LHCbDIRAC Documentation

Release v7

LHCbDIRAC Project.

Mar 27, 2019

Contents

The LHCbDIRAC project is the LHCb Grid solution. LHCbDIRAC is DIRAC extension.

DIRAC forms a layer between a particular community and various compute resources to allow optimized, transparent and reliable usage. LHCbDIRAC specializes DIRAC for LHCb.

- DIRAC documentation: http://dirac.readthedocs.io/en/latest/index.html
- DIRAC hosted repository: https://github.com/DIRACGrid

LHCbDIRAC is the LHCb extension to DIRAC:

- LHCbDIRAC documentation: http://lhcb-dirac.readthedocs.io/en/latest/index.html
- LHCbDIRAC hosted repository: https://gitlab.cern.ch/lhcb-dirac

CHAPTER 1

Developers Guide

1.1 Guide for developing LHCbDIRAC (and DIRAC, for LHCb developers)

A short, but hopefully comprehensive guide on developing in LHCbDIRAC, referencing DIRAC development model. For what are DIRAC and LHCbDIRAC doing, look elsewhere.

LHCbDIRAC is a DIRAC extension. This means that LHCbDIRAC cannot leave independently from DIRAC. There are a number of DIRAC extensions, maintained by various communities worldwide, and LHCbDIRAC is the most important out there, and the one that receives the most support by DIRAC itself. But it also means that DIRAC and LHCbDIRAC (as all the other DIRAC extensions) have different release cycles and versioning, adopts different version control systems, use different tracking systems, and that the code conventions may slightly differ.

DIRAC can also have other extensions, independent from a VO. All these are hosted at github.

1.1.1 Pre-requisites

Within this section we just look at what is necessary to know before looking at the code.

Releases

Naming

Both DIRAC and LHCbDIRAC follow the same naming conventions for releases, inherithed by the LHCb convention:

vMrNpt

where:

- **M** stands for *major version*, or simply *version*
- **N** stands for *minor version*, or simply *release*

- **t** stands for *patch version*, or simply *patch*

with a special *pre-release* naming convention: **-preX**.

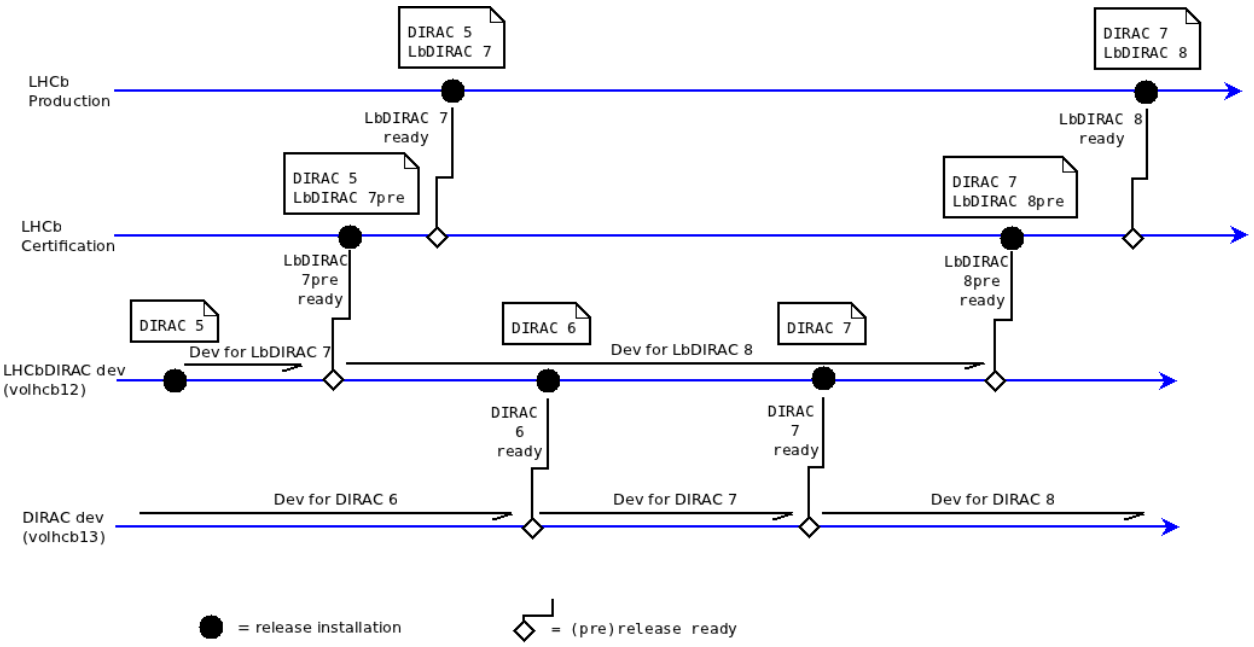
This will be clear with some examples:

- **v6r2p0** is the version 6, release 2, patch 0
- **v7r5p13** is the version 7, release 5, patch 13
- **v8r1-pre2** is the second pre-release of version 8, release 1

There are no pre-releases for patches.

Release cycle

When developing LHCbDIRAC, we need to consider that every LHCbDIRAC is developed on top of a DIRAC release. The following picture explaines the model.



So, for example. there might be 2 or more LHCbDIRAC releases based on top of the same DIRAC release. Every LHCbDIRAC developers has to know which release of DIRAC its development is for. The major version of both DIRAC and LHCbDIRAC changes rarely, let's say evry 2 years. The minor version changes more frequently in LHCbDIRAC with respect to DIRAC, but there is no strict advancement scheduling for none of the 2.

A pre-release is a release candidate that goes through a certification process.

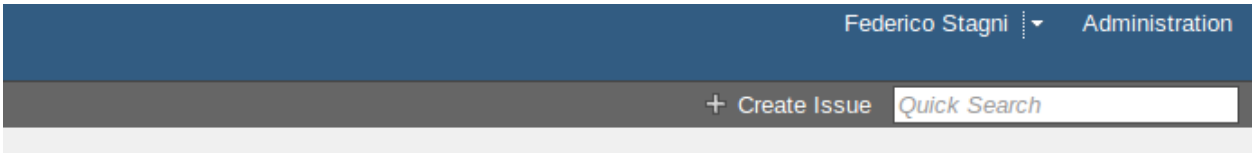
Version Control

LHCbDIRAC version control is based on GIT. GIT is a very popular distributed revision control system. The reader is supposed to be familiar with the way such systems work. The code is hosted in the CERN GitLab.

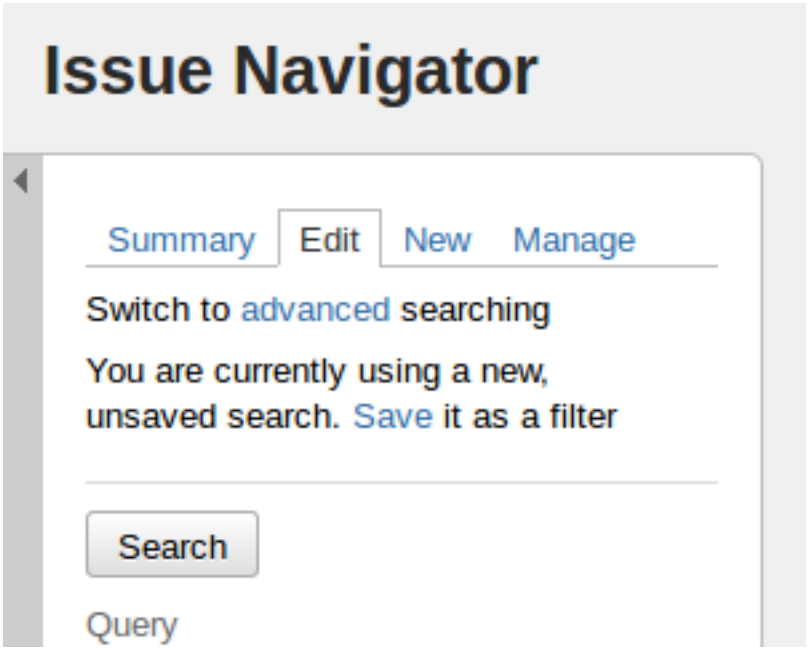
Tracking systems

The tracking system used for LHCbDIRAC is jira. Jira is a fundamental tool for LHCbDIRAC, and its use is mandatory. Every development should be tracked there. Jira is a very powerfool tool, but requires some time to master. Few notes/links:

- The *official documentation* is here. You might also be interested in watching the first ~15 minutes of this video.
- Issuing a new bug/task/story/etc. (there are many possible choices) is easy, just look at the top right of the screen:



- Remember to put a "component" when you make a new issue
- When you make a new research in the issue navigator, you can save the search: it will become useful later



Developer tools

You are free to choose the editor or IDE you prefer. I know Emacs is a great tool, and someone can't just leave without it. And that also vim is great. Eclipse with pydev is another good choice. Other possibilities include PyCharm (IntelliJIDEA) and atom

1.2 Developing DIRAC and LHCbDIRAC

Developing the code is not just about editing. You also want to "run" something, usually for testing purposes. The DIRAC way of developing can be found here and it applies also to LHCbDIRAC. Please follow carefully especially what's here

In general, if you are developing LHCbDIRAC, you should consider that:

- everything that applies to DIRAC development, also applies to LHCbDIRAC development, so, follow carefully the links above
- every LHCbDIRAC release has a strong dependency with a DIRAC release. See https://gitlab.cern.ch/lhcb-dirac/LHCbDIRAC/blob/master/CONTRIBUTING.md for more info.

1.3 HOW TOs

1.4 Browsing the code running in production

If you want to browse the DIRAC (and LHCbDIRAC) code running in production you'll first of all have to know which version is installed. Announcements of new deployments are done via the LHCb operations eLog. The code is also always installed in the CVMFS release area (*$LHCb_release_area/DIRAC/DIRAC_vX5rYpZ/DIRAC*) but you can normally use git to switch from one to another.

1.5 I developed something, I want it in the next release

Just open a merge request to the devel branch of LHCbDirac: all the releases (minor and major) are created branching from this branch.

1.6 Asking for a LHCbDIRAC patch

Just open a merge request to the master branch of LHCbDirac. If in a hurry, drop an e-mail to the lhcb-dirac mailing list.

CHAPTER 2

Administrator Guide

This page is the work in progress. See more material here soon !

2.1 LHCbDIRAC Releases

The following procedure applies fully to LHCbDIRAC production releases, like patches. For pre-releases (AKA certification releases, there are some minor changes to consider).

2.1.1 Prerequisites

The release manager needs to:

- be aware of the LHCbDIRAC repository structure and branching as highlighted in the contribution guide.
- have forked LHCbDIRAC on GitLab as a "personal project" (called "origin" from now on)
- have cloned origin locally
- have added https://gitlab.cern.ch/lhcb-dirac/LHCbDIRAC as "upstream" repository to the local clone
- have push access to the master branch of "upstream" (being part of the project "owners")
- have DIRAC installed
- have been grated write access to <webService>
- have "lhcb_admin" or "diracAdmin" role.
- have a Proxy

The release manager of LHCbDIRAC has the triple role of:

1. creating the release
2. making basic verifications
3. deploying it in production

2.1.2 1. Creating the release

Unless otherwise specified, (patch) releases of LHCbDIRAC are usually done "on top" of the latest production release of DIRAC. The following of this guide assumes the above is true.

Creating a release of LHCbDIRAC means creating a tarball that contains the release code. This is done in 3 steps:

1. Merging "Merge Requests"
2. Propagating to the devel branch
3. Creating the release tarball, add uploading it to the LHCb web service

But before:

Pre

Verify what is the last tag of DIRAC:

```
# it should be in this list:
git describe --tags $(git rev-list --tags --max-count=10)
```

A tarball containing it is should be already uploaded here

You may also look inside the .cfg file for the DIRAC release you're looking for: it will contain an "Externals" version number, that should also be a tarball uploaded in the same location as above.

If all the above is ok, we can start creating the LHCbDIRAC release.

Merging "Merge Requests"

Merge Requests (MR) that are targeted to the master branch and that have been approved by a reviewer are ready to be merged

If there are no MRs, or none ready: please skip to the "update the CHANGELOG" subsection.

Otherwise, simply click the "Accept merge request" button for each of them.

Then, from the LHCbDIRAC local fork you need to update some files:

```
# if you start from scratch otherwise skip the first 2 commands
mkdir $(date +20%y%m%d) && cd $(date +20%y%m%d)
git clone https://:@gitlab.cern.ch:8443/lhcb-dirac/LHCbDIRAC.git
git remote add upstream https://:@gitlab.cern.ch:8443/lhcb-dirac/LHCbDIRAC.git
# update your "local" upstream/master branch
git fetch upstream
# create a "newMaster" branch which from the upstream/master branch
git checkout -b newMaster upstream/master
# determine the tag you're going to create by checking what was the last one from the␣
↪following list (add 1 to the "p"):
git describe --tags $(git rev-list --tags --max-count=5)
# Update the version in the __init__ file:
vim LHCbDIRAC/__init__.py
# Update the version in the releases.cfg file:
vim LHCbDIRAC/releases.cfg
# Update the version in the Dockerfile file:
vim container/lhcbdirac/Dockerfile
# For updating the CHANGELOG, get what's changed since the last tag
t=$(git describe --abbrev=0 --tags); git --no-pager log ${t}..HEAD --no-merges --
↪pretty=format:'* %s';
```

(continues on next page)

(continued from previous page)

```
# copy the output, add it to the CHANGELOG (please also add the DIRAC version)
vim CHANGELOG # please, remove comments like "fix" or "pylint" or "typo"...
# Change the versions of the packages
vim dist-tools/projectConfig.json
git add -A && git commit -av -m "<YourNewTag>"
```

Time to tag and push:

```
# make the tag
git tag -a <YourNewTag> -m <YourNewTag>
# push "newMaster" to upstream/master
git push --tags upstream newMaster:master
# delete your local newMaster
# before change your branch use git checkout "existing branch name"
git branch -d newMaster
```

Remember: you can use "git status" at any point in time to make sure what's the current status.

Propagate to the devel branch

Now, you need to make sure that what's merged in master is propagated to the devel branch. From the local fork:

```
# get the updates (this never hurts!)
git fetch upstream
# create a "newDevel" branch which from the upstream/devel branch
git checkout -b newDevel upstream/devel
# merge in newDevel the content of upstream/master
git merge upstream/master
```

The last operation may result in potential conflicts. If happens, you'll need to manually update the conflicting files (see e.g. this guide). As a general rule, prefer the master fixes to the "HEAD" (devel) fixes. Remember to add and commit once fixed. Note: For porting the LHCbDIRAC.init.py from master to devel, we prefer the HEAD version (only for this file!!!)

Plase fix the conflict if some files are conflicting. Do not forget to to execute the following:

```
git add -A && git commit -m " message"
```

Conflicts or not, you'll need to push back to upstream:

```
# push "newDevel" to upstream/devel
git push upstream newDevel:devel
# delete your local newDevel
git branch -d newDevel
# keep your repo up-to-date
git fetch upstream
```

Creating the release tarball, add uploading it to the LHCb web service

Automatic procedure

When a new git tag is pushed to the repository, a gitlab-ci job takes care of (soon testing), creating the tarball, uploading it to the web service, and to build the docker image. You can check it in the pipeline page of the repository (https://gitlab.cern.ch/lhcb-dirac/LHCbDIRAC/pipelines).

