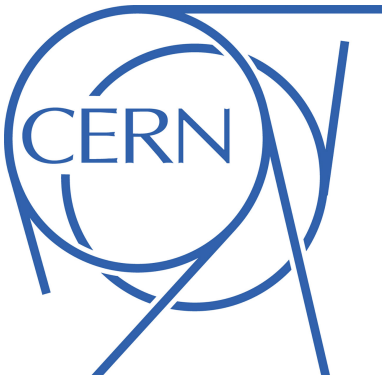
Oracle database integration into Post Mortem system



Martin Zak

Supervisors: Tiago Martins Ribeiro, Anita Stanisz

CERN Summer Student Internship report
summarizing all the work done

in the

Machine Protection and Electrical Integrity Group

August 9, 2019

Abstract

Post Mortem system is proper designed service responsible for ingestion of data buffers from LHC accelerator equipment systems due to the specific events. Bunches of collected data are stored in the system for future access, allowing for analysis to occur, which is performed by Post Mortem Analysis Framework. As the NxCALS is used as a long-term storage for data, its plain directory structure for data storage causes poor performance of client requests and makes it difficult to achieve distribution and redundancy. System doesn't meet the requirements for data availability. It takes too much time for data to be accessible for reading in NxCALS. To meet the requirements, it was decided to use database as another storage solution in parallel to NxCALS.It should increase the performance. This report contains all steps needed to integrate database into Post Mortem system.

Contents

Chapter 1

Introduction

The Large Hadron Collider (LHC) is the the largest particle accelerator in the world. Its aim is to accelerate two particle beams to the speed of light, boosting their energy along the way until they eventually collide. These beams don't circulate in LHC forever. They are redirected out of the accelerator in 2 cases. Either after a few days they loose intensity and don't provide expected luminosity hence need to be replaced by new ones or an unexpected problem occurred. In both cases sensors in the LHC send data to be processed, analyzed and to inform staff members about the current state of LHC equipment (e.g. magnets). The phase at which data is processed and analyzed, is where proper designed service Post Mortem takes place [3].

1.1 Post Mortem Service

Post Mortem service is responsible for ingestion of data buffers from accelerator equipment systems due to specific events such as beam losses (old beams replaced by new beams) or magnet quenches. The system handles high resolution data buffers, which are complementary to low resolution logging data. The collected data is stored in the system for future access, allowing for analysis to occur, which is performed by the Post Mortem Analysis Framework. Furthermore, the results of the analysis are used for fast diagnostics identifying causes of beam dumps, powering failures, and different anomalies [1].
Although the system currently works, it has a lot of drawbacks in scaling. Nowadays, there is a long shutdown that gives an opportunity to reduce the number of issues. I was involved in a task, which aim was to improve performance of handling data. For better understanding, a more detailed description of the problem and potential solution is discussed in the next subsection.

1.1.1 Problem description and potential solution

NxCALS is a system which gives a possibility for storing data into long-term storage. But the use of plain directory structure for data storage causes poor performance and makes it difficult to achieve distribution and redundancy. Also with NxCALS it wouldn't be possible to meet the required time constraints. One of the potential solutions would be to use a database. Firstly, the database would be used as a second storage where data would be

duplicated in parallel to NxcALS in the last X hours (X will be defined by client). Then if the client requests for data within a specific time frame, a query would be made to the database (not to the NxCALS) and so the query performance should be increased.

Chapter 2

Oracle database integration

As a second storage an Oracle database was used. The reason for choosing was that CERN IT provides long-term support for oracle services this was that having a relational database was one of the main requirements - oracle database met the condition - and CERN provides long-term support for oracle services.

2.1 Integration

Database integration itself was done by managing data to be stored not only NxCALS, but to the database as well. Also, one of my tasks was to provide a library for retrieving data from database. It should be noted that there were 4 types of data which need to be stored - PmRawData, PmSession, PmClassifiedEvent, PmAnalysisResult. This meant that 4 tables need to be created in the database. In my tasks I incorporated only PmRaw data as it had been expected of me. Figuring out the implementation of PmRaw data meant that the same implementation could be reused for the other types of data with little additional changes being required. These two main tasks are described in more detail in the following sections.

2.1.1 Store data

At first, I created a service which handled data storage. This service took care of whether the data should have been stored only to one storage or both of them. As incoming data was not in the proper format, before sending it to the database, it had to be converted to the correct type that the database accepted. Therefore I created *PmRawEntity* and properly converted the data. Only after conversion to *PmRawEntity* an object relational mapper *Hibernate* which I used, could have easily ensured storing of data.

2.1.2 Retrieve data

The main aim was to implement exposed interface by which the client could retrieve the data. The interface and one implementation had already existed before because the client could have made requests for data to the NxCALS. Therefore I just did another implementation for retrieving data from the Oracle database. But it was a little bit tricky because the client expected data in *PmRawData* format and *Hibernate* could have dealt only with *PmRawEntity* data. Hence I had to implement the conversion from *PmRawEntity* to *PmRawData*.

2.2 Testing

In general, software testing is one of the most important activities during development to ensure the software is defect free. It especially matters when the software gets more complex. This was the case for Post Mortem. Due to its codebase being covered only by unit tests and integration tests, I also had to write my own unit and integration tests for the work I carried out, keeping to the convention.

