Resources @CERN

Access to CERN resources are managed via subscription to the e-groups:

Granted to everybody of the Muon Collider collaboration
muoncollider-batch (batch queue)
muoncollider-readers (read from disk)

Granted to developers for the moment
muoncollider-writers (write to disk)

Accounting Groups

To see if you are able to submit you can check your Accounting Groups with this command on lxplus:

```
$) haggis rights
```

```
+----------------------------+
|          gianelle          |
+----------------------------+
| group_u_MUONCOLLIDER.users |
| group_u_LHCB.u_z5          |
+----------------------------+
```

If you're a member of multiple accounting groups, then you should try to ensure that you specify which group you wish to use for a specific activity. The system will otherwise assign you to one, normally the first is the default:

The syntax for the submit file is:

```
+AccountingGroup = "group_u_MUONCOLLIDER.users"
```

Condor local submission

BTW: submission to the HTCondor schedds at CERN normally makes use of a shared filesystem, ie AFS. This is convenient, but also shared filesystems introduce instability

To submit a muoncollider's job to condor on CERN you first need a submit file:

```
# Unix submit description file
+owner = undefined
request_memory = 2GB
executable = executable.sh
Arguments  = $(IDX)
log = log_$(IDX)_$(ClusterId).txt
output = outfile_$(IDX)_$(ClusterId).txt
error = errors_$(IDX)_$(ClusterId).txt
when_to_transfer_output = ON_EXIT

+JobFlavour = "testmatch"


queue
```

IDX is just an example of a parameter that you can use as argument for your job. JobFlavour is needed to choose max job wall time.

Then you need to define your executable:

#!/bin/bash

Job's argument
IDX=$1

```
#### Define some paths
### Eos
## User space where to store output file (you can also leave the files in the local directory)
EOS_USER_URL=root://eosuser.cern.ch
EOS_PATH=/eos/user/g/gianelle/MuonC/muontest

## Experiment space where for example BIB files are stored
EOS_EXP_URL=root://eosexperiment.cern.ch
EOS_EXP="/eos/experiment/muoncollider/"

### Docker file to use (i.e. which release version)
DOCKER=/cvmfs/unpacked.cern.ch/registry.hub.docker.com/infnpd/mucoll-ilc-framework:1.7-almalinux9
## the job's working directory
BASE="/afs/cern.ch/user/g/gianelle/muonc/job"
####

## Define a unique directory for the job using the argument
## in the local mode all the job sumbitted to condor are executed on the local directory
## so pay attention to the possible conflicts
WORKHOME="JOB_$IDX"

## cd in the working directory
cd $BASE

## a simple function to quit script
# ARGs: <error message> <error code>
quit () {
  echo $1
  # comment the line below if you project to leave outputs in the local directory
  rm -rf $WORKHOME
  exit $2
}

### function to copy file to eos space
# ARG1: input file name
# ARG2: output file name
copy() {
  IN=$1
  OUT=$2
  if [ -f $1 ]; then
    if ! xrdcp -f $IN ${OUT} ; then
      quit "ERROR! Failed to transfer $IN" 2
    fi
  else
      quit "ERROR! File $IN not exists" 2
  fi
}

# create the unique working dir and cd in it
mkdir $WORKHOME
cd $WORKHOME

## copy or link the auxiliary files of the job inside the job directory
# you can use a generic name fot ìinput file so you don't need to
# customize the steering file at each execution
cp $BASE/k10_out3.hepmc input.hepmc
ln -s $BASE/sim_steer.py sim_steer.py

# back to $BASE directory
cd $BASE

# exec the singularity container
echo "Start singularity job at `date`"
# NB It mounts the eos experiment directory so the BIB files are accessible
singularity exec -B$EOS_EXP $DOCKER /bin/bash sim.sh ${IDX} &> job_${IDX}.log
echo "End job at `date`"

# copy outfile on user EOS space
ext=$(( 10#${IDX} ))
postfix=$(printf "%03d" $IDX )
copy ${WORKHOME}/OUTPUT_${ext}.slcio ${EOS_USER_URL}//${EOS_PATH}/z2jmu_k10_10evt_${postfix}.slcio
copy ${WORKHOME}/simout.log ${EOS_USER_URL}//${EOS_PATH}/z2jmu_k10_10evt_${postfix}.log

quit "All is done" 0
```