It may happen that the pipeline fails. There are various reasons for that, but normally, it is just a timeout on the runner side, so just restart the job from the pipeline web interface. If it repeatedly fails building the tarball, try the manual procedure described bellow to understand.

Manual procedure

This should a priori not be used anymore. If the pipeline fails, you should rather investigate why.

Login on lxplus, run

```
lb-run LHCbDirac/prod bash -norc

git archive --remote ssh://git@gitlab.cern.ch:7999/lhcb-dirac/LHCbDIRAC.git devel␣
↪LHCbDIRAC/releases.cfg  | tar -x -v -f - --transform 's|^LHCbDIRAC/||' LHCbDIRAC/
↪releases.cfg

dirac-distribution -r v8r3p1 -l LHCb -C file:///`pwd`/releases.cfg (this may take␣
↪some time)
```

Don't forget to read the last line of the previous command to copy the generated files at the right place. The format is something like:

```
( cd /tmp/joel/tmpxg8UuvDiracDist ; tar -cf - *.tar.gz *.md5 *.cfg ) | ssh␣
↪lhcbprod@lxplus.cern.ch 'cd /afs/cern.ch/lhcb/distribution/DIRAC3/tars &&  tar -xvf␣
↪- && ls *.tar.gz > tars.list'
```

And just copy/paste/execute it.

If you do not have access to lhcbprod, you can use your user name.

2.1.3 2. Making basic verifications

Once the tarball is done and uploaded, the release manager is asked to make basic verifications, via Jenkins, if the release has been correctly created.

At this link you'll find some Jenkins Jobs ready to be started. Please start the following Jenkins jobs and come back in about an hour to see the results for all of them.

1. https://lhcb-jenkins.cern.ch/jenkins/view/LHCbDIRAC/job/!RELEASE!__pylint_unit/ the !RELEASE! is the actual relase for example: https://lhcb-jenkins.cern.ch/jenkins/view/LHCbDIRAC/job/v8r5__pylint_unit/

This job will: run pylint (errors only), run all the unit tests found in the system, assess the coverage. The job should be considered successful if:

- the pylint error report didn't increase from the previous job run
- the test results didn't get worse from the previous job run
- the coverage didn't drop from the previous job run

2. https://lhcb-jenkins.cern.ch/jenkins/view/LHCbDIRAC/job/!RELEASE!__pilot/

This job will simply install the pilot. Please just check if the result does not show in an "unstable" status

3. https://lhcb-jenkins.cern.ch/jenkins/view/LHCbDIRAC/job/!RELEASE!__/

   TODO

2.1.4 3. Advertise the new release

Before you start the release you must write an Elog entry 1 hour before you start the deployment. You have to select Production and Release tick boxes. When the intervention is over you must notify the users (reply to the Elog message).

2.1.5 4. Deploying the release

Deploying a release means deploying it for the various installations:

```
* client
* server
* pilot
```

release for client

Please refer to this TWIKI page a quick test to validate the installation is to run the SHELL script $LHCBRELEASE/LHCBDIRAC/LHCBDIRAC_vXrY/LHCbDiracSys/test/client_test.csh

go to this web page for asking to install the client release in AFS and CVMFS:

- in the field "Project list" put : "Dirac vNrMpK LHCbGrid vArB LHCbDirac vArBpC" (NOTE: LHCbGrid version can be found: https://gitlab.cern.ch/lhcb-dirac/LHCbDIRAC/blob/master/dist-tools/projectConfig.json)
- in the field "platforms" put : "x86_64-slc6-gcc48-opt x86_64-slc6-gcc49-opt x86_64-slc6-gcc62-opt x86_64-centos7-gcc62-opt"

Then click on the "BUILD" button

- within 10-15 min the build should start to appear in the nightlies page https://lhcb-nightlies.cern.ch/release/
- if there is a problem in the build, it can be re-started via the dedicated button (it will not restart by itself after a retag)
- The build for gcc48 is known to have missing dependencies, but must be released anyway.

If it is the production release, and only in this case, once satisfied by the build, take note of the build id (you can use the direct link icon) and make the request via https://sft.its.cern.ch/jira/browse/LHCBDEP.

- **NOTE: If some package is already released, please do not indicate in the Jira task. For example: a Jira task when:**
  - DIRAC is not released, then the message in the JIRA task: Summary:Dirac v6r14p37 and LHCbDirac v8r2p50; Description: Please release Dirac and LHCbDirac in this order based on build 1526;
  - **DIRAC is released, then the message in the JIRA task: Summary:LHCbDirac v8r2p50; Description: Please rele**
    - Dependency is not fulfilled for the platform: x86_64-slc6-gcc48-opt please ask to force the release using –no-strict option

Once the client has been deployed, you should setup the correct environment (lb-run LHCbDIRAC/<version> bash –norc), pref

- Minimal test: https://gitlab.cern.ch/lhcb-dirac/LHCbDIRAC/blob/master/tests/System/Client/basic-imports.py
- Bigger (certification like) test: https://gitlab.cern.ch/lhcb-dirac/LHCbDIRAC/blob/master/tests/System/Client/client_test.sh

Changing the prod version for Pilot

ask the CVMFS librarians to change the prod version for the pilot on cvmfs. The commands for changing the prod:

```
cd /cvmfs/lhcb.cern.ch/lib/lhcb/LHCBDIRAC
rm LHCBDIRAC_prod; ln -s LHCBDIRAC_vArBpC LHCBDIRAC_prod
```

Server

To install it on the VOBOXes from lxplus:

```
lhcb-proxy-init -g diracAdmin
dirac-admin-sysadmin-cli --host volhcbXX.cern.ch
>update LHCbDIRAC-v8r3p32
>restart *
```

The (better) alternative is using the web portal or using the following script: https://gitlab.cern.ch/lhcb-dirac/LHCbDIRAC/blob/devel/dist-tools/create_vobox_update.py

The recommended way is the following:

```
ssh lxplus
mkdir DiracInstall; cd  DiracInstall
wget https://gitlab.cern.ch/lhcb-dirac/LHCbDIRAC/blob/devel/dist-tools/create_vobox_
↪update.py
python create_vobox_update.py vArBpC
```

This command will create 6 files called "vobox_update_MyLetter" then you can run in 6 windows the recipe for one single machine like that:

```
ssh lxplus
cd  DiracInstall ; lb-run LHCbDIRAC/prod bash -norc ; lhcb-proxy-init -g lhcb_admin;␣
↪dirac-admin-sysadmin-cli
and from the prompt ::
   [host] : execfile vobox_update_MyLetter
   [host] : quit
```

Note:

It is normal if you see the following errors:

> –> Executing restart Framework SystemAdministrator [ERROR] Exception while reading from peer: (-1, 'Unexpected EOF')

In case of failure you have to update the machine by hand. Example of a typical failure:

```
--> Executing update v8r2p42
Software update can take a while, please wait ...
[ERROR] Failed to update the software
Timeout (240 seconds) for '['dirac-install', '-r', 'v8r2p42', '-t', 'server', '-e',
↪'LHCb', '-e', 'LHCb', '/opt/dirac/etc/dirac.cfg']' call
```

Login to the failing machine, become dirac, execute manually the update, and restart everything. For example:

```
ssh lbvobox11
sudo su - dirac
dirac-install -r v8r2p42 -t server -e LHCb -e LHCb /opt/dirac/etc/dirac.cfg
```

(continues on next page)

(continued from previous page)

```
lhcb-restart-agent-service
runsvctrl t startup/Framework_SystemAdministrator/
```

Specify that this error can be ignored (but should be fixed !):

```
2016-05-17 12:00:00 UTC dirac-install [ERROR] Requirements installation script /opt/
↪dirac/versions/v8r2p42_1463486162/scripts/dirac-externals-requirements failed.␣
↪Check /opt/dirac/versions/v8r2p42_1463486162/scripts/dirac-externals-requirements.
↪err
```

Using the web portal:

- You cannot do all the machines at once. Select a bunch of them (between 5 and 10). Fill in the version number and click update.
- Repeate until you have them all.
- Start again selecting them by block, but this time, click on "restart" to restart the components.

WebPortal

When the web portal machine is updated then you have to compile the WebApp:

```
ssh lhcb-portal-dirac.cern.ch
sudo su - dirac
dirac-install -r VERSIONTOBEINSTALLED -t server -l LHCb -e LHCb,LHCbWeb,WebAppDIRAC /
↪opt/dirac/etc/dirac.cfg (for example: dirac-install -r v8r4p2 -t server -l LHCb -e␣
↪LHCb,LHCbWeb,WebAppDIRAC /opt/dirac/etc/dirac.cfg)
dirac-webapp-compile
```

When the compilation is finished:

```
lhcb-restart-agent-service
runsvctrl t startup/Framework_SystemAdministrator/
```

TODO

When the machines are updated, then you have to go through all the components and check the errors. There are two possibiliti

1. Use the Web portal (SystemAdministrator)
2. Command line:

   for h in $(grep 'set host' vobox_update_* | awk { 'print $NF' }); do echo "show errors" | dirac-admin-sysadmin-cli -H $h; done | less

Pilot

Use the following script (from, e.g., lxplus after having run *lb-run LHCbDIRAC tcsh*):

```
dirac-pilot-version -S v8r2p42
```

NOTE: YOU HAVE TO KEEP TWO PILOT VERSION. AFTER YOU EXECUTED THIS COMMAND PLEASE MODIFY THE CS! for example:/Operation/LHCb-Production/Pilot/Version to v8r2p42, v8r241 The newer version should be the first in the list

for checking and updating the pilot version. Note that you'll need a proxy that can write in the CS (i.e. lhcb-admin). This script will make sure that the pilot version is update BOTH in the CS and in the json file used by pilots started in the vacuum.

Basic instruction how to merging the devel branch into master (NOT for PATCH release)

Our developer model is to keep only two branches: master and devel. When we made a major release we have to merge devel to master. Before the merging please create a new branch based on master using the web interface of GitLab. This is for safety. After you can merege devel to master:

```
mkdir $(date +20%y%m%d) && cd $(date +20%y%m%d)
git clone ssh://git@gitlab.cern.ch:7999/lhcb-dirac/LHCbDIRAC.git
cd LHCbDIRAC
git remote rename origin upstream
git fetch upstream
git checkout -b newMaster upstream/master
git merge upstream/devel
git push upstream newMaster:master
```

2.1.6 5. Mesos cluster

Mesos is currently only used for the certification. In order to push a new version on the Mesos cluster, 3 steps are needed:

- Build the new image
- Push it the lhcbdirac gitlab repository
- Update the version of the running containers

Automatic procedure

The first two steps should be automatically done by the gitlab-ci of the LHCbDIRAC repository. The last step will be taken care of by the gitlab-ci of the MesosClusterConf repository (https://gitlab.cern.ch/lhcb-dirac/MesosClusterConf) For a simple version upgrade, edit directly on the gitlab web page the file clusterConfiguration.json and replace the "version" attribute with what you want. Of course add a meaningful commit message.

Manual procedure

This should in principle not happen. Remember that any manual change of the mesos cluster will be erased next time the gitlab-ci of the MesosClusterConf repository will run. However, you can do all the above step manually.

All these functionalities have been wrapped up in a script (dirac-docker-mgmt), available on all the lbmesosadm* machines (01, 02)

The next steps are the following:

```
# build the new image
# this will download the necessary files, and build
# the image localy
```

(continues on next page)

(continued from previous page)

```
dirac-docker-mgmt.py -v v8r5 --build

# Push it to the remote lhcbdirac registry
# Your credentials for gitlab will be asked
dirac-docker-mgmt.py -v v8r5 --release

# Update the version of the running containers
# The services and number of instances running
# will be preserved
dirac-docker-mgmt.py -v v8r5 --deploy
```

2.2 Renewal of certificate for ONLINE machine

Login as lhcbprod on **lbdirac.cern.ch** and generate the certificate request

```
openssl req -new -subj /CN=lbdirac.cern.ch -out newcsr.csr -nodes -sha1
```

Open in your browser the page http://ca.cern.ch cut the content of *newcsr.csr* (created in the previous step) in the web page and click on the submit button. Save the Base 64 encoded certificate as a file *newcert.cer*. Copy this file to lbdirac.cern.ch. Then convert the certificate in the correct format.

```
openssl pkcs12 -export -inkey privkey.pem -in newcert.cer -out myCertificate.pks (You␣
↪will have to type the PEM password you typed in the previous step. Type also an␣
↪export password, and don't forget it. Your certificate in PKCS12 format is ready in␣
↪file myCertificate.pks, you can delete the other files.)
openssl pkcs12 -in myCertificate.pks -clcerts -nokeys -out hostcert.pem
openssl pkcs12 -in myCertificate.pks -nocerts -out hostkey.pem.passwd
openssl rsa -in hostkey.pem.passwd -out hostkey.pem (remove the password)
```

If you want to test that the new host certificate is valid without any password, just do

```
dirac-proxy-init -C <cert> -K <key>
```

2.3 ONLINE steps

2.3.1 Installation of LHCbDirac

The machine running the transfers from the pit is lbdirac, and is in the online network. This machine runs:

- A complete RMS: ReqManager (url: RequestManagement/onlineGateway), a ReqProxy (known only from inside) and a RequestExecutingAgent
- The RAWIntegrity system: the RAWIntegrityHandler and RAWIntegrityAgent

A special catalog is defined in the local configuration in order to keep track of the files transfered:

```
RAWIntegrity
{
  AccessType = Read-Write
  Status = Active
}
```

We also have two special configuration for StorageElements:

```
# Setting it to NULL to transfer without
# checking the checksum, since it is already done by
# the DataMover and the RAWIntegrityAgent
# It should avoid the double read on the local disk
ChecksumType=NULL
# Setting this to True is dangerous...
# If we have a SRM_FILE_BUSY, we remove the file
# But we have enough safety net for the transfers from the pit
SRMBusyFilesExist = True
```

Finally, you need to overwrite the URLS of the RMS to make sure that they use the internal RMS:

```
URLs
{
  ReqManager = dips://lbdirac.cern.ch:9140/RequestManagement/ReqManager
  ReqProxyURLs = dips://lbdirac.cern.ch:9161/RequestManagement/ReqProxy
}
```

2.3.2 Workflow

The DataMover is the Online code responsible for the interraction with the BKK (register the run, the files, set the replica flag), to request the physical transfers, and to remove the file of the Online storage when properly transfered.

The doc is visible here: https://lbdokuwiki.cern.ch/online_user:rundb_onlinetoofflinedataflow

The DataMover registers the Run and the files it already knows about in the BKK. Then it creates for each file a request with a PutAndRegister operation. The target SE is CERN-RAW, the Catalog is RAWIntegrity. The RequestExecutingAgent will execute the copy from the local online storage to CERN-RAW, and register it in the RAWIntegrity DB.

The RAWIntegrityAgent looks at all the files in the DB that are in status 'Active'.

For each of them, it will check if the file is already on tape, and if so, compare the checksum.

If the checksum is incorrect, the file remains in status 'Active', and will require manual intervention. If the checksum is correct, we attempt to register the file in the DFC only.

If the registration fails, the file goes into 'Copied' status in the DB, c If the registration works, we attempt to remove the file from the Online storage. This removal Request sends a signal to the DataMover, which will mark the file for removal (garbage collection), and the replica flag to yes in the BKK.

