RabbitMQ@LHCb

EP-SFT Group Meeting
22/10/2018
Ben Couturier (for the LHCb Computing team)

Context: Releases in the CVMFS lhcbdev stratum-0

LHCb Continuous builds are built using jenkins and every day:

- ~25 different software stacks are built with several sets of options
- ~200 GiB and 2.5e6 files installed on CVMFS (we keep them ~ 10 days)

CVMFS publish/ Garbage collecting are very time consuming :

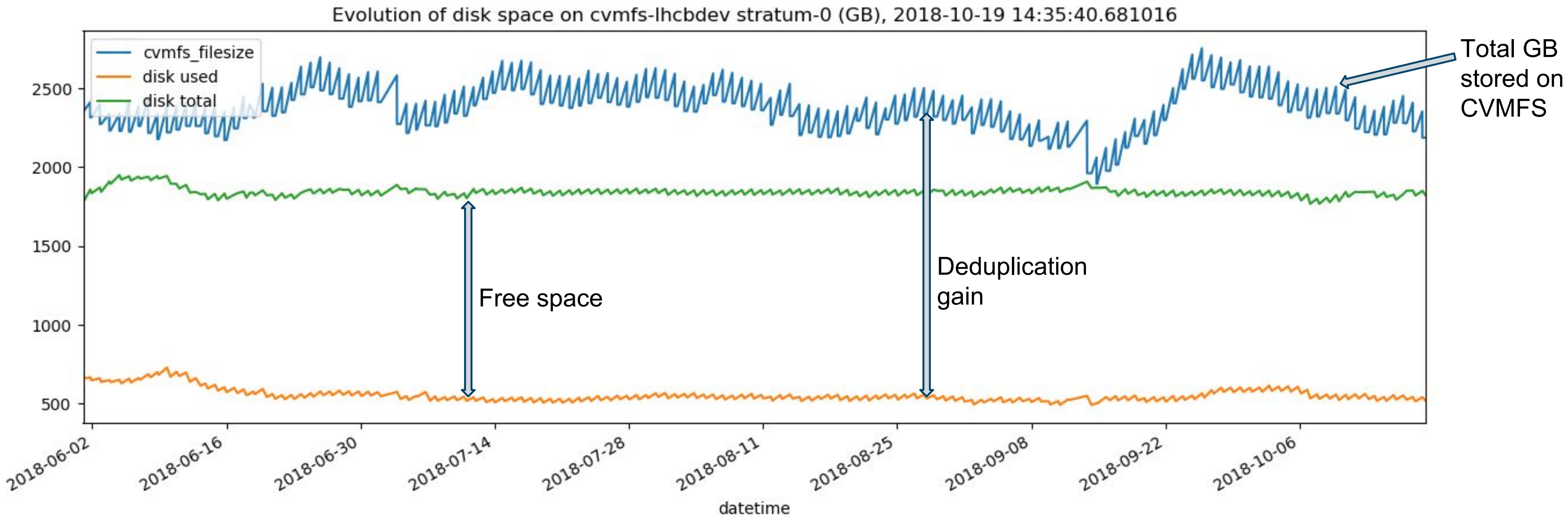
- Limiting factor for the nightly builds: large transactions can take hours to publish
- This can be a problem when trying to get files to developers as early as possible
- More acute problem since we moved all continuous builds outputs to CernVM-FS

Trying to improve the deployment infrastructure (c.f. Jakob and Radu's work) but in the meantime we needed a workaround

Tasks on the CVMFS stratum-0:

- Install nightlies as soon as they are built
- fetch some git repos
- Install extra packages on request (gitlab-ci driven)
- GC once a day (takes around 4 hours)

CernVM-FS: lhcbdev.cern.ch disk usage



System has been stable for months

Installation process v1

CVMFS constraints:

- ➔ Transaction need to be serialized
- ➔ Publish process is sensitive to open file descriptors on /cvmfs

Jenkins is used to produce the nightly builds

- ➔ Pushing directly to CVMFS didn’t seem a great option (CVMFS publish fragile, no coordination of publication, locking needed, jobs blocked during publish and therefore slots unused, difficult to get a global picture…)
  Artefacts pushed to EOS instead
- ➔ Command line tool (*lbn-install*) allows installation of a slot of a given build ID

First iteration of the installer:

- ➔ CRON job *lbn-install*s slot one by one in a predefined order
- ➔ Large transactions, taking a long time to complete

Slowness in publication resulted in bad user experience (and we had no leverage to prioritize deployments)

Reviewing requirements

Essentially we wanted to:

- ➔ Install packages as soon as they are available
- ➔ In case of queue (i.e. all the time), prioritize the deployment of slots and platforms to make sure most useful content gets available first

Decreased the granularity of the CVMFS transactions and therefore rewrote the scripts

Started playing with the order various slots in our scripts and they way they were installed

