EPJ Web of Conferences **251**, 02034 (2021) https://doi.org/10.1051/epjconf/202125102034
CHEP 2021

Preparing for HL-LHC: Increasing the LHCb software publication rate to CVMFS by an order of magnitude

Enrico Bocchi[1*], *Jakob* Blomer[1], *Benjamin* Couturier[1], *Christopher* Burr[1], and *Dan* van der Ster[1]

[1]CERN, Esplanade des Particules 1, 1211 Geneva 23, Switzerland

Abstract. In the HEP community, software plays a central role in the operation of experiments' facilities and for reconstruction jobs, with CVMFS being the service enabling the distribution of software at scale. In view of High Luminosity LHC, CVMFS developers investigated how to improve the publication workflow to support the most demanding use cases. This paper reports about recent CVMFS developments and infrastructural updates that enable faster publication into existing repositories. A new CVMFS component, the *CVMFS Gateway*, allows for concurrent transactions and the use of multiple publishers, increasing the overall publication rate on a single repository. Also, the repository data has been migrated to Ceph-based S3 object storage, which brings a relevant performance enhancement over the previously-used Cinder volumes. We demonstrate how recent improvements allow for faster publication of software releases in CVMFS repositories by focusing on the LHCb nightly builds use case, which is currently by far the most demanding one for the CVMFS infrastructure at CERN. The publication of nightly builds is characterized by a high churn rate, needs regular garbage collection, and requires the ability to ingest a huge amount of software files over a limited period of time.

1 Introduction

High Luminosity LHC (HL-LHC) sets ambitious goals in the particle physics road-map and demands substantial increases in computing requirements. Software is nowadays a fundamental component for physics research and lays at the base of the operation of experimental installations, events filtering during data acquisition, and reconstruction jobs for analysis.

The HEP community also needs to distribute software efficiently to take advantage of the massive computing capacity made available through the WLCG. In this context, the CernVM File System (CVMFS) is the service for the distribution of production software, integration builds, and auxiliary datasets at a global scale. In view of HL-LHC, we investigate how to improve the publication workflow (i.e., the process that allows publishing software into CVMFS repositories) to support the most demanding use cases.

This paper takes the LHCb nightly builds repository (i.e., `lhcbdev.cern.ch`) as a primary example with challenging requirements (listed in Sec. 2), reports on recent CVMFS additions for faster publication (Sec. 3), and shows the improvements brought in by S3 object

*e-mail: enrico.bocchi@cern.ch

storage (Sec. 4). Sec. 5 describes the production setup for LHCb-managed CVMFS repositories explaining how installations into CVMFS are managed, repositories are migrated to more performant S3 storage, and distribution of software to clients is more resilient.

2 LHCb Nightly Builds Repository Requirements

The LHCb experiment is one of the four large experiments at the Large Hadron Collider (LHC) and is currently being upgraded in preparation for Run 3 of the LHC. As part of this work, the detector's readout electronics and trigger system are being replaced to facilitate a fully software-based trigger system that is capable of reconstructing LHC collisions at a rate of 40 MHz. Upon completion, this will be one of the most advanced data acquisition systems in High Energy Physics with 40 Terabits of data being filtered each second. The LHCb software stack is based on the Gaudi framework which comprises around 10 main packages and 6 M lines of C++ code. Most packages contain historical versions, used for processing legacy datasets, that are still actively developed.

Nightly builds are a crucial component for maintaining this ever-expanding collection of software and most builds are installed on a dedicated CVMFS area (`/cvmfs/lhcbdev.cern.ch`) to assist developers. This allows for development against partial builds of the stack and also enables larger tests to be made using the WLCG via LHCbDIRAC. This, however, poses a significant challenge for the CVMFS infrastructure as approximately 3.8 M files (200 GB) must be installed each day, preferably with minimal latency. Furthermore, data quickly becomes obsolete causing nightly garbage collection to be essential to avoid the CVMFS repository growing to an unmanageable size. Historically, the installation of builds has been managed by a conventional CVMFS set-up which has long been plagued by performance issues, with builds being queued for up to a day and up to half of builds to never be installed at all. Additionally, the heavy load from nightly installations often delayed other types of deployment jobs, such as development releases of the LHCbDIRAC middleware.

