Integration and validation testing for PhEDEx, DBS and DAS with the PhEDEx LifeCycle agent

C Boeser[1], T Chwalek[1], M Giffels[2], V Kuznetsov[3] and T Wildish[4]

[1] Institut für Experimentelle Kernphysik, Karlsruhe, Germany
[2] PH-CMG-CO, CERN, CH-1211 Genève 23, Switzerland
[3] Cornell University, Ithaca, NY, USA
[4] Princeton University, Princeton, NJ, USA

E-mail: awildish@princeton.edu

Abstract. The ever-increasing amount of data handled by the CMS dataflow and workflow management tools poses new challenges for cross-validation among different systems within CMS experiment at LHC. To approach this problem we developed an integration test suite based on the LifeCycle agent, a tool originally conceived for stress-testing new releases of PhEDEx, the CMS data-placement tool. The LifeCycle agent provides a framework for customising the test workflow in arbitrary ways, and can scale to levels of activity well beyond those seen in normal running. This means we can run realistic performance tests at scales not likely to be seen by the experiment for some years, or with custom topologies to examine particular situations that may cause concern some time in the future.

The LifeCycle agent has recently been enhanced to become a general purpose integration and validation testing tool for major CMS services (PhEDEx, DBS, DAS). It allows cross-system integration tests of all three components to be performed in controlled environments, without interfering with production services.

In this paper we discuss the design and implementation of the LifeCycle agent. We describe how it is used for small-scale debugging and validation tests, and how we extend that to large-scale tests of whole groups of sub-systems. We show how the LifeCycle agent can emulate the action of operators, physicists, or software agents external to the system under test, and how it can be scaled to large and complex systems.

1. Introduction

PhEDEx [1], DBS [2] and DAS [3] are the core components of the CMS Data Management system [4]. PhEDEx is responsible for moving data around the CMS computing sites, DBS is responsible for maintaining knowledge of the physics content of the data, and DAS is a portal that provides physicists with integrated access to PhEDEx, DBS, and other services.

The development model for these services is highly decoupled. Each is developed by a separate team of people, and each provides a web-based data service for access to information. Ensuring stability and consistency of these interfaces has, until recently, been loosely managed. We have relied on stability of the underlying services and coordination between developers to ensure that successive releases of the software are coherent and correct.

In this paper we describe part of the solution to this problem. The *LifeCycle agent*, written originally for scale-testing of PhEDEx, has been extended to provide cross-component integrated testing in a flexible and powerful manner. Components can be tested individually or together

in different ways for verification, debugging, or scale and performance testing. The LifeCycle agent itself scales well, and is easily adapted to new requirements (e.g. testing of new APIs from the services). It can inject data into the component systems, retrieve it, manipulate it, and compare it with expectations to detect errors. It can also test expected error conditions for correct behaviour (e.g. access with an unauthorised certificate).

2. The LifeCycle agent

2.1. Architecture

The LifeCycle agent emulates the actions of users or external processes in a system. The key abstraction is the *workflow*, which consists of a sequence of *events* that depend on each other and are performed at distinct intervals. A sample workflow for PhEDEx could consist of the following events:

- Generate data - a dataset is generated with a set of files.
- The files are 'injected' (registered) into PhEDEx at a given site.
- The datasets are subscribed for transfer to one or more other sites.
- After a time, the data may be deleted from one or more of the sites.

By varying the details of how the generated data is organised, which sites it is sent to, and which sites (if any) it is finally deleted from, we can emulate different real-life situations.

Many workflows can run in parallel. We can emulate the production of dozens of datasets, each destined for different sites, each with different characteristics, to give us a realistic emulation of the total workload of the production transfer system.

2.2. Event handlers

Each workflow has an array of *events*. This is an array of strings which the agent maps to their handlers. Handlers can be Perl modules loaded at runtime, or external scripts written in any language.

To use an external script as a handler, it must accept two filename arguments, one for input and one for output. The LifeCycle agent packages the workflow as a JSON object, writes it to the input file, then calls the script. The script reads the file and reconstitutes the object in it's own representation (e.g. a python dictionary), then accesses the elements it needs to do its work. When the script finishes it re-packages the workflow and writes it to the output file, which the LifeCycle agent will read and use to update its internal representation of the workflow.

Workflows are processed in parallel. As soon as one event-handler for a workflow finishes, the next event for that workflow is queued, for execution after a configurable delay. External script handlers are run asynchronously, handlers written as Perl modules can be either synchronous or asynchronous. A workflow completes when its last event has been processed. After that, depending on the configuration for that workflow, it can be restarted again at a later time.