If the removal fails, the file status is set to 'Registered' in the DB, and will be reattempted from there at the next loop. If the removal works, the file is set to 'Done' in the DB.

2.4 Data distribution

2.4.1 Archive

The defautl option is at *Operations/<Setup>/TransformationPlugins/Archive2SEs*. it can be overwritten in each plugin. The choice is done randomly.

2.4.2 DST broadcast

The broadcast done by LHCbDSTBroadcast plugin is done according to the free space

2.5 RAW files processing and distribution

The RAW files all have a copy at CERN, and are then distributed across the Tier1. The processing is shared between CERN and the Tier1.

The selection of the site for copying the data and the site where the data will be processed (so called *RunDestination*) is done by the *RAWReplication* plugin. To do so, it uses shares that are defined in *Operations/<Setup>/Shares*

2.5.1 Selection of a Tier1 for the data distribution

The quota are defined in *Operations/<Setup>/Shares/RAW*.

Since CERN has a copy of every file, it does not appear in the quota.

In practice, the absolute values are meaningless, what matters is their relative values. The total is normalized to a 100 in the code.

When choosing where a run will be copied, we look at the current status of the distribution, based on the run duration. The site which is the furthest from its objectives is selected.

2.5.2 Selection of a Tier1 for the data processing

Once a Tier1 has been selected to copy the RAW file, one needs to select a site where the data will be processed: either CERN or the Tier1 where the data is: the *RunDestination*. Note that the destination is chosen per Run, and will stay as is: all the production will process the run at the same location.

This is done using *Operations/<Setup>/Shares/CPUforRAW*. There, the values are independent: they should be between 0 and 1, and represents the fraction of data it will process compared to CERN. So if the value is 0.8, it means 80% of the data copied to that site will be processed at that site, and the 20 other percent at CERN.

This share is used by the processing plugin *DataProcessing*. The equivalent exists when reprocessing (plugin *DataReprocessing*): *Operations/<Setup>/Shares/CPUforReprocessing*

2.5.3 Change of values in the shares

Note: if a change is to be made after a transformation has already distributed a lot of files, it is better to start a new transformation.

The principle goes as follow, but is obviously better done with an Excel sheet.

From Rebus (https://gstat-wlcg.cern.ch/apps/pledges/resources/), we take for each T1 the CPUPledge (in MHS06) and the TapePledge (PB). We deduce easily the CPUPledgePercent and TapePledgePercent.

From the StorageUsageSummary, we get the CurrentTapeUsage (e.g. dirac-dms-storage-usage-summary –LCG –Site LCG.CERN.cern)

We then have:

```
AdditionalTape = TapePledge - CurrentTape
```

From which we deduce AdditionalTapePercent.

We then compute the ratio:

```
CPU / NewTape = CPUPledgePercent / AdditionalTapePercent
```

It represents the increase of CPU pledge vs the increase of Tape with respect to the total.

We then chose a certain percentage of data which is going to be processed at CERN. Say 20%. We then get:

```
CPUShare = CPUPledgePercent*(1-0.2)
```

The next step is to assign a CPUFraction (in [0:1]) by hand following this guideline: the lower the CPU/Tape ratio, the lower the fraction processed "locally".

The final step is to compute:

```
RAWShare = CPUShare/CPUFraction
```

It represents the percentage of data to be copied to the given T1.

Obviously, since we have an extra constraint, we have to give a degree of freedom. We normally give it to RAL with the following:

```
RALRAWShare = 100% - Sum(OtherShares)
RALCPUFraction = RALCpuShare / RALRAWShare
```

CPUShare corresponds to *Operations/<Setup>/Shares/CPUforRAW*

RAWShare corresponds to *Operations/<Setup>/Shares/RAW*

2.6 Productions flushing

2.6.1 Flushing a transformation

Transformations normally have grouping factors: total size of the input files, number of files, etc. There are cases when the grouping conditions cannot be reached, for example if there are not enough files in the run to reach the threshold defined. In that case, the transformation can be *flushed*, meaning create tasks anyway with whatever is there.

The flushing is a manual operation that only has an impact on the files present at the moment of triggering it, meaning that if new files arrive later, they will accumulate again: a transformation does not stay in "flush mode".

2.6.2 Flushing a run in a transformation

Many transformations have a grouping by Run on top of a running by size/files. The same as described previously can happen: within a given run, the grouping conditions cannot be reached. In that case, it is possible to flush the run. There are two major differences compared to flushing a transformation:

1. Flushing a run is definitive
2. The procedure *can* be automatic

1. Flushing a run is definitive

Once a run is set to flush, it will stay in this state. This means that if new files arrive after flushing the run, they will not be accumulated, and a new task will be create for each and every file that arrives. This is not what you want normally.

2. Automatic run flushing for Merging

The principle always consists in going back to the RAW files of a run, and making sure that all of them have descendants in the current production. In practice, we count the number of RAW ancestors of the files in the production, and compare it with the number of RAW files declared in the BKK. These two numbers must match. This count is done by stream.

The only runs that are considered for flushing are the runs marked as 'finished' in the bookkeeping.

However, it might happen that a run does not get flushed. This normally shows an issue at the Stripping level. Consider the following example, with a Run that contains 3 raw files:

RAW file	RDST file	Stripping output
A.RAW	A.RDST	A.stream1, A.stream2, A.stream3
B.RAW	B.RDST	B.stream2, B.stream3, B.stream4
C.RAW	C.RDST	C.stream3, c.stream4

So, when looking at the ancestors per stream, we find:

Stream	Nb of RAW ancestors
stream1	1
stream2	2
stream3	3
stream4	2

In that case, the flushing of the run will be triggered by stream3, since it finds the 3 ancestors. However, if in the stripping production, one file is never stripped because problematic, no stream will ever have all the raw files as ancestors, and the run will never be flushed. Hence, the run status in the merging is a good way to check the stripping :-)

Note that the script transformation-debug is more clever that the plugin, and can warn of such situations.

CHAPTER 3

Certification

3.1 LHCbDIRAC Certification (development) Releases

The following procedure applies to pre-releases (AKA certification releases) and it is a simpler version of what applies to production releases.

This page details the duty of the release manager. The certification manager duties are detailed in the next page.

3.1.1 What for

The *release manager* of LHCbDIRAC has the role of:

1. creating the pre-release
2. making basic tests
3. deploying it in the certification setup

The *certification manager* would then follow-up on this by: 4. making even more tests

And, after several iterations of the above, before: 5. merging in the production branch

Points 4 and 5 won't anyway be part of this first document.

3.1.2 1. Creating the release

Unless otherwise specified, certification releases of LHCbDIRAC are done "on top" of the latest pre-release of DIRAC. The following of this guide assumes the above is true.

Creating a pre-release of LHCbDIRAC means creating a tarball that contains the code to certify. This is done in 2 steps:

1. Merging "Merge Requests"
2. Creating the release tarball, add uploading it to the LHCb web service

But before:

Pre

If you use a version of git prior to 1.8, remove teh option *–pretty* in the command line

Verify what is the last tag of DIRAC:

```
# it should be in this list:
git describe --tags $(git rev-list --tags --max-count=10)
```

A tarball containing it is should be already uploaded here

You may also look inside the .cfg file for the DIRAC release you're looking for: it will contain an "Externals" version number, that should also be a tarball uploaded in the same location as above.

If all the above is ok, we can start creating the LHCbDIRAC pre-release.

Merging "Merge Requests"

Merge Requests (MR) that are targeted to the devel branch and that have been approved by a reviewer are ready to be merged

If there are no MRs, or none ready: please skip to the "update the CHANGELOG" subsection.

Otherwise, simply click the "Accept merge request" button for each of them.

Then, from the LHCbDIRAC local fork you need to update some files:

```
# if you start from scratch otherwise skip the first 2 commands
mkdir $(date +20%y%m%d) && cd $(date +20%y%m%d)
git clone https://:@gitlab.cern.ch:8443/lhcb-dirac/LHCbDIRAC.git
git remote add upstream https://:@gitlab.cern.ch:8443/lhcb-dirac/LHCbDIRAC.git
# update your "local" upstream/master branch
git fetch upstream
# create a "newDevel" branch which from the upstream/devel branch
git checkout -b newDevel upstream/devel
# determine the tag you're going to create by checking what was the last one from the␣
↪following list (add 1 to the "p"):
git describe --tags $(git rev-list --tags --max-count=5)
# Update the version in the __init__ file:
vim LHCbDIRAC/__init__.py
# Update the version in the releases.cfg file:
vim LHCbDIRAC/releases.cfg
# For updating the CHANGELOG, get what's changed since the last tag
#please use the proper LHCbDIRAC tag; replace v8r2p46
git log --pretty=oneline ${t}..HEAD | grep -Ev "($(git log --pretty=oneline  ${t}..
↪v8r2p46 | awk {'print $1'} | tr '\n' '|')BOOM)"
# copy the output, add it to the CHANGELOG (please also add the DIRAC version)
vim CHANGELOG # please, remove comments like "fix" or "pylint" or "typo"...
#If needed, change the versions of the packages
vim dist-tools/projectConfig.json
# Commit in your local newDevel branch the 3 files you modified
git add -A && git commit -av -m "<YourNewTag>"
```

Time to tag and push:

```
# make the tag
git tag -a <YourNewTag> -m <YourNewTag>
# push "newDevel" to upstream/devel
git push --tags upstream newDevel:devel
# delete your local newDevel
git branch -d newDevel
```

Remember: you can use "git status" at any point in time to make sure what's the current status.

Creating the release tarball, add uploading it to the LHCb web service

Login on lxplus, run

```
lb-run LHCbDirac/prod bash -norc
git archive --remote ssh://git@gitlab.cern.ch:7999/lhcb-dirac/LHCbDIRAC.git devel␣
↪LHCbDIRAC/releases.cfg  | tar -x -v -f - --transform 's|^LHCbDIRAC/||' LHCbDIRAC/
↪releases.cfg
dirac-distribution -r v8r4-pre1 -l LHCb -C file:///`pwd`/releases.cfg (this may take␣
↪some time)
```

Don't forget to read the last line of the previous command to copy the generated files at the right place. The format is something like:

```
( cd /tmp/joel/tmpxg8UuvDiracDist ; tar -cf - *.tar.gz *.md5 *.cfg ) | ssh
↪$USER@lxplus.cern.ch 'cd /afs/cern.ch/lhcb/distribution/DIRAC3/tars &&  tar -xvf - &
↪& ls *.tar.gz > tars.list'
```

And just copy/paste/execute it.

3.1.3 2. Making basic verifications

Once the tarball is done and uploaded, the release manager is asked to make basic verifications, via Jenkins, if the release has been correctly created.

The tests may vary, but are announced on the Trello board, and on the Slack channel 'lhcb-certification' of the 'lhcb-dirac' team.

3.1.4 3. Deploying the release

Deploying a release means deploying it for some installation:

```
* client
* server
* pilot
```

release for client

Please refer to this TWIKI page a quick test to validate the installation is to run the SHELL script $LHCBRELEASE/LHCBDIRAC/LHCBDIRAC_vXrY/LHCbDiracSys/test/client_test.csh

go to https://jenkins-lhcb-nightlies.web.cern.ch/job/nightly-builds/job/release/build page for asking to install the client release in AFS and CVMFS:

- in the field "Project list" put : "Dirac vNrMpK LHCbGrid vArB LHCbDirac vArBpC " (LHCbGrid version can be found: https://gitlab.cern.ch/lhcb-dirac/LHCbDIRAC/blob/devel/dist-tools/projectConfig.json)
- in the field "platforms" put : "x86_64-slc6-gcc48-opt x86_64-slc6-gcc49-opt x86_64-slc6-gcc62-opt x86_64-centos7-gcc62-opt"

Then click on the "BUILD" button

- within 10-15 min the build should start to appear in the nightlies page https://lhcb-nightlies.cern.ch/release/
- if there is a problem in the build, it can be re-started via the dedicated button (it will not restart by itself after a retag)

When the release is finished https://lhcb-nightlies.cern.ch/release/, you can deploy to the client to afs dev area or prod.

prod area

If you want to deploy this release to production release area, you have to create a JIRA task and make the request via https://sft.its.cern.ch/jira/browse/LHCBDEP.

- **NOTE: If some package is already released, please do not indicate in the Jira task. For example: a Jira task when:**
  - DIRAC is not released, then the message in the JIRA task: Summary:Dirac v6r14p37 and LHCbDirac v8r2p50; Description: Please release Dirac and LHCbDirac in this order based on build 1526;
  - DIRAC is released, then the message in the JIRA task: Summary:LHCbDirac v8r2p50; Description: Please release LHCbDirac based on build 1526;

afs deve area

Note: Please execute the following commands sequentially.

The following commands used to prepare the RPMs:

```
ssh lhcb-archive
export build_id=1520
lb-release-rpm /data/artifacts/release/lhcb-release/$build_id
lb-release-rpm --copy /data/artifacts/release/lhcb-release/$build_id
```

If the rmps are created, you can deploy the release (Do not execute parallel the following commands):

```
ssh lxplus
cd /afs/cern.ch/lhcb/software/lhcb_rpm_dev
export MYSITEROOT=/afs/cern.ch/lhcb/software/lhcb_rpm_dev
export MyProject=Dirac
export MyVersion=vArBpC
./lbpkr rpm -- -ivh --nodeps /afs/cern.ch/lhcb/distribution/rpm/lhcb/${MyProject^^}_$
↪{MyVersion}*
export MyProject=LHCbDirac
export MyVersion=vArB-preC
./lbpkr rpm -- -ivh --nodeps /afs/cern.ch/lhcb/distribution/rpm/lhcb/${MyProject^^}_$
↪{MyVersion}*
```

Server

To install it on the VOBOXes (certification only) from lxplus:

```
lhcb-proxy-init  -g diracAdmin
dirac-admin-sysadmin-cli --host volhcbXX.cern.ch
>update LHCbDIRAC-v8r4-pre1
>restart *
```

The (better) alternative is using the web portal.

Pilot

Use the following script (from, e.g., lxplus after having run *lb-run –dev LHCbDIRAC bash*):

```
dirac-pilot-version
```

for checking and updating the pilot version. Note that you'll need a proxy that can write in the CS (i.e. lhcb-admin). This script will make sure that the pilot version is update BOTH in the CS and in the json file used by pilots started in the vacuum. The command to update is

dirac-pilot-version -S v8r4-pre1

Make sure that you are in the certification setup (e.g. check the content of your .dirac.cfg file)

3.2 The certification process

Certifying a release is a process. There are a number of steps to make to reach the point in which we can finally say that a release is at production level. Within LHCbDirac, we are trying to streamline and automatize this process as much as possible. Even with that, some tests still require manual intervention. We can split the process in a series of incremental tests.

Within the following sections we describe, step by step, all the actions needed.

The whole certification process varies from release to release. The list of things to do is maintained in trello boards.

3.2.1 Unit test

When a new release candidate is created from the devel branch, we first run pylint on the whole codebase, and all the unit tests. Jenkins automizes this for us.

3.2.2 Integration and Regression tests

Run by Jenkins.