Last script is the executable that you run inside the container (for example a bash script). Remember that when you execute the singularity command you jump (inside the container)
in the same directory from which you have run the command, usually your afs home directory, or as in the previous script the $BASE directory.

```
#!/bin/bash

# Job's argument
IDX=$1

# define as in the previous script the _same_ unique job directory
WORKHOME=JOB_$IDX
# number of events to process
NEVT=10

# set muoncollider environment
source /opt/ilcsoft/muonc/init_ilcsoft.sh

# cd in the working directory
cd $WORKHOME

# define the arguments for the ddsim script
INPUT=input.hepmc
# btw the output file name must be the same of the ones used in the previous script
ext=$(( 10#${IDX} ))
OUTPUT=OUTPUT_${ext}.slcio
skipEvt=$((  ( 10#${IDX} ) * $NEVT ))

echo "Start simulation at `date`"
# submit a simulation job
ddsim --steeringFile sim_steer.py --inputFile ${INPUT} --outputFile ${OUTPUT} --numberOfEvents  ${NEVT} --skipNEvents ${skipEvt}  &> simout.log

echo "End simulation at `date`"
```

To submit the file, setting an arguments, use the usual condor command:

> *condor_submit IDX=01 job.sub*

Here the job submission documentation from CERN

Condor spool submission

There are some schedds that do not allow shared filesystems on the worker node, which should make them more suitable for users who have longer jobs and are willing to have slightly more constraints.

To select the schedds:

> *module load lxbatch/spool*

Some modification are needed to scripts:

```
# Unix submit description file
+owner = undefined
request_memory = 2GB
executable = executable.sh
Arguments  = $(IDX)
log = log_$(IDX)_$(ClusterId).txt
transfer_input_files = sim_steer.py, sim.sh
transfer_output_files = job.log
output = outfile_$(IDX)_$(ClusterId).txt
error = errors_$(IDX)_$(ClusterId).txt
should_transfer_files = Yes
when_to_transfer_output = ON_EXIT
+JobFlavour = "testmatch"

queue
```

You need to define the files that need to be transfer both for input and output

```
#!/bin/bash
```

```
## Job's argument
IDX=$1

#### Define some paths
### Eos
## User space where to store output file (you can also leave the files in the local directory)
EOS_USER_URL=root://eosuser.cern.ch
EOS_PATH=/eos/user/g/gianelle/MuonC/muontest

### Docker file to use (i.e. which release version)
DOCKER=/cvmfs/unpacked.cern.ch/registry.hub.docker.com/infnpd/mucoll-ilc-framework:1.7-almalinux9
## the job's working directory
BASE="/afs/cern.ch/user/g/gianelle/muonc/job"
####

## a simple function to quit script
# ARGs: <error message> <error code>
quit () {
  echo $1
  exit $2
}

### function to copy output files to eos space
# ARG1: input file name
# ARG2: output file name
copyout() {
  IN=$1
  OUT=$2
  if [ -f $1 ]; then
    if ! xrdcp -f $IN ${OUT} ; then
      quit "ERROR! Failed to transfer $IN" 2
    fi
  else
      quit "ERROR! File $IN not exists" 2
  fi
}

### first copy the input file from the eos path
xrdcp ${EOS_USER_URL}//${EOS_PATH}/k10_out3.hepmc input.hepmc

# exec the singularity container
echo "Start singularity job at `date`"
singularity exec -B $PWD $DOCKER  /bin/bash sim.sh ${IDX} &> job.log
echo "End job at `date`"

# copy outfile on user EOS space
ext=$(( 10#${IDX} ))
postfix=$(printf "%03d" $IDX )
copyout OUTPUT_${ext}.slcio ${EOS_USER_URL}//${EOS_PATH}/z2jmu_k10_10evt_${postfix}.slcio
copyout simout.log ${EOS_USER_URL}//${EOS_PATH}/z2jmu_k10_10evt_${postfix}.log

quit "All is done" 0
```

The major difference is that we use the xrdcp command to transfer input and output files from and to eos space. Shared filesystems (i.e. afs) are still available on the worker nodes, but it is not safe to refers to it.

In the singularity command we muount the condor spool directory as the user "HOME".

Last script is the executable that you run inside the container (for example a bash script). Remember that when you execute the singularity command you jump (inside the container)
in the same directory from which you have run the command, usually your afs home directory, or as in the previous script the condor spool directory.