3 CVMFS

Data in CVMFS is stored in a content-addressed form. When writing new and updated files to the file system, these files are compressed and hashed in a processing step called "publishing" [1]. Publishing content traditionally takes place on a dedicated machine available to the experiment software librarians. Once published, data is replicated and caches in the CVMFS content distribution network. This approach results in very high scalability for reading but the dedicated publisher node is a natural bottleneck for writing.

3.1 CVMFS Gateway

The CVMFS gateway addresses, up to a certain point, the scalability issues of a dedicated publisher node. [2] The gateway provides a REST API to the authoritative storage to which multiple publisher nodes can connect concurrently. These remote publishers can install and process new and updated files concurrently as long as each publisher operates on a different directory subtree of the repository. While access to the storage system (i.e., S3) is still serialized in the gateway, the computationally expensive part of publishing can take place in parallel by multiple nodes provided that the repository is structured carefully (e.g., such that software of different hardware architectures is hosted in different subtrees). At least up to a handful of remote publishers, we previously observed excellent weak scaling [3].

3.2 Garbage Collection

Due to the content-addressed format of data in CVMFS, publishing content never removes data from the internal storage representation. In this respect, CVMFS works similarly to a version control system. If the repository maintainer deletes files, those files are simply hidden from the file system representation. While this can be a desired property for repositories with stable production software, repositories hosting nightly builds quickly rotate their content and thus produce a large and quickly growing volume of dark content. The CVMFS garbage collection (GC) identifies and removes such dark content from the internal representation.

The CVMFS GC takes a "stop the world, mark and sweep" approach. It blocks any other publishing operation from taking place while running. The performance of the garbage collection is therefore critical for the overall throughput of a repository. During the mark phase, the GC traverses the entire directory tree and its history to identify dark content. The mark phase involves reading of the order of tens of gigabytes of meta-data from the repository to create the complete directory view. In recent developments, both creating the complete directory view and deleting objects in the sweep phase were parallelized to make use of many cores and parallel network connections, resulting in a speed-up of several factors. For the LHCb nightly repository, GC now runs once per day for 60–90 minutes, removing several hundred thousand files per run.

4 S3 Storage

4.1 Ceph-based Object Storage at CERN

CERN operates a multi-petabyte object storage cluster compatible with the Amazon Simple Storage Service (S3) protocol [4]. The service is backed by Ceph [5] and makes use of its Object Gateway daemon (radosgw) to provide object storage access. The cluster has a total raw capacity of 5.8 PB and is configured with a 4+2 erasure coding profile for data blobs, while metadata stored in the form of bucket indexes are replicated 3 times.

S3 offers several advantages compared to traditional filesystem storage for CVMFS:

- The data of each CVMFS repository is stored in a unique S3 bucket that can efficiently scale to store millions of objects.
- Both capacity and performance can be scaled horizontally by adding backend Ceph servers or frontend S3/HTTP gateways as needed.
- Repository quota can be adjusted without downtime using the radosgw admin tooling.
- High availability and data durability are guaranteed by the Ceph RADOS backend.
- Avoids the need for highly available HTTP servers combined with POSIX filesystems.
- Traffic to Ceph backends can be routed according to arbitrary rules using round-robin DNS and a cluster of HTTP frontends.

4.2 S3 Load-balancing and Dedicated Frontends

Traffic to and from as S3 uses a single DNS alias, `s3.cern.ch`, and is subject to load-balancing at different layers. First, network-layer round-robin DNS allows spreading the load across all the IP addresses which are part of the DNS alias (we typically use 10 machines with both IPv4 and IPv6 addresses). Second, application-layer load balancing is provided by Traefik [6], a modern reverse proxy, which forwards requests to the Ceph radosgw daemons. Fig. 1 shows the block diagram for S3 load-balanced access at CERN.