3.2.3 System tests

Even if it should not be considered strictly as a test, running all the agents and service within certification is an action to take. Agents and services spits errors and exceptions. While the second are obviously bugs, the first are not to be considered bugs until an expert look. Nonetheless, we have created a tool to easily identify all new exceptions and errors:

```
codeLocation=https://gitlab.cern.ch/lhcb-dirac/LHCbDIRAC/raw/devel/tests/System/
↪LogsParser/
mkdir /tmp/logTest
cd /tmp/logTest
wget -r -np -nH --cut-dirs=7 $codeLocation
/bin/bash logParser.sh
```

For testing that the RMS works, there is an ad-hoc test:

```
wget http://github.com/DIRACGrid/DIRAC/blob/integration/DataManagementSystem/test/
↪IntegrationFCT.py
python IntegrationFCT.py lhcb_user CERN-USER RAL-USER CNAF-USER
python IntegrationFCT.py lhcb_prod CERN-FAILOVER RAL-FAILOVER CNAF-FALIOVER
```

Those commands will create and put to the Request Management System two new requests:

1. for lhcb_user group, which should be banned from using the FTS system
2. for lhcb_prod or lhcb_prmgr group, which this should be executed using FTS

You could monitor their execution using *Request monitor* web page or by using CLI comamnd:

```
dirac-rms-show-request test<userName>-<userGroup>
```

The execution itself will take a while, but at the end both requests statuses should be set to 'Done'.

Another test, again for the RMS, combined with FTS, is to simply use the following standard DIRAC scripts:

```
dirac-dms-create-replication-request CNAF_MC-DST /lhcb/certification/test/ALLSTREAMS.
↪DST/00000751/0000/00000751_00000014_1.allstreams.dst
```

Which will actually schedule the replication of such file using FTS. This will print an ID that can be used for the script

```
dirac-rms-show-request ID
```

That should show how the request goes (quickly) in status "Scheduled", and then "Done".

The following script, instead, will remove the copy just created.

```
dirac-dms-create-removal-request CNAF_MC-DST /lhcb/certification/test/ALLSTREAMS.DST/
↪00000751/0000/00000751_00000014_1.allstreams.dst
```

Again, monitoring is available as above.

For testing the replications and removals, use the following:

```
dirac-dms-add-replication --BKQuery=/validation/MC11a/Beam3500GeV-2011-MagDown-Nu2-
↪EmNoCuts/Sim05/Trig0x40760037Flagged/Reco12a/Stripping17Flagged/12463412/ALLSTREAMS.
↪DST --Plugin=ReplicateDataset --Test
```

That will just print out how many files can be replicated. If there is at least one file (for this particular query there should be 35), then you can start it with:

```
dirac-dms-add-replication --BKQuery=/validation/MC11a/Beam3500GeV-2011-MagDown-Nu2-
↪EmNoCuts/Sim05/Trig0x40760037Flagged/Reco12a/Stripping17Flagged/12463412/ALLSTREAMS.
↪DST --Plugin=ReplicateDataset --NumberOfReplicas=2 --SecondarySEs Tier1-DST --Start
```

You can monitor the advancement using:

```
dirac-dms-replica-stats --BKQuery=/validation/MC11a/Beam3500GeV-2011-MagDown-Nu2-
↪EmNoCuts/Sim05/Trig0x40760037Flagged/Reco12a/Stripping17Flagged/12463412/ALLSTREAMS.
↪DST
```

Which should tell you the replica statistics, something like:

```
[fstagni@lxplus0032 ~]$ dirac-dms-replica-stats --BKQuery=/validation/MC11a/
↪Beam3500GeV-2011-MagDown-Nu2-EmNoCuts/Sim05/Trig0x40760037Flagged/Reco12a/
↪Stripping17Flagged/12463412/ALLSTREAMS.DST
Executing BK query: {'Visible': 'Yes', 'ConfigName': 'validation',
↪'ConditionDescription': 'Beam3500GeV-2011-MagDown-Nu2-EmNoCuts', 'EventType':
↪'12463412', 'FileType': 'ALLSTREAMS.DST', 'ConfigVersion': 'MC11a', 'ProcessingPass
↪': '/Sim05/Trig0x40760037Flagged/Reco12a/Stripping17Flagged', 'SimulationConditions
↪': 'Beam3500GeV-2011-MagDown-Nu2-EmNoCuts'}

34 files (0.0 TB) in directories:
/lhcb/validation/MC11a/ALLSTREAMS.DST/00000654/0000 34 files
34 files found with replicas

Replica statistics:
0 archives: 0 files
1 archives: 25 files
2 archives: 9 files
0 replicas: 0 files
1 replicas: 0 files
2 replicas: 0 files
3 replicas: 33 files
4 replicas: 0 files
5 replicas: 1 files

SE statistics:
    CERN-ARCHIVE: 15 files
    CNAF-ARCHIVE: 5 files
  GRIDKA-ARCHIVE: 11 files
   IN2P3-ARCHIVE: 1 files
     RAL-ARCHIVE: 8 files
    SARA-ARCHIVE: 3 files
   CERN_MC_M-DST: 34 files
     CNAF_MC-DST: 4 files
   CNAF_MC_M-DST: 8 files
   GRIDKA_MC-DST: 1 files
 GRIDKA_MC_M-DST: 3 files
    IN2P3_MC-DST: 9 files
  IN2P3_MC_M-DST: 6 files
      PIC_MC-DST: 5 files
    PIC_MC_M-DST: 4 files
      RAL_MC-DST: 20 files
    RAL_MC_M-DST: 6 files
     SARA_MC-DST: 3 files
   SARA_MC_M-DST: 1 files

Sites statistics:
     LCG.CERN.ch: 34 files
     LCG.CNAF.it: 12 files
   LCG.GRIDKA.de: 4 files
    LCG.IN2P3.fr: 15 files
      LCG.PIC.es: 9 files
```

(continues on next page)

(continued from previous page)

```
      LCG.RAL.uk: 26 files
     LCG.SARA.nl: 4 files
```

Later, when you see that at least 2 replicas exist, you can issue

```
dirac-dms-add-replication --BKQuery=/validation/MC11a/Beam3500GeV-2011-MagDown-Nu2-
↪EmNoCuts/Sim05/Trig0x40760037Flagged/Reco12a/Stripping17Flagged/12463412/ALLSTREAMS.
↪DST --Plugin=DeleteReplicas --NumberOfReplicas=1 --Start
```

3.3 Acceptance test steps

3.3.1 Installation of LHCbDirac

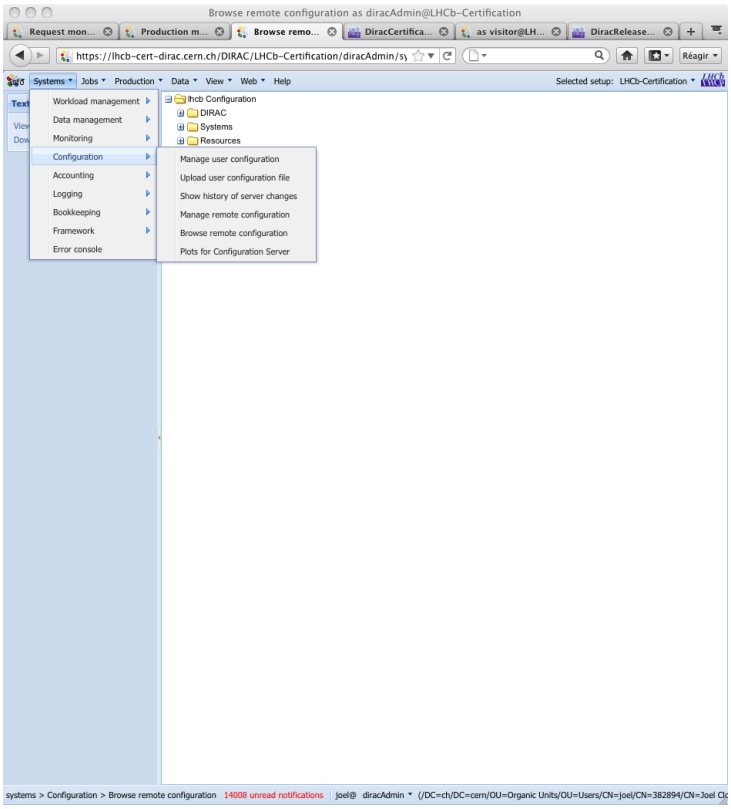
Login to a machine where LHCbDirac is already installed. Set the LHCbDirac environment, get a proxy with admin rights and launch the sysadmin CLI

```
lb-run LHCbDirac/prod bash
lhcb-proxy-init -g diracAdmin
dirac-admin-sysadmin-cli
```

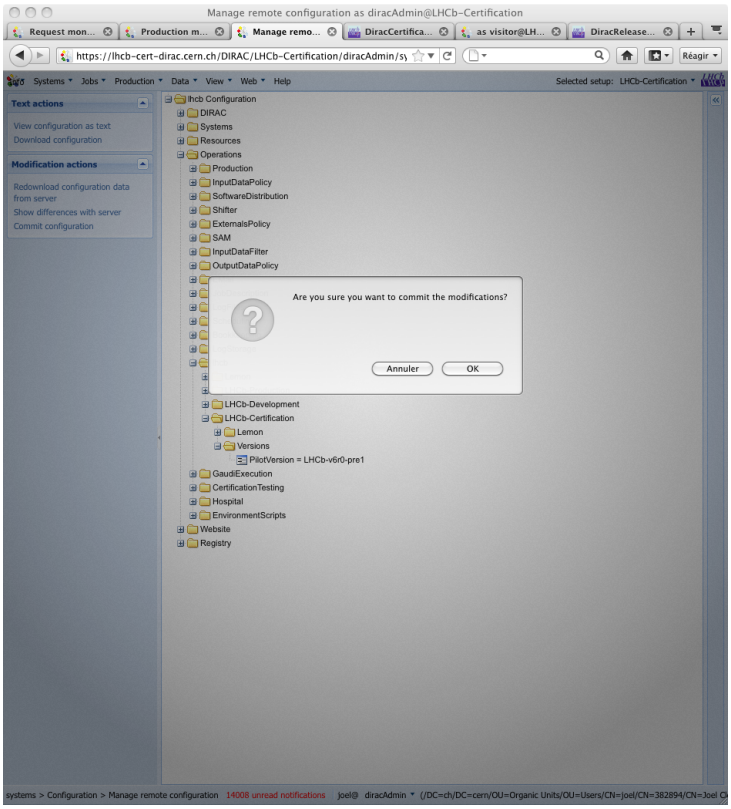
Update the LHCbDirac version and restart all the services

```
set host volhcbXX.cern.ch
update LHCb-vArBpC
restart *
```

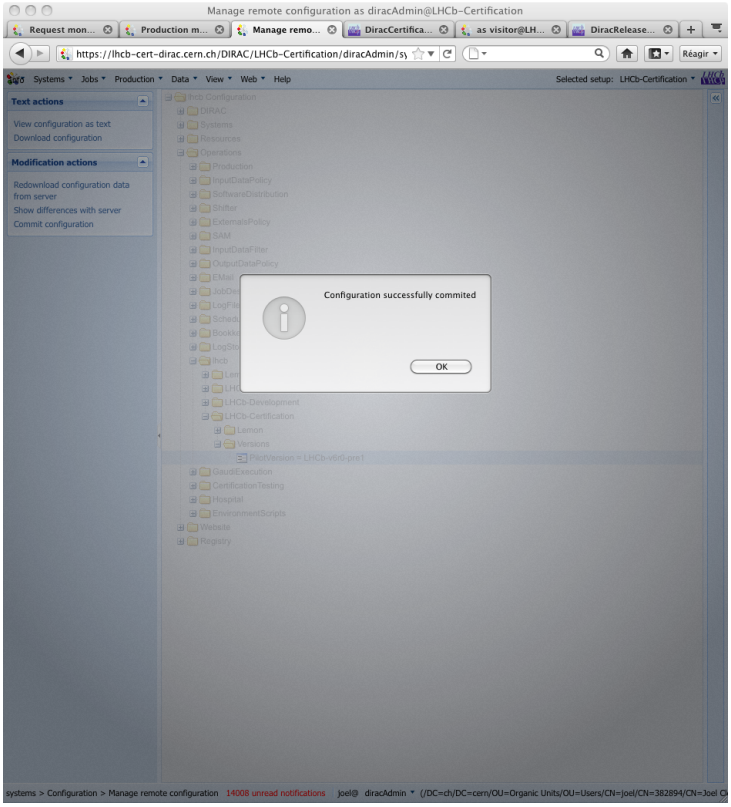
Change the version of the pilot in the CS. Go to the web portal, login with your certificate and the role **diracAdmin**. Click on **Systems**, **Configuration** and **Manage Remote configuration**.



The version is in the section /Operations/lhcb/LHCb-Certification/Versions/PilotVersion. Clicks on the **PilotVersion** and on change option value. Once you have changed the version number, click on **submit**. and do not forget to commit the change.