The only difference from the local approach is that we don't need to create a unique directory for the job.

```
#!/bin/bash

echo "Start job at `date`"# Job's argument
IDX=$1

# number of events to process
NEVT=10

# set muoncollider environment
source /opt/ilcsoft/muonc/init_ilcsoft.sh
```

```
# define the arguments for the ddsim script
INPUT=input.hepmc
# btw the output file name must be the same of the ones used in the previous script
ext=$(( 10#${IDX} ))
OUTPUT=OUTPUT_${ext}.slcio
skipEvt=$((  ( 10#${IDX} ) * $NEVT ))

echo "Start simulation at `date`"
# submit a simulation job
ddsim --steeringFile sim_steer.py --inputFile ${INPUT} --outputFile ${OUTPUT} --numberOfEvents  ${NEVT} --skipNEvents ${skipEvt}  &> simout.log

echo "End simulation at `date`"
```

To submit the file, setting an arguments, use the usual condor command:

> *condor_submit* ***-spool*** *IDX=01 job.sub*

A more complex example

If you need to reconstruct events using your own customized code (i.e. a custom processor) you need first of all to commit your code on git in a "ad hoc" branch (or you can zip your code as a input file).

In the following example we will use as configuration file one of the official one committed on the ProductionConfig package. Also we will see how to manage BIB files.

We use the "spool" method, so the submission script is like the previous one (note that for reconstruction we need more memory):

```
# Unix submit description file
+owner = undefined
request_memory = 10GB
executable = executable.sh
log = log_$(IDX)_$(ClusterId).txt
transfer_input_files = job.sh
transfer_output_files = job.log, reco.out
output = outfile_$(IDX)_$(ClusterId).txt
error = errors_$(IDX)_$(ClusterId).txt
should_transfer_files = Yes
when_to_transfer_output = ON_EXIT
+JobFlavour = "testmatch"
queue
```

Also the executable is similar to the previous one, Note that we cannot copy all the 1000 BIB files, so we choose NBIBs file randomly that will be overlays with our signal:

```
#!/bin/bash

## ARG1: an index to identify the job
IDX=$1

## Software docker image
DOCKER="/cvmfs/unpacked.cern.ch/registry.hub.docker.com/infnpd/mucoll-ilc-framework:1.6-centos8"

## Eos
EOS_USER_URL=root://eosuser.cern.ch
EOS_EXP_URL=root://eosexperiment.cern.ch
EOS_PATH="/eos/user/g/gianelle/MuonC/muontest"
EOS_BIB="/eos/experiment/muoncollider/data/BIB/MuCollv1_25ns_nEkin150MeV_QGSPBERT"

# Signal input file
INPUT="muonGun_testBIB_100k_sim.slcio"

# How many BIBs ?
NBIBs=10

### Exit function:
# ARG1: message
# ARG2: exit code
quit () {
  echo $1
  exit $2
}
```

```
### function to copy file to eos space
# ARG1: input file name
# ARG2: output file name
copy() {
  IN=$1
  OUT=$2
  if ! xrdcp -f $IN ${OUT} ; then
    quit "ERROR! Failed to transfer $IN" 2
  fi
}

## Retrieve input Signal file
copy ${EOS_USER_URL}//${EOS_PATH}/${INPUT} input.slcio

## Retrieve input BKG files
BKGPre=${EOS_EXP_URL}//${EOS_BIB}/sim_mumu-1e3x500-26m-lowth-excl_seed
BKGPost="_allHits.slcio"
# total number of available BIBs files
BKGTot=1000
# array with BIB file names
BIBs=()
for (( i=1; i<=${NBIBs}; i++ )); do
  RNDBKG=$(printf "%04d" $(($RANDOM % $BKGTot)) )
  BKGFILE=${BKGPre}${RNDBKG}${BKGPost}
  BIBFILE=BKG_seed${RNDBKG}.slcio
  copy $BKGFILE $BIBFILE
  BIBs+=( $BIBFILE )
done

# exec the singularity container
echo "Start singularity job at `date`"
singularity exec -B $PWD $DOCKER ./job.sh ${IDX} "${BIBs[@]}" &> job.log
echo "End job at `date`"

## save output files
POSTOUT=$(printf "%03d" $IDX )
copy Output_REC.slcio ${EOS_USER_URL}//${EOS_PATH}/Output_REC_${POSTOUT}.slcio
copy reco.out ${EOS_USER_URL}//${EOS_PATH}/reco_${POSTOUT}.out

quit "Well Done" 0
```