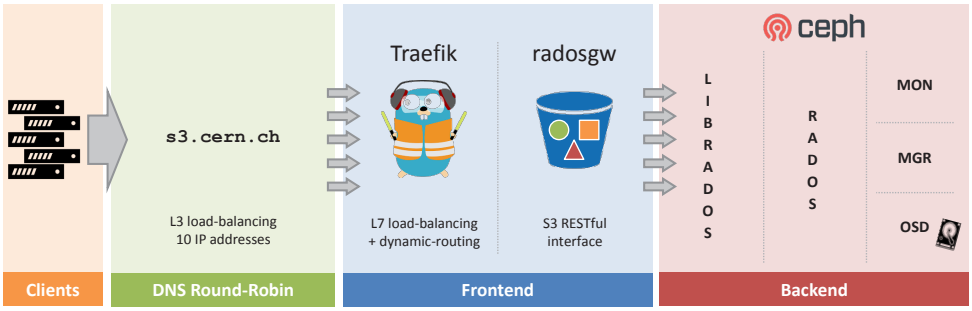

Figure 1. Block diagram of the access path to the S3 service at CERN. Clients target `s3.cern.ch`, a load-balanced DNS alias addressing the frontend machines. Each frontend runs Traefik (a reverse proxy for application-level load-balancing and traffic routing) and the radosgw daemon (the S3 RESTful interface to Ceph). Radogws reach the backend through the librados library and native Ceph protocols.

Traefik implements advanced features including HTTP requests rewriting, TLS termination, service auto-discovery, and health-checks. A relevant feature for load-balancing is the application-level dynamic routing of requests: Traefik supports the definition of rules according to which HTTP requests can be routed to different backend services.

We use this feature to route CVMFS requests to a dedicated set of radosgws (4 concurrently active at the time of writing). With these dedicated backend gateways, we are able to protect and isolate CVMFS from other S3 users, thereby providing consistent performance in terms of input/output operations. Indeed, while the average number of IOPS is moderate (we observe rates of ~300 Hz), CVMFS activity can be very spiky with peaks exceeding 4 kHz request rate. Rules to implement CVMFS-specific routing are based on matching a regular expression either on the URL path (e.g., `s3.cern.ch/cvmfs-lhcbdev/myobj`) or on the virtual hostname (e.g., `cvmfs-lhcbdev.s3.cern.ch//myobj`) to support both path-based and virtual-host-based bucket addressing in S3.

4.3 S3 support in CVMFS and Performance Improvements

CVMFS supports S3 storage since version 2.1.20 (2015). New capabilities have been implemented to manage transfers against S3 (including an error concealment mechanism based on timeouts and retries), configure ACLs of uploaded objects (CVMFS repositories are world-readable), and use parallel connections for upload/download. This latter feature has positive performance implications and a direct impact on the time needed to finalize a publication.

Fig. 2 measures the overall time needed by a CVMFS server to publish a sample transaction made of 250 k files with 4 kB size organized in 250 folders (each with a dedicated `.cvmfscatalog`) on S3 storage and a traditional ZFS filesystem, previously used as the storage backend for CVMFS repositories. Results for S3 clearly show the speedup obtained with parallel connections: While with a single connection (i.e., no concurrent uploads to S3) the publication takes more than 20 minutes to complete, the use of 8 parallel connections already outperforms the ZFS filesystem. CVMFS publishers are configured to use 64 parallel connections by default, which brings the publication time down to 44 seconds, more than 5 times faster with respect to the ZFS filesystem (234 seconds).

While these results cannot be generalized to all the workloads, they provide an insight into the performance impact that different storage types have on the publication workflow. In

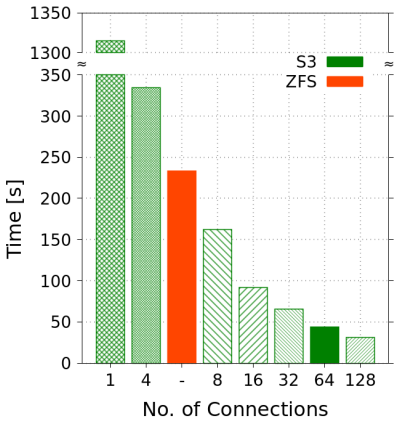

Figure 2. CVMFS publication time with a sample workload of 250 files, 4 kB each. Green bars show the obtained results with S3 and a variable number of parallel connections. The red bar shows the time taken on a ZFS filesystem. S3 with 64 parallel connections (default setting) is 5x faster than ZFS.

addition, they demonstrate the relevance of parallel transfers with S3-based storage services, which can be easily achievable thanks to the native RESTful protocol of S3.