So you click on the left column on **Commit Configuration**

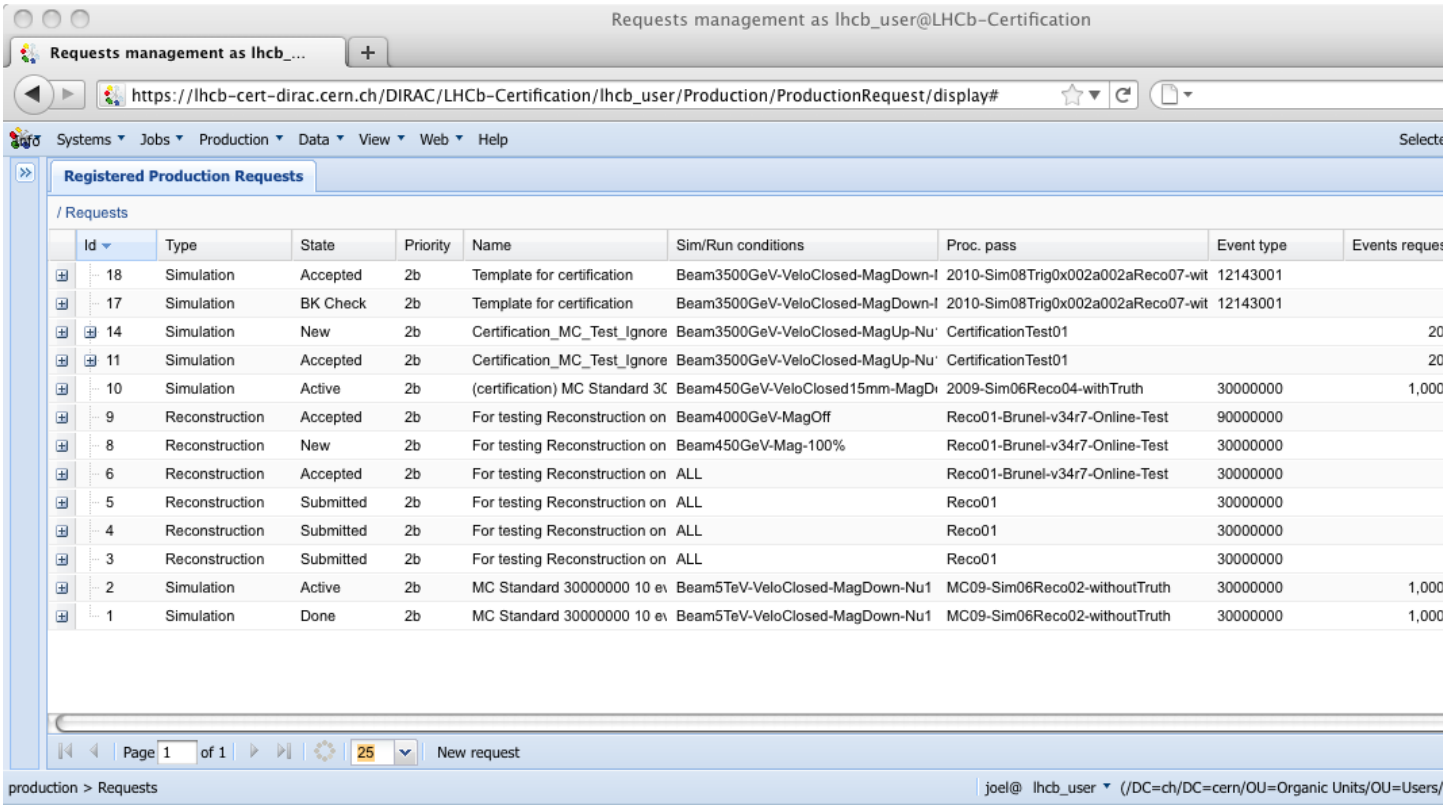


Now you should restart the task queue director

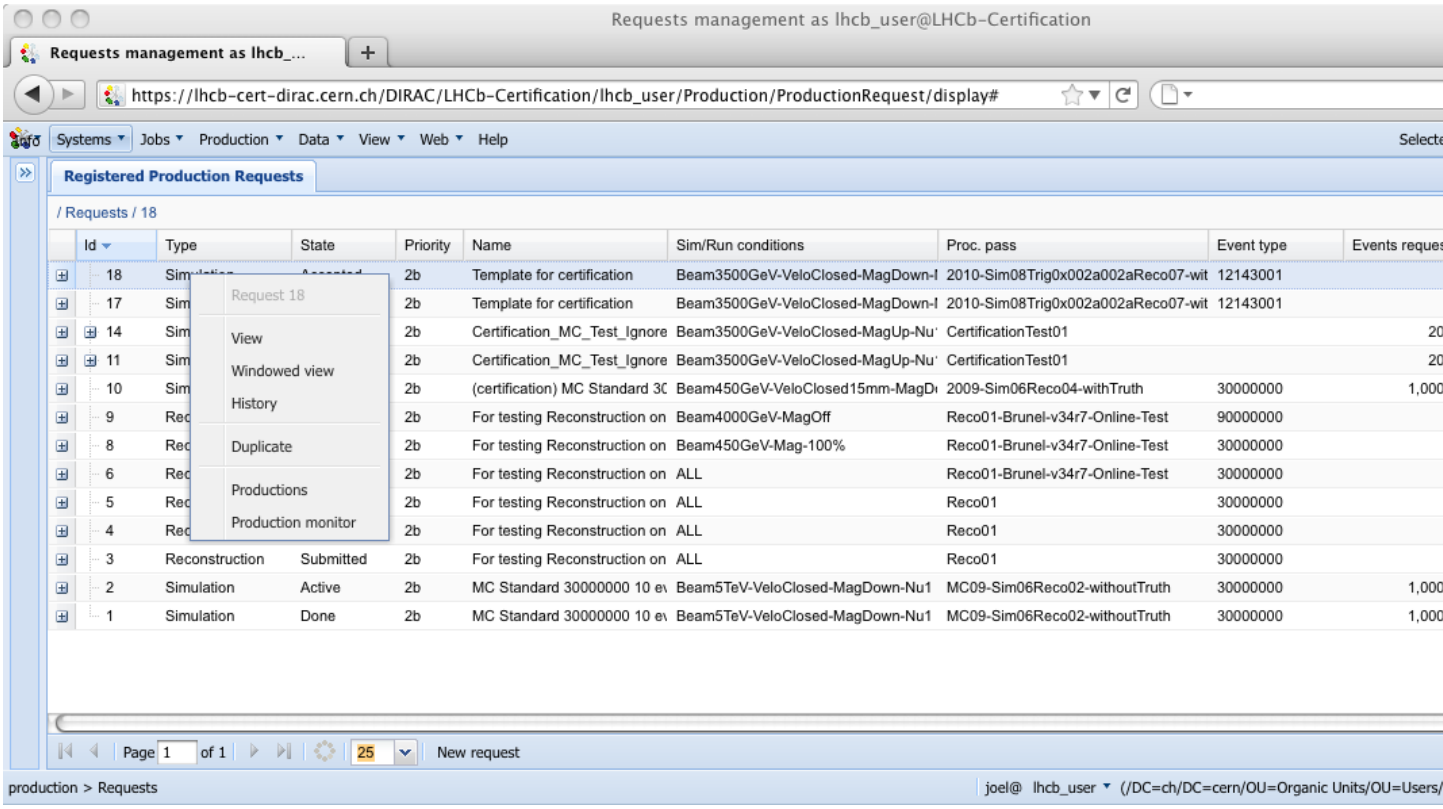
```
cd /opt/dirac/runit
runsvctrl d WorkloadManagement/TaskQueueDirector
runsvctrl u WorkloadManagement/TaskQueueDirector
```

3.3.2 Production test activity

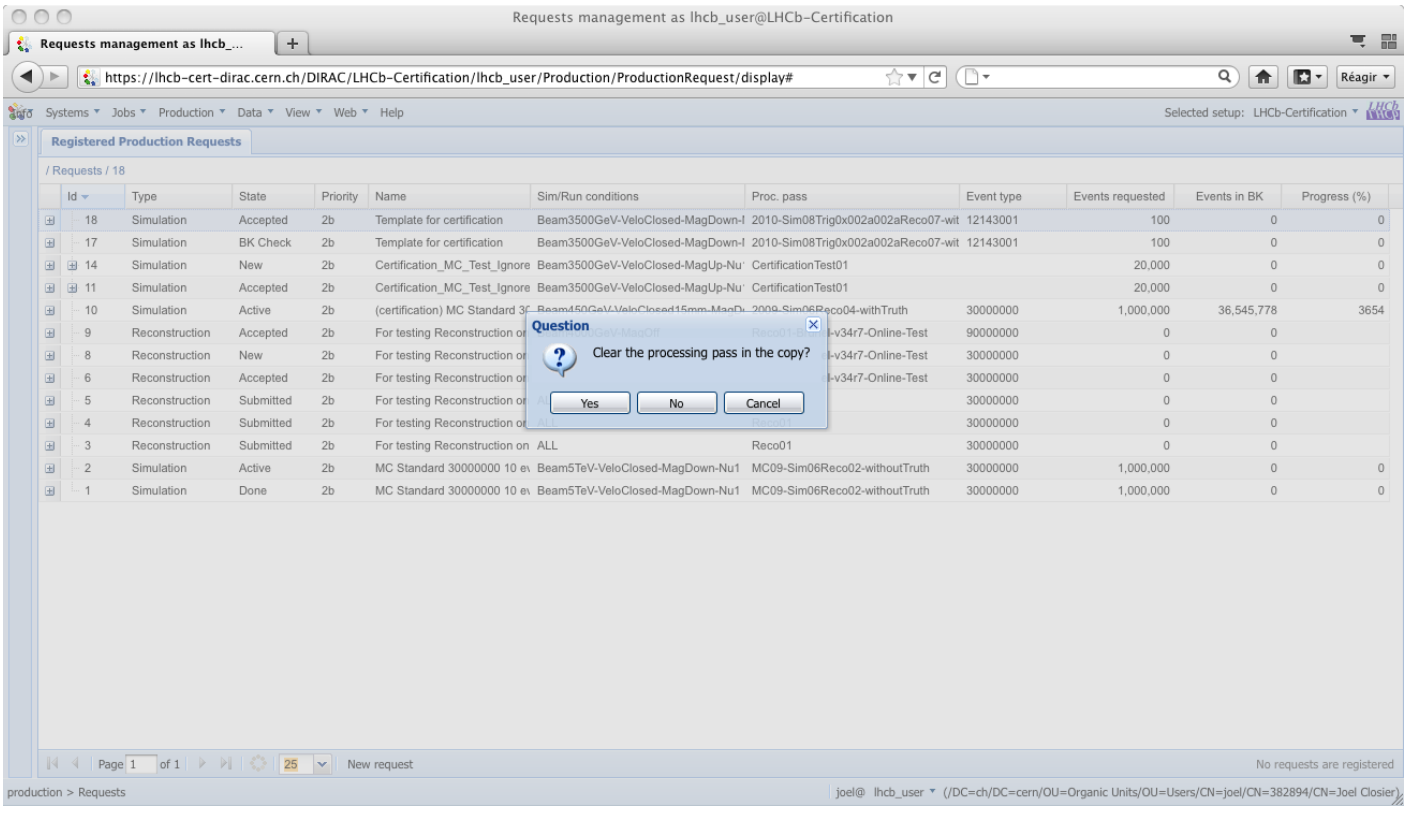
Open your browser and connect to the certification instance of the LHCbDirac web portal (http://lhcb-cert-dirac.cern.ch) select the setup **LHCb-Certification** and load your certificate in the portal. Check that that your role is **lhcb_user**. Go to the tab **Production** and click on the **Requests** choice



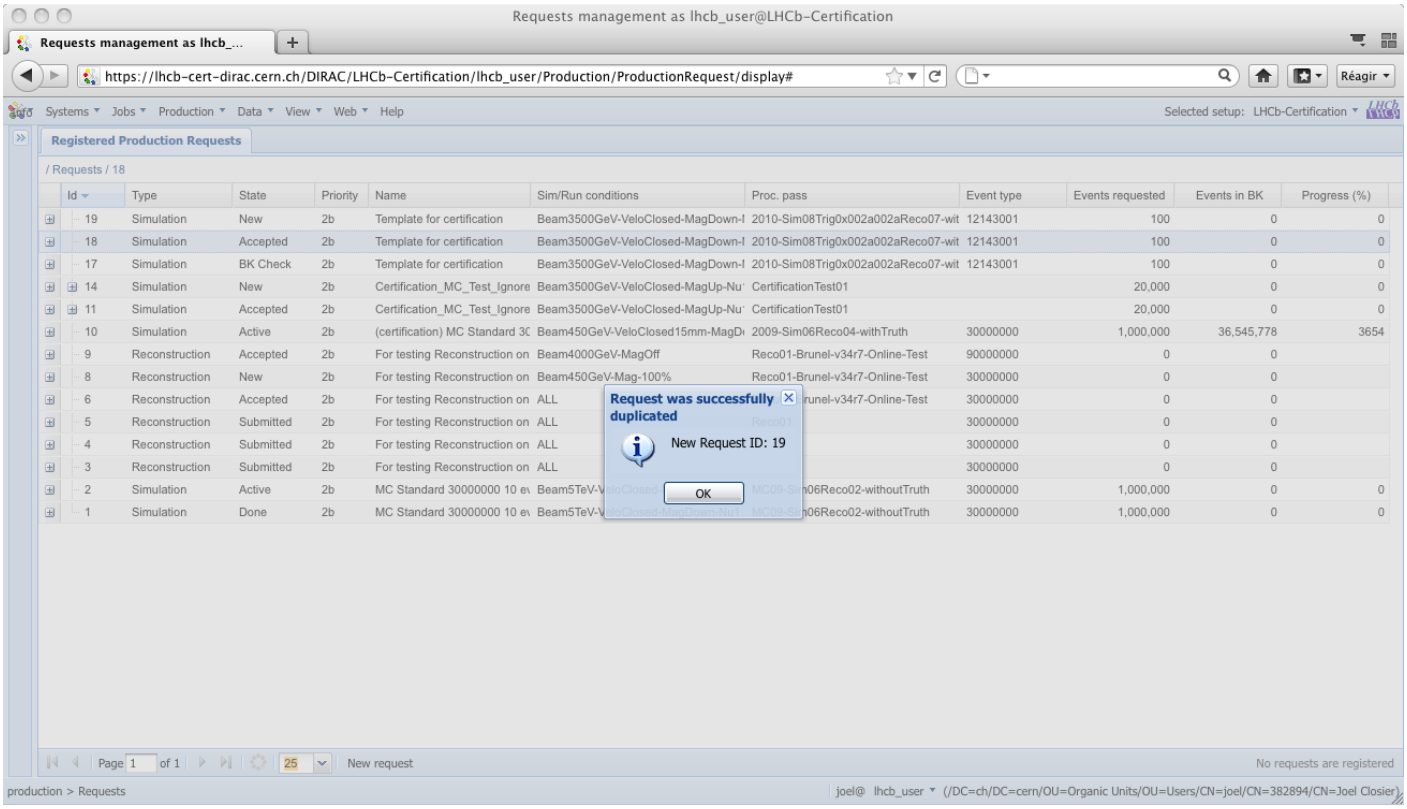
Click on the production which is defined label “template for certification” (nb = 28) and in the menu which appears select **Duplicate**



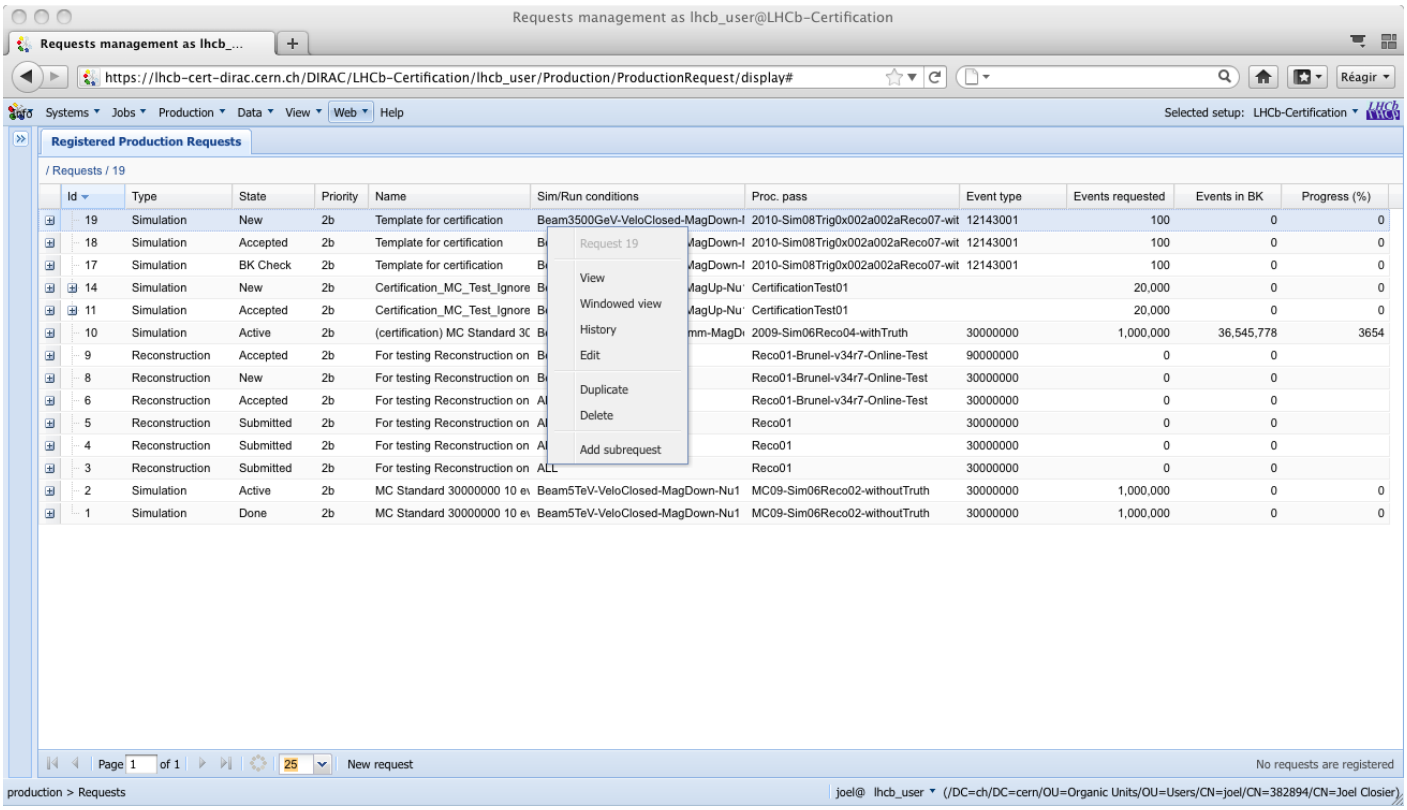
You are ask if you want to **Clear the processing pass in the copy**. Select **No**. This will keep all the steps which are pre-defined.