More interesting is the job that need to be execute in the container:

```
#!/bin/bash

## ARG1: the job index
## ARG2...ARGn: BIB file names

JOB=$1
shift
# the other args are BIB files
BIBs=("$@")

# define how many events per job
EVTxJOB=500
if [ ${JOB} -eq 0 ]; then
  # the first job need an event more...
  NEVT=$(( ${EVTxJOB} + 1 ))
  SKIP=0
else
  NEVT=${EVTxJOB}
  SKIP=$(( ${EVTxJOB} * ${JOB} ))
fi

source /opt/ilcsoft/muonc/init_ilcsoft.sh
BASE=`pwd`
```

```
## Function to compile processor, it also redefine MARLIN_DLL
# ARG1: processor name
compile () {
  PROCNAME=$1
  WD=`pwd`
  mkdir BUILD/${PROCNAME}
  cd BUILD/${PROCNAME}
  cmake -C $ILCSOFT/ILCSoft.cmake -DBOOST_INCLUDEDIR=/usr/include/boost173 -DBOOST_LIBRARYDIR=/usr/lib64/boost173 ../../${PROCNAME}
  make
  cd $WD
  echo $MARLIN_DLL | grep ${PROCNAME} &> /dev/null
  if [ $? -eq 0 ]; then
    NEWDLL=$(echo $MARLIN_DLL | sed -e "s#/opt/ilcsoft/muonc/${PROCNAME}/.*/lib/lib${PROCNAME}.so#${BASE}/BUILD/${PROCNAME}/lib
/lib${PROCNAME}.so#g")
  else
    NEWDLL=$MARLIN_DLL:${BASE}/BUILD/${PROCNAME}/lib${PROCNAME}.so
  fi
  export MARLIN_DLL=$NEWDLL
}


### Clone the needed repositories
# configuration file
git clone https://github.com/MuonColliderSoft/ProductionConfig.git
# clone new processors, specifying the required branch
git clone -b MoveHitsCut https://github.com/MuonColliderSoft/MarlinTrkProcessors.git
git clone -b jetReco https://github.com/MuonColliderSoft/LCTuple.git
# compile processors
echo "Starting compiling code at `date`"
mkdir BUILD
compile "MarlinTrkProcessors"
compile "LCTuple"


echo "Start job at `date`"

Marlin --global.MaxRecordNumber=${NEVT} --global.SkipNEvents=${SKIP} --OverlayTrimmed.BackgroundFileNames="${BIBs[*]}" allProcess.xml &> reco.
out

echo "Everything end at `date`"
```

Submitting Multiple Jobs

To submit more jobs you can use a "for" cycle from bash command line changing the IDX parameter, or you can modify you submit file in this way:

```
start = $(Process) + 20
IDX = $INT(start,%d)
Arguments = $(IDX)

...
queue 40
```

With these lines, for example, condor submits 40 jobs starting from Arguments=20 (see first line). In other words this is equivalent to the bash command line:

```
for index in  {20..59}; do condor_submit --spool IDX=${index} job.sub; done
```

For other option see for example here

Commands

This is a list of the more used commands, see documentation for more info:

```
## submit job
condor_submit -spool  job.sub

## query all mine jobs
condor_q
condor_q -l <jobid>
condor_q -nobatch
```

query all jobs
condor_q -all

remove single job
condor_rm <jobid>

remove all _terminated_ jobs
condor_rm -constraint 'JobStatus == 4'

ssh to job's worker node
condor_ssh_to_job <jobid>

transfer condor stdout/err
condor_transfer_data <jobid>

Eos

/eos/experiment/muoncollider/ is the experiment available space

eosexperiment.cern.ch is the host of the EOS (xroot) server

```
export EOS_MGM_URL=root://eosexperiment.cern.ch
```

Commands

User can access (list and read) files in CERN EOS from non lxplus nodes, without being authenticated.

In order to get full permission(write/delete), one should use CERN account to get **Kerberos** authentication:

> *[gianelle@muonc ~]$ kinit gianelle@CERN.CH*
> *Password for gianelle@CERN.CH:*
> *[gianelle@muonc ~]$ klist*
> *Ticket cache: FILE:/tmp/krb5cc_3807*
> *Default principal: gianelle@CERN.CH*
>
> *Valid starting Expires Service principal*
> *06/29/23 11:42:42 06/30/23 12:42:39 krbtgt/CERN.CH@CERN.CH*
> *renew until 07/04/23 11:42:39*

Create directory:

eos mkdir /eos/experiment/muoncollider/test

Copy file:

xrdcp test.txt ${EOS_MGM_URL}//eos/experiment/muoncollider/

List files:

eos ls -l /eos/experiment/muoncollider/

HowTO

See here for a tutorial to access eos from lxplus.cern.ch

Here a talk on **EOS for Users**