5 Production Setup

5.1 Installation to CVMFS

To effectively exploit the new CVMFS capabilities, a new system was developed to orchestrate the installation of software on the LHCb-managed CVMFS instances. Celery [10] is a Python library for handling distributed task queues and was chosen as the basis for the design. It provides much of the basic functionality required such as retry logic, error handling, persistency, and a robust interface for inspecting and managing the state of the system. RabbitMQ was chosen as the messaging backend with a MySQL database (CERN DBoD [8]) being used to store the "result" of running each installation. The Celery application was implemented as a submodule of a Python package, `lbtaskrun`, with each task acting as a thin wrapper that calls the implementation in a separate submodule.

An additional Python package, `lbtaskweb`, was developed to contain a lightweight Flask application with the primary purpose of serving a REST API via HTTPS that can be used to trigger installations. As LHCb's CI infrastructure is split between a medium-sized Jenkins instance (connected to around 50 build nodes) and CERN's GitLab, it is desirable to be able to provide long-lived secrets which have extremely limited scope. This is achieved with dedicated bearer tokens that can be easily rotated if required. A global configuration file is used to allow installations to be rejected or re-prioritised based on the current demand. Additionally, `lbtaskweb` provides a lightweight web interface for experts to monitor the system and resubmit tasks if required.

The LHCb services are hosted on CERN OpenShift, a platform for hosting containerised applications, as four distinct applications:

- RabbitMQ: A two-pod RabbitMQ cluster is used as the messaging backend for the Celery application. As the AMQP port must be exposed outside of OpenShift to allow the CVMFS publishers to subscribe, SSL is used with certificates provided by Let's Encrypt.

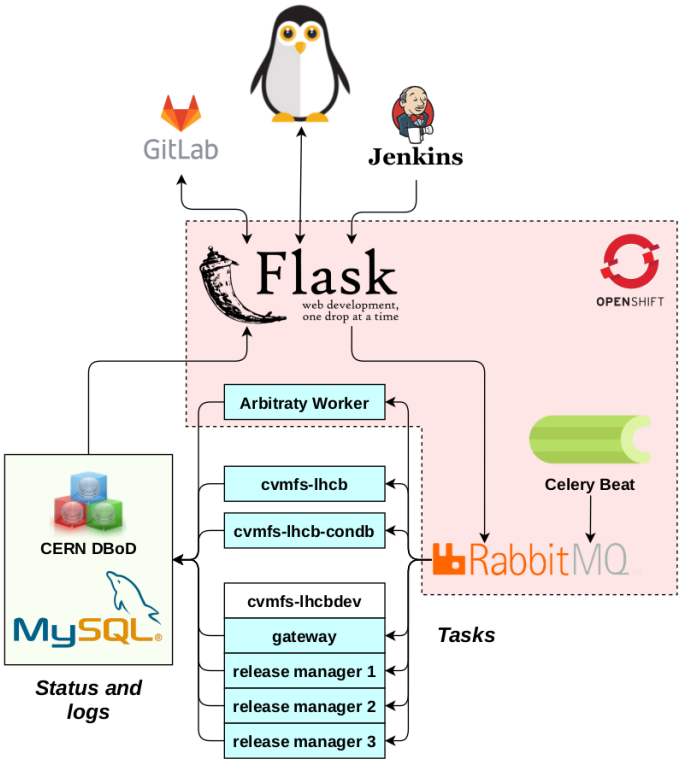

Figure 3. Architectural diagram of the system used to manage CVMFS installations in LHCb. The dotted red region shows components that are hosted within CERN's OpenShift instance.

- `lbtaskweb`: The flask-based API and web application.
- Celery Beat: A scheduler process that can be used to trigger periodic tasks, such as the update of the certificate authorities and certificate revocation lists that are copied to `/cvmfs/lhcb.cern.ch`.
- Arbitrary worker: A celery worker pod that is available for running tasks not directly related to CVMFS installation jobs.

An overview of this system is in Fig. 3.

5.2 Queues, Locking and Garbage Collection

The LHCb setup to deploy continuous integration artefacts to CVMFS features several publisher nodes (currently 3), on which a celery agent performs the installations. To ensure consistency, each transaction on the publisher must lock part of the CVMFS repository before proceeding. In the case of the LHCb nightly build outputs, the installer derives from its parameters the installation path and tries to take the lock. If no other publisher is currently installing in that subtree of the filesystem, it can just proceed. In case of parallel installation on another node, the lock cannot be acquired and the installation job is re-queued to be processed later. The granularity of the installation jobs allows that simple algorithm to perform well in the LHCb use cases as shown by the results in Fig. 4.