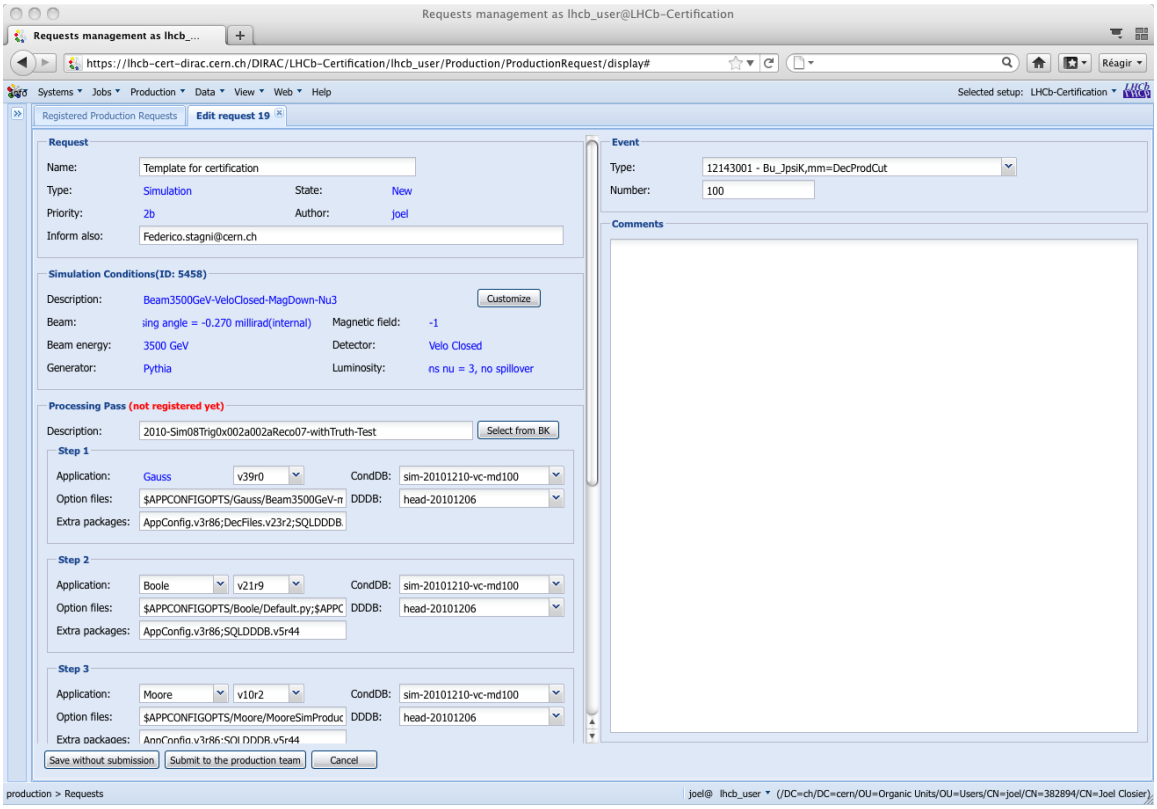
The new request is created and you get a number that will appear in the web page.



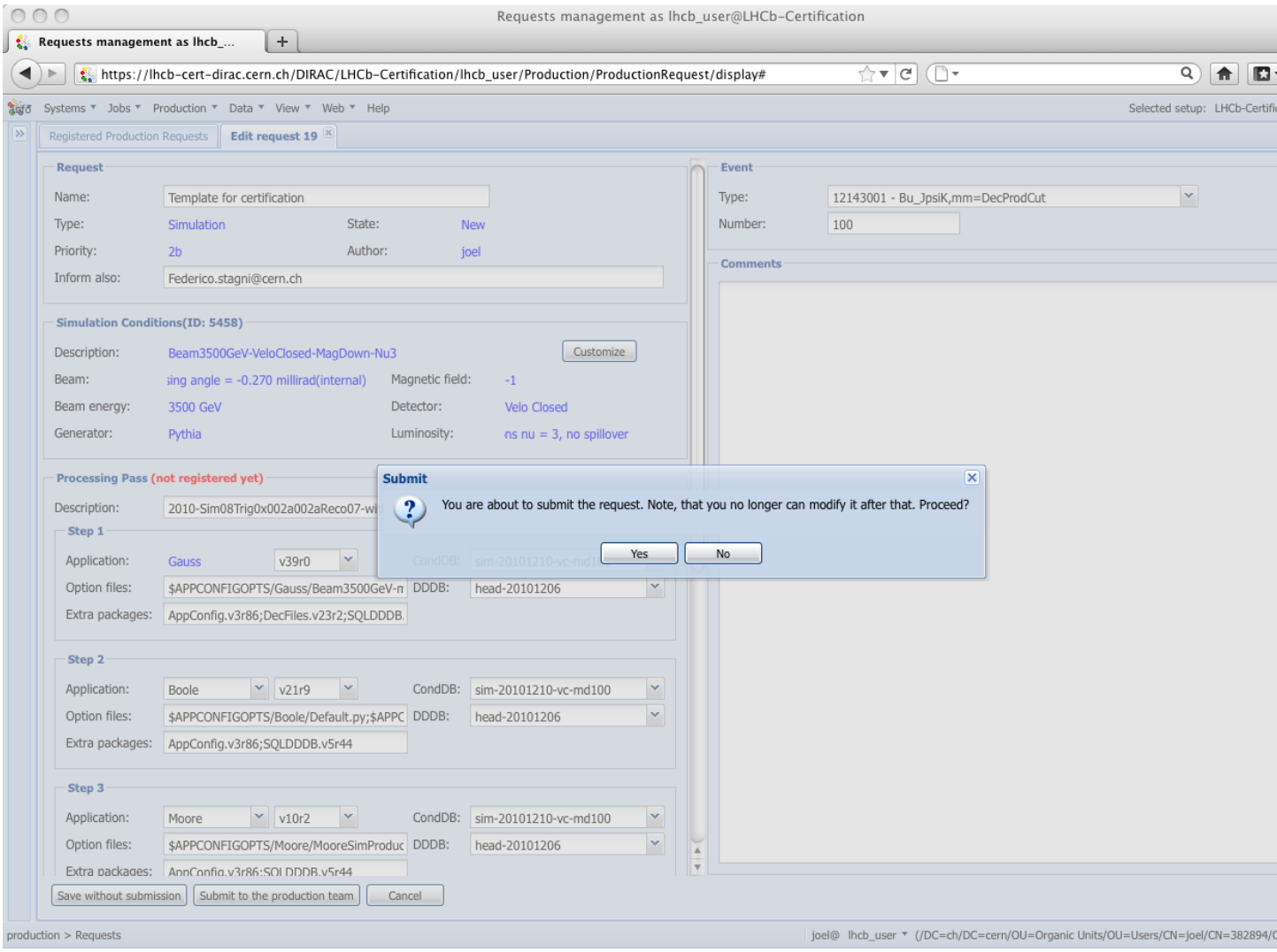
Click on the new request that you just created the step below and select the **edit** option



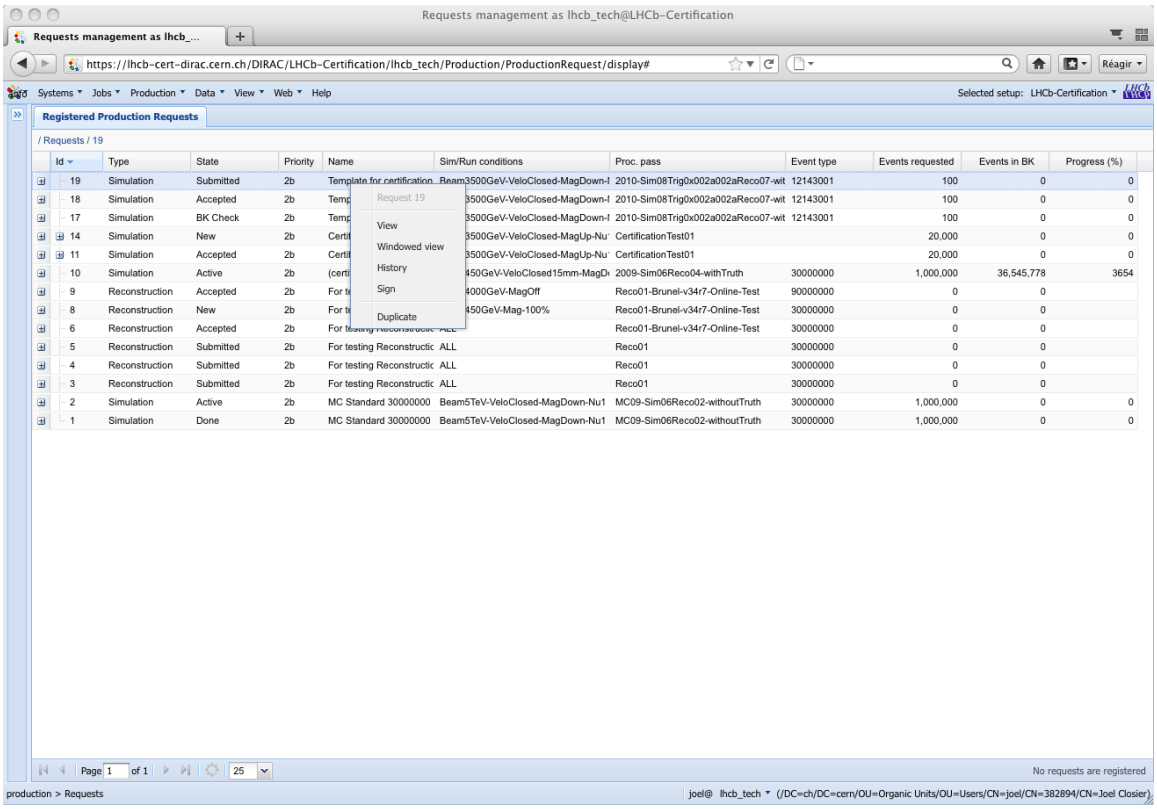
Then modify all the fields which needs a new value. Once you have finished, submit your request to the production team.

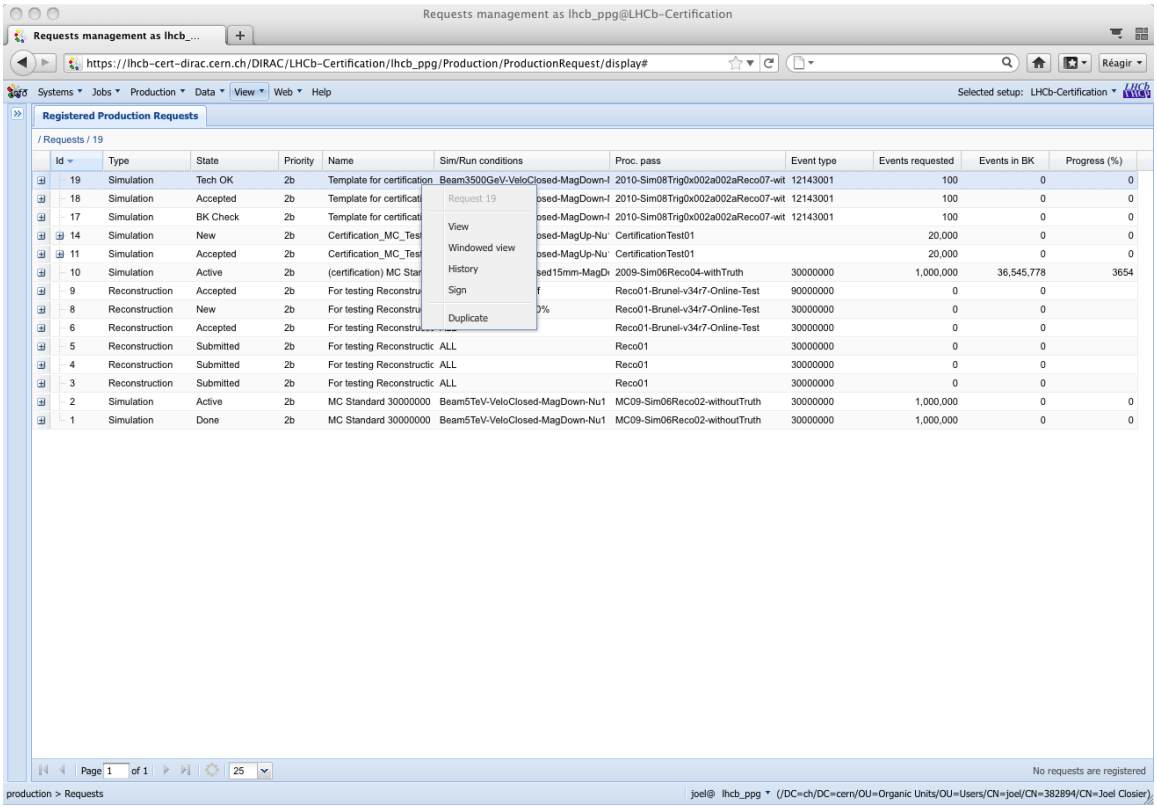


You have just to approve it.