> *Essentially started writing a (very very) bad message queue...*

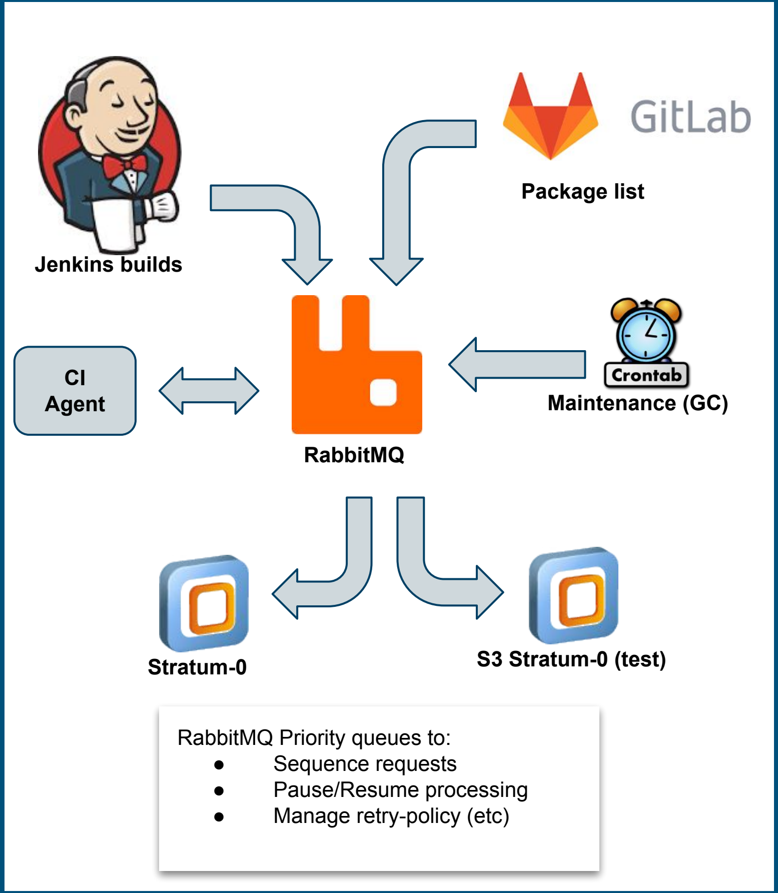
Then we started investigated lightweight message queue systems to implement the new system

New system design

- ➔ RabbitMQ server (https://www.rabbitmq.com/) with persistent queues (and priorities)
- ➔ Python client developed using Pika (https://pika.readthedocs.io/en/stable/)

- ➔ Jenkins publishes message when individual project builds are ready
- ➔ CI Agent decides on what should be installed and priority
  - We can for example expedite some installations (or take manual control)
- ➔ Python client run by stratum-0 picks up the messages and performs installations
  - Set of python packages to manage the code e.g. https://lhcb-pypi.web.cern.ch/lhcb-pypi/simple/lbcvmfstools/
  - Python context manager to deal with CVMFS transactions

Dev team: S.Chitic, B. Couturier



RabbitMQ Priority queues to:

- Sequence requests
- Pause/Resume processing
- Manage retry-policy (etc)

Why RabbitMQ

Many advantages:

- ➔ Simple to install
- ➔ Easy to to program (well documented) python API (pika) with rich functionality (ack, persistent queues…)
- ➔ Decoupling between message producer and consumer (exchange/queue mechanism)
- ➔ Easy to install run (clustering possible, but that’s overkill…)

And also, but we didn’t really care:

- ➔ Industry standard protocol (AMQP)
- ➔ Very performant engine (we are several order of magnitudes below the engine capabilities)
- ➔ Advances features like clustering...

Other engines would do for sure, but RabbitMQ was fit for purpose...

Advantages of a Message Queue

Complete decoupling of producer/consumer

- ➔ Decouple install configuration from Jenkins (e.g. allows second installation for S3 test transparently, c.f. exchange/queue binding system)
- ➔ Robust error management possible (error queues, functionality richer than Jenkins retries and easier to code)
- ➔ Persistent queue survives broker restart (unlike Jenkins jobs…)

Priority queue is part of the functionality

- ➔ CI Agent subscribes to messages from Jenkins, and decides on what to install and priority
- ➔ Made it possible to prioritize one platform per slot so that all developers are happy
- ➔ Manual intervention possible by submitting top priority message

Message queue can be used by other system:

- ➔ E.g. to start HLT validation test of the HLT on LHCb Online (easier this way than starting a jenkins slave)
- ➔ […]

Drawbacks of a Message Queue

Of course this is extra code:

- ➔ That we would have prefered not to have to write, but once you have to...
- ➔ pika is really easy to code, not much boilerplate

Extra server to maintain:

- ➔ Rather nice admin tool
- ➔ In practice, not a big deal…

Conclusion

Started using RabbitMQ when the complexity of the deployment scripts increased

- Investment paid off as the messaging system is very robust (modulo a few initial bugs of course)
- RabbitMQ server itself requires very little maintenance
- Allowed us to deal with deployment performance issues and even to add test instances...