/test/python/WMCore_t/WMSpec_t
$ python3 WMSpec_t.py
```

As soon as all unit tests have passed that also passed under CouchDB 1.6.x (because of an in-progress python3 migration some unit tests do not pass), all relevant changes have to be presented to the WMCore Git repository via Pull Requests, awaiting review and merging.

By the end of the Summer Student Internship, the development Docker Image has been deployed and issues in *Database*, *JobSplitting* and *JobStateMachine* regarding the replication protocol and the deprecated & removed temporary views have been identified.

Furthermore, a first Pull Request to remove the *all_or_nothing* flag has been created [8], reducing the amount of failed unit tests to 80 (around 30 have already failed before the migration) out of 1463.

2.11 Build, deploy and test final setup

At last, the final setup has to be deployed and tested. To achieve a smooth migration, extensive documentation has to be written, which serves as a step to step guide for the upgrade.

After deployment, all services have to be tested in order to avoid bugs not being unveiled by the unit tests.

With regards to subsection 2.3, the cross-version compatibility of replication between CouchDB 1.x and 3.x has to be tested as well, so that the migration of specific components of WMS is possible while running the remaining ones still on CouchDB 1.x.

⁴Unit Testing is a software testing method by which individual units of software code are tested to determine whether they are fit to use [13]

3 Conclusion

Unfortunately - due to the complexity and other imponderabilia of the CouchDB migration - the Summer Student Internship was over before the full migration could be finished.

As of now, the most crucial milestone subsection 2.10 is still in its early stages. Therefore future focus should be put on extensive testing and running of all unit tests in order to identify the necessary changes to *WMCore*.

On a personal level, experiences regarding building and deploying RPM images via SPEC files, as well as insights into the 'heart' of WMS have been gained.

4 Future Work

In addition to completing the remaining milestones, a potential migration to CouchDB 4 should also be kept in mind. This migration might be a big challenge because large parts of the current WMCore code will have to be updated [3]:

The following features are deprecated in CouchDB 3.0 and will be removed in CouchDB 4.0:

- Show functions (`/ {db} / {ddoc} / _show`)
- List functions (`/ {db} / {ddoc} / _list`)
- Update functions (`/ {db} / {ddoc} / _update`)
- Virtual hosts and ini-file rewrites
- Rewrite functions (`/ {db} / {ddoc} / _rewrite`)

Finally, newly introduced features like native clustering, as well as features and enhancements like CouchDB's support for *ECMAScript5* & *ECMAScript6* (thanks to *SpiderMonkey 60*) [4] should be explored and utilized at their best.

Acknowledgements Most of the information about WMS and its components, especially the ones mentioned in section 1, have been taken from [11]. I would like to thank Alan and Todor for the excellent supervision and the interesting topic. Even though the Summer School was completely online, I always felt well-integrated and had a lot of fun. At last, I am also very grateful for working at CERN!

References

- [1] AOD: Analysis Object Data. <http://opendata.cern.ch/glossary/AOD>. Accessed: 20/08/2021.
- [2] CMS Offline Software build configuration . <https://github.com/cms-sw/cmsdist>.
- [3] CouchDB 3: Deprecated feature warnings. <https://docs.couchdb.org/en/main/whatsnew/3.0.html#deprecated-feature-warnings>. Accessed: 30/08/2021.
- [4] CouchDB 3.x: Upgrade Notes. <https://docs.couchdb.org/en/stable/whatsnew/3.0.html#upgrade-notes>. Accessed: 30/08/2021.
- [5] CouchDB: Eventual Consistency. <https://guide.couchdb.org/draft/consistency.html>. Accessed: 30/08/2021.
- [6] CouchDB: Upgrading from prior CouchDB releases. <https://docs.couchdb.org/en/3.1.1/install/upgrading.html>. Accessed: 30/08/2021.
- [7] Data Management and Workload Management Group Source Code. <https://github.com/dmwm/>.
- [8] Pull Request - CouchDB 3 Migration: Remove AllOrNothing flag. <https://github.com/dmwm/WMCore/pull/10780>.
- [9] RPM Packaging Guide. <https://rpm-packaging-guide.github.io/>. Accessed: 29/08/2021.
- [10] CERN WMCore Team. WMCore Git Repository. <https://github.com/dmwm/WMCore/>.
- [11] CERN WMCore Team. Wmcore wiki on github. <https://github.com/dmwm/WMCore/wiki>.
- [12] S. Gilbert and N. Lynch. Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002.
- [13] D. Kolawa, Adam; Huizinga. *Automated Defect Prevention: Best Practices in Software Management*. Wiley-IEEE Computer Society Press, New Jersey, 2007.