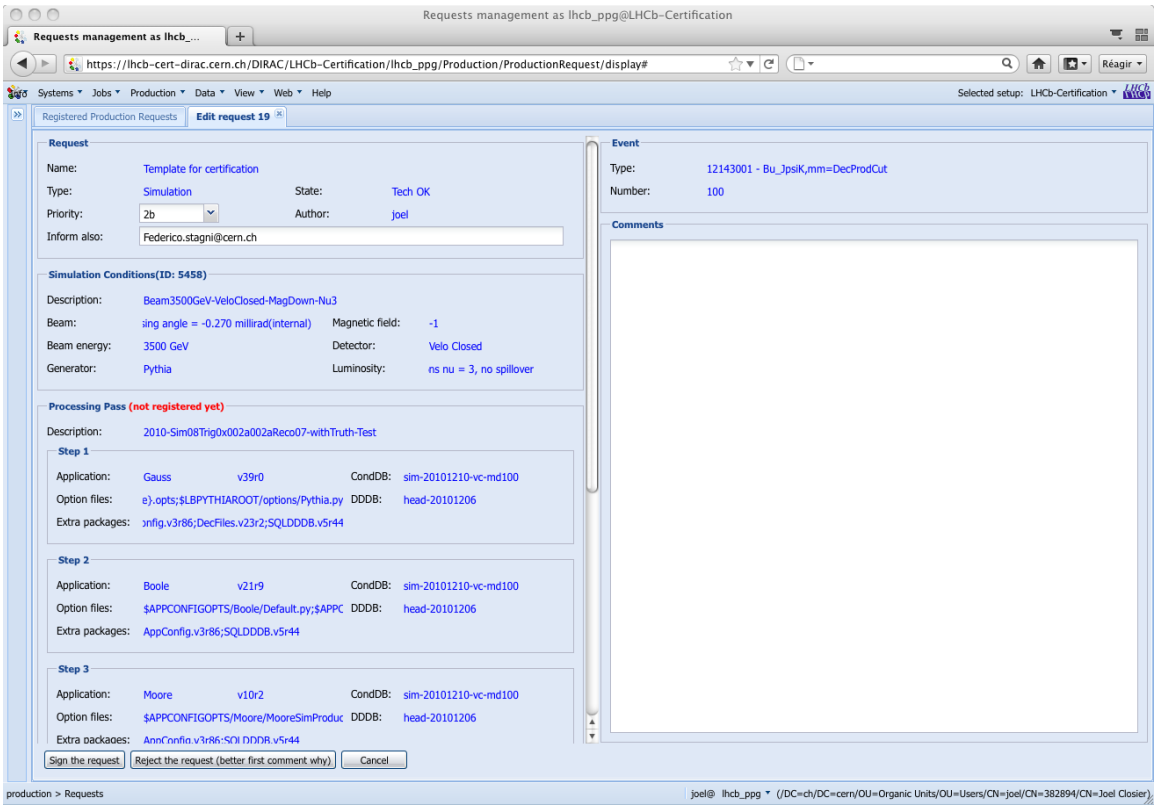


Now you should change your role to become **lhcb_tech** and **lhcb_ppg** to validate the request. You click on the new request and in the menu you choose the option *sign*

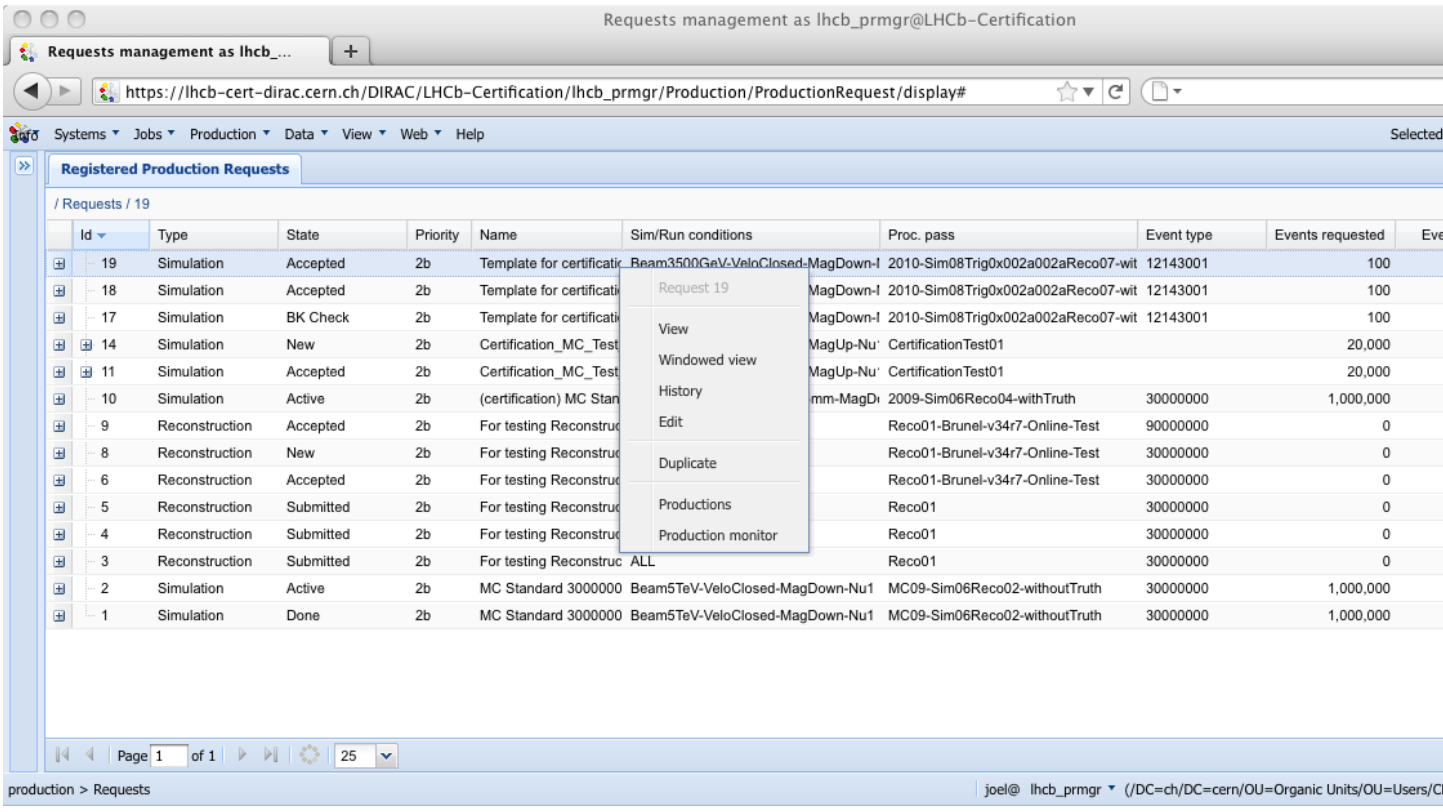




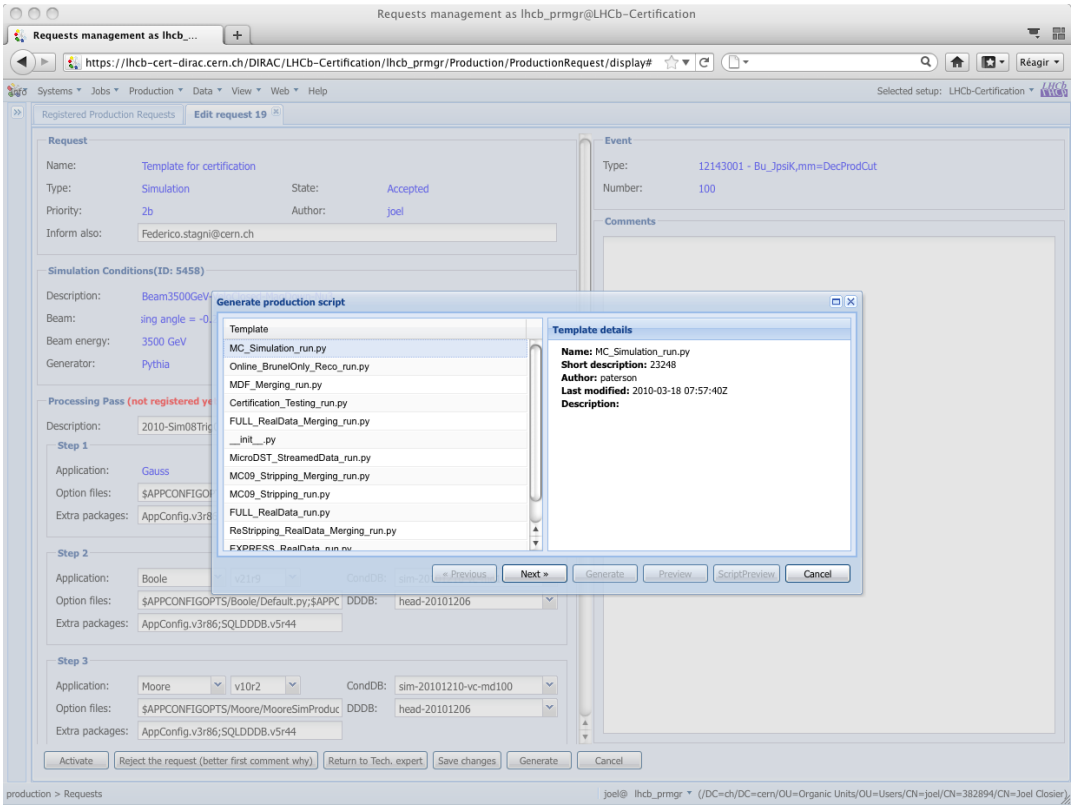
You can sign or reject the request.



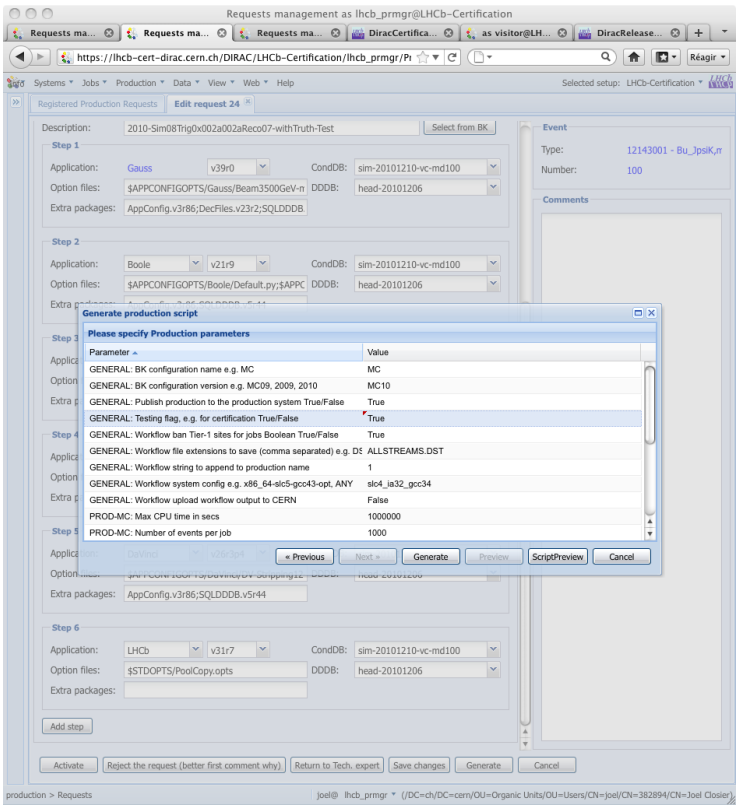
Once the request has been accepted by lhcb_ppg and lhcb_tech, the status become **accepted**. Choose now the role **lhcb_pmgr** and click on the request. Then choose the option *edit*



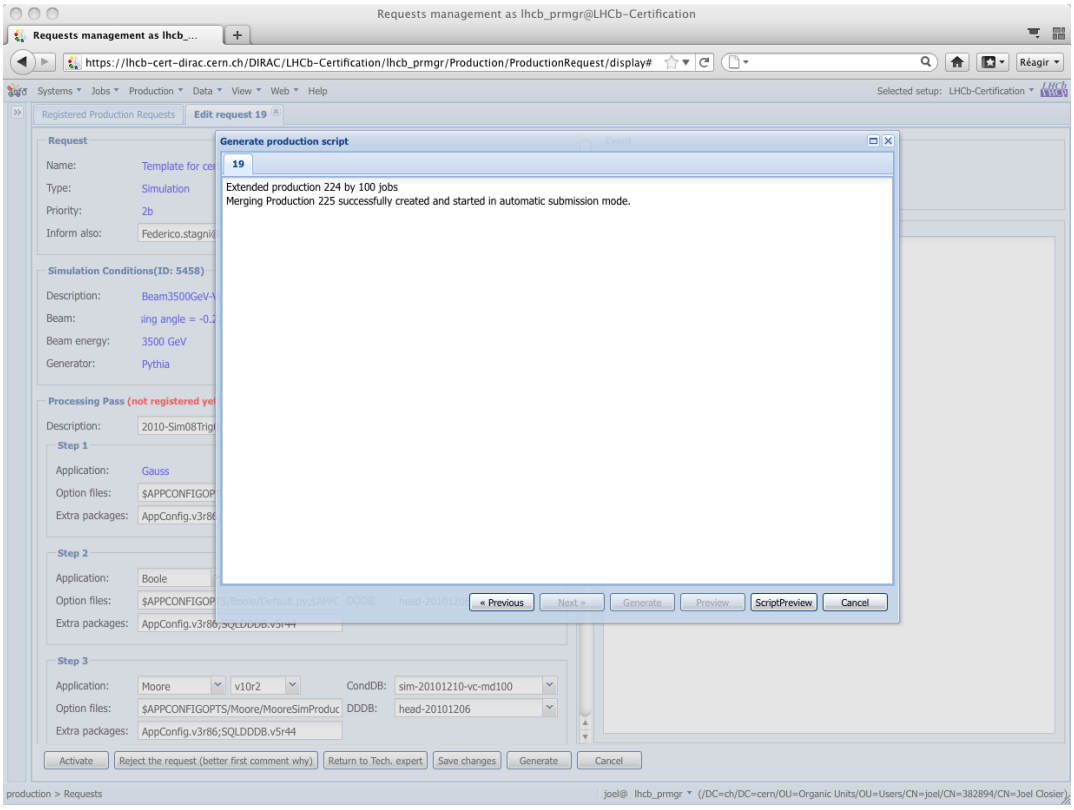
You give the correct Event Type and number of Events. Then you click on **Generate** At this stage you are asked to choose which template should be used. In our case we will choose "MC_Simulation_run.py" and click on the **next** button.



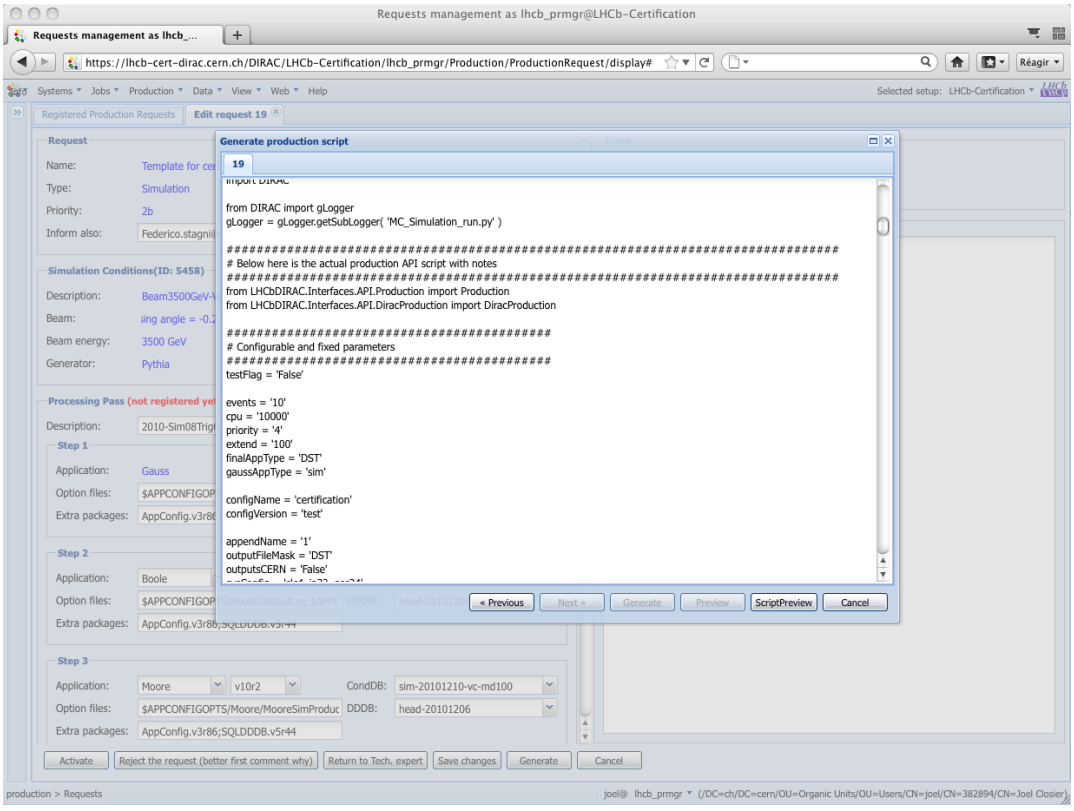
You get now the list of value that you could change before submitting the production. For the certification purpose you should change the value for "MC configuratioon name" to be **certification**, the "configuration version" should be **test**. Verify which plugin you want to use, the number of event that you want to process, the cputimelimit,... Once you have finished, click on the **generate** button.