With the version of CVMFS used at this stage, it is not possible to prioritize processes trying to acquire a lock. Trying to lock the whole repository, as needed for garbage collection, is therefore difficult as it can only happen when no installations are running. This is not the case of `cvmfs-lhcbdev.cern.ch` where installation processes are running continuously. We therefore implemented a global lock, using the shared filesystem between all the nodes to solve this issue. This allows running the garbage collection but also makes it possible for the administrator to take control of the system when needed.

On `lhcbdev.cern.ch`, due to the high turnover of installations, the garbage collection has to run frequently in order to avoid the accumulation of unused files in the storage system. As the time to run grows with the number of files in the system, we run it every evening when the installation load is lower. Due to the CVMFS design, garbage collection must run when there are no other publications ongoing and has to run on the gateway node, which takes a global lock on the entire repository for the duration of the garbage collection process.

5.3 Other Installation Jobs

The system described in Sec. 5.1 allowed for the automation of the installation of LHCb's physics stack continuous integration artefacts, which represent the bulk of the load. However, it also enables the deployment of other software packages:

- The LHCb login scripts, a set of python packages deployed as a Conda [9] environment;
- Analysis tools, also deployed as a set of Conda environment;
- LHCb "Data Packages", reference data and scripts used to process the physics data.

In all those cases, Gitlab-CI [7] is used to validate and test the code, and, if the tests are positive, to trigger the installation on CVMFS. The most common use cases have now been integrated this way, reducing manual interventions on the LHCb CVMFS publisher nodes.

5.4 Monitoring

Monitoring for the LHCb managed components is split into two components. High-level metrics such as the number, status, and time since the last execution of each installation job type are exposed as a JSON blob by `lbtaskweb`. This is then scraped into CERN's `monit` [11] infrastructure to allow the metrics to be tracked over time. These logs are then published to a Grafana dashboard to provide basic alarms and an easy overview of the system health. Two of the plots from this dashboard are shown in Figure 4.

In addition, lower-level monitoring is achieved by aggregating all messages from Python's `logging` module with `python-logging-rabbitmq`[12]. This sends the logs to the same RabbitMQ instance as the one used by celery. Logstash [13] is then used to extract the logs from RabbitMQ and send them to a shared elasticsearch cluster where kibana can be used to monitor for errors and to browse individual logs if required.

5.5 Migration of `lhcbdev.cern.ch`

As the contents of `/cvmfs/lhcbdev.cern.ch` are predominantly short-lived, it was decided to create a new CVMFS repository from scratch instead of transferring the old content over S3. This allowed the old and new Stratum 0s to run in parallel for several weeks and for validating the new system in terms of stability, performance, and content installation.

Once validated, the final transition to the new repository was made in two steps. First, the revision number of the new system was bumped to be higher than the old Stratum 0 such that

Figure 4. Extract from the Grafana dashboard for monitoring LHCb's CVMFS installations with the server status (left) and the total number of installion tasks (right).

clients will consider its content as being up to date. Second, the Stratum 1 configuration was modified to serve the repository content from S3 storage (see Sec. 5.6 for further details).

This way of transitioning to a repository with entirely new content might provoke input/output errors on the client side. This is due to clients trying to fetch chunks referenced by CVMFS catalogs belonging to the old repository, while the storage already points to the new system (i.e., S3 storage) where the desired chunks do not exist. IO errors would manifest only for the amount of time during which the client is still locked on the latest revision of the old system before switching to the new one. This is typically very short (i.e., below 3 minutes), as clients constantly check for new revisions of the repository in order to be up to date.

As `/cvmfs/lhcbdev.cern.ch` is predominantly used for interactive development, the risk of IO errors was deemed acceptable once all users were notified. In practice, no users reported errors during the transition period and monitored clients demonstrated to switch seamlessly to the new system within few minutes.

5.6 Distribution to Clients

A crucial part of the CVMFS infrastructure are the Stratum 1 server and the site caches. Clients access the repository contents by addressing the Stratum 1 server through a set of hierarchical caches that improve the content distribution efficiency.