2.2.1 Unit tests

In general, these tests should test isolated parts of the code (methods) and demonstrate that each individual sections is correct, meeting its design and behaves as intended. When some dependency is passed as parameter to some method, it is mocked. I wrote and ran unit tests every time I added any new code just to be make sure nothing had broke.

2.2.2 Integration tests

These tests usually occurs afterwards all unit tests are written. The reason is that software modules, which proper functioning was ensured by unit tests, can be combined and tested as a group. Unlike unit tests all dependencies are used which means that real scenarios are tested. In my case, I wrote many integration tests. They usually tested all possible edge cases which could happen during sending and retrieving data.

2.3 Technologies used

Technologies I needed to make complete my work:

- Java version 8 + Spring
- Hibernate 5
- Oracle database 19

Chapter 3

Software management

Each software development team have to manage their work and deliver the product to the customer. First, it is good to have one central location with information about all the issues needed to be fixed, with information on who is responsible for which task, as well as the status of each task i.e. in progress. Also, the best practice is to version the system. In the case of unexpected failure or to see previously fixed defects code, it is possible to return back or just check the previous state of code. Furthermore, the CI/CD is a set of processes allowing the application development teams to deliver code changes more efficiently, frequently and reliably. Also automatic inspection of code quality helps to avoid code smells. More detailed description about tools ensuring all of the above tasks is in the following sections.

3.0.1 Issue tracking system

As I mentioned before, issue tracking system helps teams to manage their work. In my team, we were using JIRA that allows issue tracking and agile project management (explained in more detail in 4th chapter). Thanks to JIRA everybody was able to see the state of all the tasks. They could have been in several states - *Select for development*, *Breakdown*, *Review*, *Validate*, *Demo* and *Done*.

3.0.2 Version control system

For versioning the Post Mortem System and keeping track of incrementally different versions of code, our team used *Gitlab*. One of its advantages is that JIRA issues can be easily linked to *Gitlab* branches.
When I wanted to work on some task, I had to create my own branch from *Develop* branch. When I successfully finished my task I could merge my branch into *Develop*. If something went wrong unexpectedly, hotfix branch was created and after fixing of the problem, it was merged into *Develop* as well. When everything was right, *Develop* branch was merged into *Master* branch automatically (more explained in next subsection about CI/CD).

3.0.3 CI/CD

Bamboo is continuous integration (CI) and continuous development (CD) server developed also by Atlassian, which was used by my team. It took care of building the software which

consists of compiling and testing. As I mentioned in previous section, when everything worked in *Develop* branch, Bamboo automatically merged this branch into *Master*.

3.0.4 Code quality inspection

My team used SonarQube, which is a tool for automatic inspection of codebase. Inspection was executed every time the build ocured. When some code smell was detected, sonar task on Bamboo failed. It had to be fixed to make the build successful.

Chapter 4

Agile project management

Except of all technical needs and software management, project management is needed for planning and controlling the team to help them manage their work, achieve specific goals and get stuff done till deadline [2]. There are several approaches of project management. Everything is up to team and their needs. My team used Kanban which is type of agile project development. More details about our project management are in the following sections.

4.0.1 Kanban board

Our tasks which everybody was working on were defined in Kanban board. We had two Kanban boards. One was virutal board within the JIRA software tool. As I mentioned in the section about issue tracking system, tasks could have been in many states. Second one was real, physical board with printed tasks and with swim lanes representing various states for tasks. Primarily, this board was used during daily stand-ups (more in the following section). This boards contained a little bit more states than virtual one - *Backlog*, *Breakdown (doing)*, *Breakdown (done)*, *Do (doing)*, *Do (done)*, *Review*, *Validate (doing)* and *Validate (demo)*.

4.0.2 Daily stand-ups

Every day we had daily meetings called stand-ups at 11 a.m. During this time, scrum master checked all the tasks being in a progress and asked responsible people about the state, how it goes and whether they have any difficulties that could be discussed.

4.0.3 Code reviews

Code review is an activity that, which although slows down the development but assure quality of code. Even though only one or two persons implement proper solution for the task, many people know how does it work just because of code reviews. It usually helps to make the code quality and functionality better, improves the readability of code as there are many people who need to understand it, helps to identify potential refactorings and last but not least ensures knowledge sharing. In my team, there were 2 types of reviews: pre-live reviews and live reviews. The aim of pre-live review was to get introduce reviewers to the problem and my implemented solution. Usually, it took a short minutes because I went through whole added code really fast and explained general idea. Afterwards, reviewers

reviewed code properly and put some comments because of uncertainties, code smells or potential refactorings. When I acknowledged all comments, the live review followed. The objective of live review was demonstrate, how I dealt with problems mentioned in comments. If everything was right, my task was done. Otherwise I some time to resolve additional comments.

Chapter 5

Conclusion

I think I learned a lot. I had an honor of having a really good supervisors ready to help me every time I had some problems. They were really patient and willing to answer all my questions I had. I learned a lot about unit testing, integration testing, software design and some design patterns. I think this was the best summer I have ever had. I found good balance between working and enjoying my time here in CERN.

Bibliography

[1] M. Powierz. Ingestion layer of the next generation CERNs Post Mortem service, 2018.

[2] Wikipedia. Project management, 2019.

[3] K. Yurkewicz. Protecting the LHC from itself, 2007.