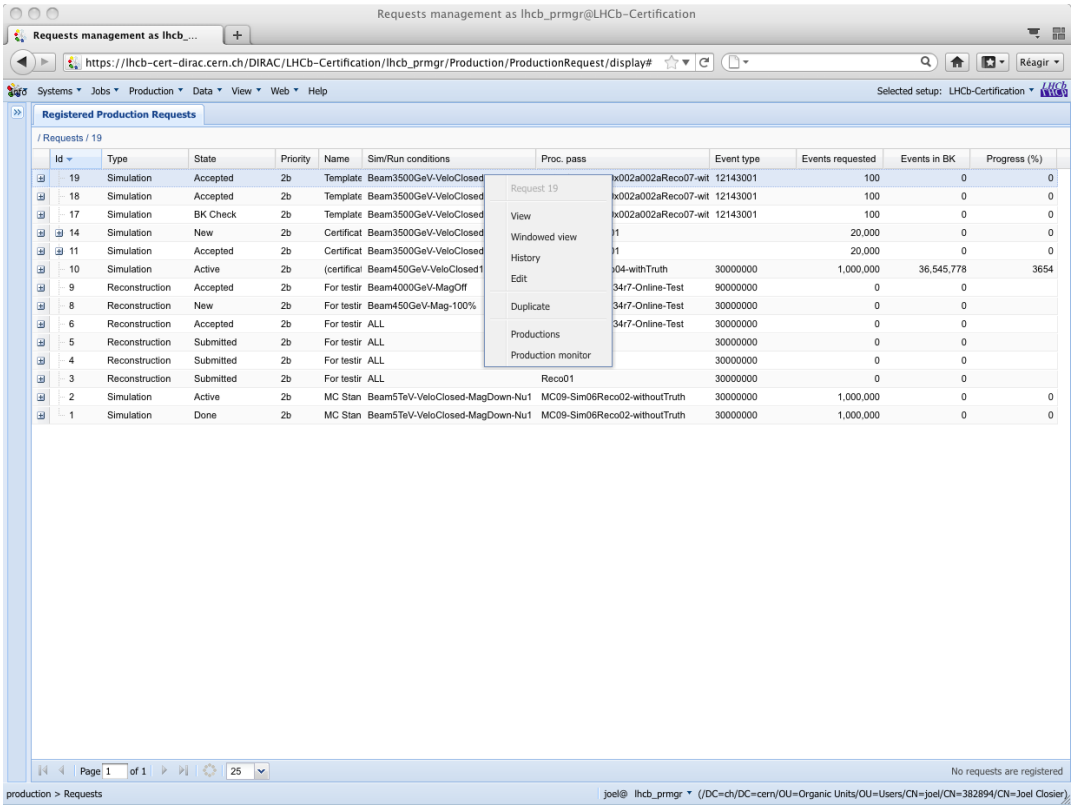
After the generation of the production you will get in a new window the production ID and the number of jobs generated. If you want you can see and save the script which will generate this production by clicking on the **script preview** button.



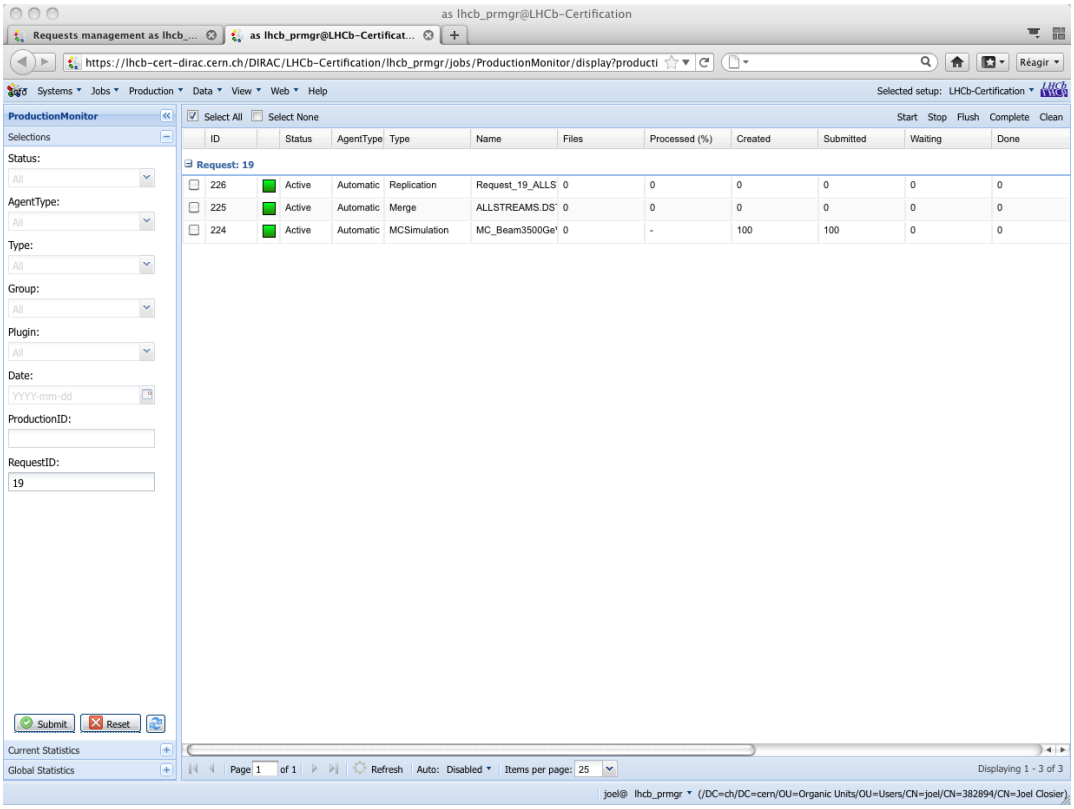
This is the window of the python script which could be used to generate again the production. To exit thi swindow click on **cancel**



If you click on the request and you choose **production monitor** you will be re-direct to the production monitor.



Production monitor with the fresh generated productions.



dirac-bookkeeping-production-informations 830 -o /DIRAC/Setup=LHCb-Certification

```
lxplus448] x86_64-slc5-gcc46-opt /afs/cern.ch/user/j/joel> dirac-bookkeeping-
↪production-informations 830 -o /DIRAC/Setup=LHCb-Certification
Production Info:
Configuration Name: LHCb
Configuration Version: Collision11
Event type: 91000000
-----------------------
StepName: merging MDF
ApplicationName    : mergeMDF
ApplicationVersion : None
OptionFiles        : None
DDDB                : None
CONDDB             : None
ExtraPackages      :None
-----------------------
Number of Steps   1
Total number of files: 2
     LOG:1
     RAW:1
Number of events
File Type           Number of events    Event Type          EventInputStat
RAW                 30988               91000000            30988
Path:  /LHCb/Collision11/Beam3500GeV-VeloClosed-MagDown/Real Data/Merging
/LHCb/Collision11/Beam3500GeV-VeloClosed-MagDown/Real Data/Merging/91000000/RAW
```

You can then check the produced files:

```
nsls -l /castor/cern.ch/grid/lhcb/certification/test/ALLSTREAMS.DST/00000225/0000
dirac-dms-lfn-replicas /lhcb/certification/test/ALLSTREAMS.DST/00000225/0000/00000225_
↪00000001_1.allstreams.dst
dirac-dms-add-replication --Production 259:268 --FileType RADIATIVE.DST --Plugin␣
↪LHCbMCDSTBroadcastRandom --Request 30
dirac-dms-add-replication --Production 239 --FileType ALLSTREAMS.DST --Plugin␣
↪LHCbMCDSTBroadcastRandom --Request 29
Transformation 273 created
Name: Replication-ALLSTREAMS.DST-239-Request29 , Description:␣
↪LHCbMCDSTBroadcastRandom of ALLSTREAMS.DST for productions 239
BK Query: {'FileType': ['ALLSTREAMS.DST'], 'ProductionID': ['239'], 'Visibility': 'Yes
↪'}
3 files found for that query
Plugin: LHCbMCDSTBroadcastRandom
RequestID: 29
[lxplus433] x86_64-slc5-gcc43-opt /afs/cern.ch/lhcb/software/DEV/LHCBDIRAC/LHCBDIRAC_
↪v6r0-pre12> dirac-bookkeeping-production-informations 239Production Info::
    Configuration Name: certification
    Configuration Version: test
    Event type: 12143001

 StepName: MCMerging10
    ApplicationName    : LHCb
    ApplicationVersion : v31r7
    OptionFiles        : $STDOPTS/PoolCopy.opts
    DDB                : head-20101206
    CONDDB             : sim-20101210-vc-md100
    ExtraPackages      :None
```

(continues on next page)

(continued from previous page)

```
Number of Steps   4
Total number of files: 8
         LOG:4
         ALLSTREAMS.DST:4
Number of events
File Type           Number of events    Event Type          EventInputStat
ALLSTREAMS.DST      540                 12143001            540
Path:  /certification/test/Beam3500GeV-VeloClosed-MagDown-Nu3/MC10Sim01-
↪Trig0x002e002aFlagged/Reco08/Stripping12Flagged
/certification/test/Beam3500GeV-VeloClosed-MagDown-Nu3/MC10Sim01-
↪Trig0x002e002aFlagged/Reco08/Stripping12Flagged/12143001/ALLSTREAMS.DST
 dirac-bookkeeping-production-files 239 ALLSTREAMS.DST
FileName
↪                Size       GUID                                      Replica
/lhcb/certification/test/ALLSTREAMS.DST/00000239/0000/00000239_00000044_1.allstreams.
↪dst             14515993   165DD5A9-1D40-E011-AD80-003048F1E1E0     Yes
/lhcb/certification/test/ALLSTREAMS.DST/00000239/0000/00000239_00000045_1.allstreams.
↪dst             2971054    988731FC-1C40-E011-AFCD-90E6BA442F3B     Yes
/lhcb/certification/test/ALLSTREAMS.DST/00000239/0000/00000239_00000074_1.allstreams.
↪dst             202748580  E2BAF0A1-A340-E011-BF97-003048F1B834     Yes
/lhcb/certification/test/ALLSTREAMS.DST/00000239/0000/00000239_00000076_1.allstreams.
↪dst             2804277    F086C525-EB43-E011-96F9-001EC9D8B181     Yes

[lxplus433] x86_64-slc5-gcc43-opt /afs/cern.ch/lhcb/software/DEV/LHCBDIRAC/LHCBDIRAC_
↪v6r0-pre12> dirac-dms-lfn-replicas /lhcb/certification/test/ALLSTREAMS.DST/00000239/
↪0000/00000239_00000044_1.allstreams.dst
{'Failed': {},
 'Successful': {'/lhcb/certification/test/ALLSTREAMS.DST/00000239/0000/00000239_
↪00000044_1.allstreams.dst': {'CERN_MC_M-DST': 'srm://srm-lhcb.cern.ch/castor/cern.
↪ch/grid/lhcb/certification/test/ALLSTREAMS.DST/00000239/0000/00000239_00000044_1.
↪allstreams.dst'}}}
[lxplus433] x86_64-slc5-gcc43-opt /afs/cern.ch/lhcb/software/DEV/LHCBDIRAC/LHCBDIRAC_
↪v6r0-pre12> dirac-dms-lfn-replicas /lhcb/certification/test/ALLSTREAMS.DST/00000239/
↪0000/00000239_00000045_1.allstreams.dst
{'Failed': {},
 'Successful': {'/lhcb/certification/test/ALLSTREAMS.DST/00000239/0000/00000239_
↪00000045_1.allstreams.dst': {'CNAF_MC_M-DST': 'srm://storm-fe-lhcb.cr.cnaf.infn.it/
↪t1d1/lhcb/certification/test/ALLSTREAMS.DST/00000239/0000/00000239_00000045_1.
↪allstreams.dst'}}}
[lxplus433] x86_64-slc5-gcc43-opt /afs/cern.ch/lhcb/software/DEV/LHCBDIRAC/LHCBDIRAC_
↪v6r0-pre12> dirac-dms-lfn-replicas /lhcb/certification/test/ALLSTREAMS.DST/00000239/
↪0000/00000239_00000074_1.allstreams.dst
{'Failed': {},
 'Successful': {'/lhcb/certification/test/ALLSTREAMS.DST/00000239/0000/00000239_
↪00000074_1.allstreams.dst': {'CERN_MC_M-DST': 'srm://srm-lhcb.cern.ch/castor/cern.
↪ch/grid/lhcb/certification/test/ALLSTREAMS.DST/00000239/0000/00000239_00000074_1.
↪allstreams.dst'}}}
[lxplus433] x86_64-slc5-gcc43-opt /afs/cern.ch/lhcb/software/DEV/LHCBDIRAC/LHCBDIRAC_
↪v6r0-pre12> dirac-dms-lfn-replicas /lhcb/certification/test/ALLSTREAMS.DST/00000239/
↪0000/00000239_00000076_1.allstreams.dst
{'Failed': {},
 'Successful': {'/lhcb/certification/test/ALLSTREAMS.DST/00000239/0000/00000239_
↪00000076_1.allstreams.dst': {'CNAF_MC_M-DST': 'srm://storm-fe-lhcb.cr.cnaf.infn.it/
↪t1d1/lhcb/certification/test/ALLSTREAMS.DST/00000239/0000/00000239_00000076_1.
↪allstreams.dst'}}}
```

How to enable/disable FTS channel ? To check TFS transfer, look at the log for DataManagement/FTSSubmitAgent

3.3.3 Specific tests

Every release is somewhat special, and introduce new features that should be tested. It has to be noted that developers should always participate in the testing of very specific new developments, anyway the certification manager should look into if these tests have been done.

Within Jira, there is a special board, named ready for integration. that contain tasks marked as "Resolved", but not yet "Done". Dragging tasks from left to right will mark them as "Done".

So, the certification manager can decide to investigate directly, by submitting tests, if know, or ask the developer to confirm the task can be closed.

CHAPTER 4

Indices and tables

- genindex
- search