Since March 2020, LHCb repositories (i.e., `lhcb.cern.ch`, `lhcb-condb.cern.ch`, and `lhcbdev.cern.ch`) benefit from a dedicated set of caches at CERN made of 3 virtual machines spread across different OpenStack availability zone and network branches to achieve high availability. Such caches provide a very high hit ratio and, more importantly, better isolation from the load caused by other repositories on the content distribution infrastructure.

The Stratum 1 server traditionally makes replicas of repositories content and frequently runs snapshots to keep its replicas up to date. Similarly to the publisher nodes, also the Stratum 1 replicas are subject to garbage collection, which might introduce noticeable delays as no snapshots can run while garbage collection is in progress. In the case of high turnover repositories, such as `lhcbdev.cern.ch`, the garbage collection might take several hours, causing clients to stall on the latest revision replicated before garbage collection started. To avoid this problem, the Stratum 1 server at CERN has been reconfigured to serve the `lhcbdev.cern.ch` repository data directly from S3 (via `ProxyPass`). A local replica of the repository is still kept on the Stratum 1 as an emergency measure in case of S3 unavailability.

6 Conclusions

This paper presented recent achievements obtained with a major upgrade of the CVMFS infrastructure and the implementation of an advanced system to orchestrate installations. The

migration to S3 object storage carries a significant performance improvement, estimated in a 5x speedup, while the deployment of the CVMFS Gateway allows for parallel publications on different branches of the directory structure. At the moment, 3 publishers are used for `lhcbdev.cern.ch` and the collected monitoring metrics show that most of the time is spent in processing the workload to be published. This suggests that the access to the storage is not saturated and that the system could scale horizontally by adding additional publishers. Orchestration to regulate installations is equally important and should allow taking advantage of the infrastructure underneath. LHCb has re-engineered its system to manage CVMFS installations which now embeds task queue, retry logic, and error handling capabilities. This system demonstrated to be reliable for production scenarios, satisfies today's needs for publishing on CVMFS, and is easily extensible to adhere to future requirements.

Some limitations still exist in the current implementation of the CVMFS Gateway. For instance, it is not possible to assign a degree of priority to publications so that the gateway knows which publisher should take the lock. At the moment, all the publications have equal weight in CVMFS and priorities should be managed by the system responsible for orchestrating publications. Also, garbage collection requires a global lock on the repository and can be executed on the gateway only. While the global lock will still be a requirement, it is foreseen to extend the gateway APIs so that publishers can trigger garbage collection.

References

[1] J. Blomer et al. "Distributing LHC application software and conditions databases using the CernVM file system" Journal of Physics: Conference Series. Vol. 331. No. 042003 IOP Publishing, 2011.

[2] J. Blomer et al. "Towards a serverless CernVM-FS" EPJ Web Conf. Vol. 331. 2019.

[3] T. Stefan "Testing and Improving Deployment of ATLAS Releases to CVMFS" CERN technical report CERN-STUDENTS-Note-2018-180. 2018.

[4] Amazon Simple Storage Service, https://aws.amazon.com/s3/ (last accessed 12/02/2021)

[5] Weil, Sage A., et al. "Ceph: A scalable, high-performance distributed file system." Proceedings of the 7th symposium on Operating systems design and implementation. 2006.

[6] Traefik, The Cloud Native Application Proxy, https://traefik.io/traefik/ (last accessed 12/02/2021)

[7] GitLab, The complete DevOps platform, https://about.gitlab.com/ (last accessed 16/02/2021)

[8] Aparicio, R. G., and Coterillo Coz I. "Database on Demand: insight how to build your own DBaaS." Journal of Physics: Conference Series. Vol. 664. No. 4. IOP Publishing, 2015.

[9] Conda. Package, dependency and environment management for any language, https://docs.conda.io/en/latest/ (last accessed 16/02/2021)

[10] Celery. Distributed Task Queue, https://docs.celeryproject.org/en/stable/ (last accessed 16/02/2021)

[11] MONIT. Monitoring for CERN IT Data Centre and WLCG Infrastructure, https://monit.web.cern.ch/ (last accessed 16/02/2021)

[12] Logging handler to ships logs to RabbitMQ, https://github.com/albertomr86/python-logging-rabbitmq (last accessed 16/02/2021)

[13] Logstash. Centralize, transform & stash your data, https://www.elastic.co/logstash (last accessed 16/02/2021)