Mechanisms are provided for event handlers to signal errors to abort the workflow or the entire LifeCycle agent, or to provide statistics that are accumulated by the agent and reported periodically.

Workflows need not be fixed. The event handler has access to the entire workflow object and can manipulate the event chain in many ways. It can re-insert the current event, to block the workflow, polling for an external condition to be satisfied. It can append events to the event chain, to steer the workflow according to the conditions it has encountered so far. Handlers can even spawn new workflows, by cloning and modifying their own workflow.

3. PhEDEx

3.1. Scale tests of PhEDEx

The LifeCycle agent was first written to perform scale tests of PhEDEx. We use a fake transfer layer consisting of a script that mimics FTS responses, using the input file-sizes and a timestamp in a cache to allow it to respond according to realistic expectations for throughput.

We use a topology that resembles the full CMS system (number of Tier-0/1/2 nodes, links between them), we configure data-flows that resemble the real activities of the experiment, and we run the full PhEDEx system on a number of machines (virtual or batch).

This lets us emulate realistic CMS conditions without needing network, disk or tape resources, so we can run at 'data rates' far higher than expected in the near future. This has shown that PhEDEx will not hit any scaling problems in the next year or two if circumstances remain the same. If we did find a scaling problem, we will have time to address it before it happens in the production system.

3.2. Testing new PhEDEx releases

The LifeCycle agent is used to test PhEDEx releases. A private PhEDEx instance is set up in a testbed with a few nodes, and fake transfers between these nodes are driven using the LifeCycle agent to orchestrate the flow. In this way bug fixes and new features of new PhEDEx versions can be tested before releasing the version.

For special cases, such as bugs that occur under unusual consitions, the LifeCycle agent can be configured to reproduce these conditions, repeatedly and systematically.

3.3. Integration test of the web-based PhEDEx DataService

One of the harder things to test is the website access by different users with different access-rights. Rather than requiring the support of several people with different credentials to systematically test new releases of the website, with all the coordination and overhead that implies, we use the LifeCycle agent again.

We extend the private testbed to include a website, with self-signed certificates allocated to specific roles. The LifeCycle agent uses the website to perform a series of actions (create request, approve it, change its priority), repeating this workflow for each certificate, and for the different source and destination nodes. It checks that actions succeed where they should and fail where appropriate. In this way, several combinations of actions (inject, subscribe,...), destination nodes (T0, T1,...), and roles (site-admin, data-manager,...) can be tested in an automated way by configuring and running one single script.

4. DBS

4.1. Data Bookkeeping Service

The Data Bookkeeping Service (DBS) [2] provides a catalog of event metadata for Monte Carlo and recorded data of the CMS experiment. DBS contains records of what data exists, their process-oriented provenance information, including parentage relationships of files and datasets and their configurations of processing steps. It also contains associations with run numbers and luminosity sections to allow lookup of any subset of events within a dataset, on a large scale of about $200,000$ datasets and more than 40 million files, which adds up in around 700 GB of metadata. The DBS is an essential part of the CMS Data Management and Workload Management (DMWM) systems [4]. All kind of data-processing - Monte Carlo production, processing of recorded event data as well as physics analysis done by the users - relies heavily on the information stored in DBS.

4.2. Scale testing of DBS 3

The LifeCycle agent is perfectly suited for the scale testing of DBS 3, which is a crucial step to ensure a reliable system even in conditions of high load. DBS 3 is written in Python, therefore a Python framework was developed to simplify the interaction with the LifeCycle agent. The framework facilitates the handling of JSON payloads, error handling, building HTTP API calls from payloads and provides timing utilities. The collected data can be sent back to the LifeCycle agent or exported to a SQLite database for further analysis. For the analysis of these data an additional package, *LifeCycleAnalysis* was developed. It contains a ROOT histogram manager automatically creating various timing and error distribution histograms for all the APIs used in the scale test. This framework is generic, it can be easily used by any python-based tool to interact with the LifeCycle agent. The actual scale tests were performed by using several LifeCycle agents and workflows submitted to a batch system, which allows to run an arbitrary number of workflows against the server being tested. Therefore, additional scripts are part of the framework to simplify the interaction with the LSF batch system at CERN.

5. DAS

5.1. Data Aggregation Service

The Data Aggregation Service (DAS) [3] is a critical component of the CMS Data Management System. It provides the ability to query CMS data-services via a uniform query language without worrying about security policies and differences in underlying data representations. When a physicist poses a question like *"Find me files from RelVal dataset at certain site"*, DAS converts the question into a series of requests to the underlying data-services. DAS takes care of aligning naming conventions, data conversion into common JSON format and aggregating data on a requested entity key, e.g. file names. As a result, the user gets back an aggregated record across participating data-providers. This workflow fits really well into LifeCycle design which utilize DAS as cross-check validation tool for its components.

6. Integration testing of PhEDEx, DBS, and DAS

The LifeCycle agent was used for cross-integration tests of all three components (PhEDEx, DBS 3, and DAS) within a controlled environment and thus without interfering with production services. Fake data was created and injected into PhEDEx and DBS 3. DAS then retrieved information about the data from both sources, and compared the results. For an illustration of the workflow see Fig. 1. By injecting specific errors in either PhEDEx or DBS 3 (changing filenames etc) we can fake errors that we expect to detect with DAS. Special event-handlers are used to compare the errors detected by DAS with the injected errors, and alert us to any unexpected failures.

In the current implementation six kinds of failure can be faked and the probability to occur can be adjusted for each single failure type individually.

- Single file is skipped in the PhEDEx data
- Single file is skipped in the DBS 3 data
- The checksum for a single file is changed for the PhEDEx data
- The checksum for a single file is changed for the DBS 3 data
- The size for a single file is changed for the PhEDEx data
- The size for a single file is changed for the DBS 3 data

The payload-provider (data provider) generates fake data, which then can be injected to a node of the LifeCycle testbed. It also generates extensions to filenames specifying the failure type the file is affected by (according to the probability that was specified).

Special modules read the filenames and act on all files with a certain extension specifying the kind of failure the file is affected by. This includes whether the PhEDEx information, the DBS 3 information, or both are affected and the kind of failure (different checksum, different size or skipped file). Also combinations of these failures are possible for one single file. According to the kind of failure these modules either remove the files, change their size or their checksum, or perform a combination of the mentioned actions, when injecting the information to PhEDEx and DBS 3.

After passing these error-handler modules, all files are injected into PhEDEx and DBS 3. A DAS-query is performed retrieving the information on the injected data from PhEDEx and from DBS 3. These two sets of information on the injected data are compared and the differences found are checked against the true information generated by the workflow. In this way we can check that DAS properly reports all occuring mismatches in the information on a certain datset coming both from PhEDEx and/or DBS 3.

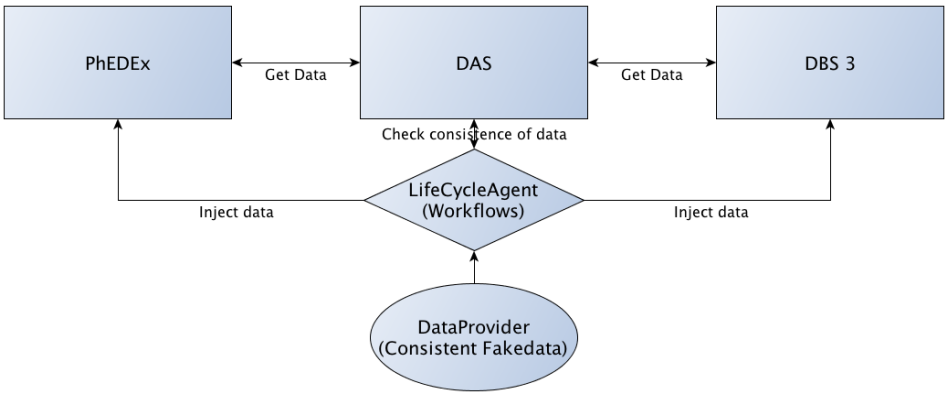

Figure 1. Workflow of the integration testing of PhEDEx, DBS 3, and DAS.

7. Conclusions

The LifeCycle agent provides a flexible and powerful framework for debugging, integration testing, stress testing and scale testing. It scales well from testing a single component in isolation to emulating the dataflow of the entire CMS experiment over long timescales.

It can also be used as a generic workflow engine, steering the actions of external components in arbitrary and complex ways.

It interfaces easily to external scripts, making it usable in a number of domains, within and outside CMS.

References

[1] Egeland R, Wildish T and Metson S 2008 Data transfer infrastructure for CMS data taking, *XII Advanced Computing and Analysis Techniques in Physics Research (Erice, Italy: Proceedings of Science)*
[2] Giffels M, Data Bookkeeping Service 3 - Providing event metadata in CMS, *submitted to CHEP 2013*
[3] Kuznetsov V, Evans D and Metson S, The CMS Data Aggregation System, *doi:10.1016/j.procs.2010.04.172*
[4] Giffels M, Guo Y, Kuznetsov V, Magini N and Wildish T, The CMS Data Management System *submitted to CHEP 2013*
[5] Campana S et.al. 2013 WLCG and IPv6 - the HEPiX IPv6 working group *submitted to CHEP 2013*