# Table of Contents

- CRIC Evaluation
  - Introduction
  - Evaluation of CRIC by the CMS experiment
  - Requirements
    - User Interface
    - Data Structures
      - Region Information
      - Site Information
      - Compute Resource Information
      - Storage Element Information
      - Compute Unit Information
      - Storage Unit Information
      - Compute Queue Information
  - Example Use Cases
  - Dynamic Information
  - SiteDB APIs
  - Mutable information
    - Things to discuss
  - First Step (2016-May-13)
  - Feedback for first prototype
  - Installing CRIC prototype for CMS
- Some useful links
CRIC Evaluation

Introduction

The WLCG Configuration system should describe WLCG distributed resources. From the functionality point of view it should provide:

- service discovery
- central validation of the provided information
- description of all resources used by the LHC VOs both pledged and opportunistic (clouds, HPC) and all kind of services used by the LHC VOs.
- historical data and logging of the information update (who, when, how )
- consistent set of UIs/APIs for the LHC VOs

From the implementation point of view, the system should consist of two parts:

- The core generic part which describes all physical service endpoints. It caches information from the information sources as GocDB and OIM and provides a single entry point for WLCG service discovery
- Experiment-specific part, implemented in a form of plugins. It describes how the physical resources are used by the experiments and contains additional attributes and configuration which is required by the experiments for operations and organization of their data and work flows.

The first prototype of the core generic part (CRIC) can be found here.

Evaluation of CRIC by the CMS experiment

Summary of the meeting with CMS 11.03.2016

Requirements

LHC experiments use the Worldwide LHC Computing Grid (WLCG) to perform their event reconstruction, processing, simulation, and data analysis. WLCG encompasses various grid installations and technologies. To use the computing resources requires detailed knowledge about them. The Computing Resource Information Cache (CRIC) will collect the relevant information about WLCG computing resources, provide the experiments with a unified and consistent view of this information, and keep it up-to-date.

Grid computing is still a rapidly developing field. CRIC must thus be agile and able to quickly adapt to new type of computing resources, new information sources, and requests for different/additional information from the experiments. The LHC experiments adapt their computing steadily to meet the changing needs of the collaboration. CRIC must be flexible, easy to enhance, and simple to use. The core part, common to all LHC experiments, should be fully supported, i.e. developed, maintained, enhanced, and operated, by WLCG/CERN-IT. The contents in CRIC should be maintained, i.e. filled, verified, and updated as needed, by WLCG/CERN-IT. The lifetime of the information varies greatly. It is envisioned that update intervals of less than 15 minutes are not needed. CRIC should have a history option that, if enabled for a quantity, keeps historical information.

The LHC collaborations must be able to extend CRIC to cache or store experiment-specific grid resource information. This could be extensions to already existing data structures, e.g. experiment-specific site names, or new data structures, e.g. experiment-specific grouping of compute elements. Collaborations are stretched on skilled programming resources. CRIC must be extensible by non-experts and the extensions supportable with little effort. The system must be fully documented and extension examples provided.
User Interface

CRIC must have a web-based user interface to browse all information in the system. Common, i.e. not experiment-specific, information should be accessible by at least anyone with a CERN affiliation/account (identified access, i.e. user providing a certificate without any user restriction, as is for most grid information systems, is preferred). All experiment specific information should be accessible only to people affiliated with the experiment. The web interface should be intuitive and designed for non-experts. A full description of each quantity should be accessible. The source of the cached information, the time it was cached, and the schedule of the next check/update must be readily accessible.

CRIC must also provide JSON and XML access to the information to enable easy access from within programs. Information must be accessible in subsets, e.g. only information of compute elements, only pledges, only site administrator information, etc. JSON and XML access must allow a simple selection/filter option, e.g. to return only pledges for a given experiment and a given year, site administrator information for a specific site, etc.

To update experiment specific information write access is required. Both a web interface and an API are required. Authorization to write, i.e. insert or update/change, information must be granular and allow both horizontal and vertical authorization, i.e. write access to information of a region/site/compute, storage unit/compute, storage element and to information of a particular quantity, e.g. compute element information, space available to the experiment in a storage element, etc. The client API must be available for the current versions of Scientific Linux and be easy to install. Authentication should be based on X.509 certificates or CERN Kerberos. Any write access must be recorded with timestamp, user, and action, i.e. previous value and new value.

CRIC must provide management access to change the layout/organization of cached/stored information (not necessarily web based). No management granularity within an experiment is required.

Data Structures

Grid information in envisioned to be arranged hierarchically, regions contain sites, and sites contain resource elements. CMS requires an additional "resource units" layer between sites and resource elements which can be part of the experiment specific extension. Below are incomplete examples of information needed in various data structures.

Region Information

name, contact(s), pledge(s), list-of-sites

Site Information

name, contact(s) exec/admin/tech, location, pledge(s), list-of-resource-elements, CMS-sitename, list-of-resource-units

Compute Resource Information

name, contact(s), location, type, status, capacity, performance, quota, queue(s), support-level, squid(s), stage-out(s), factory-config, factory-tag subsite, --> grid-CE

Storage Element Information

name, contact(s), location, type, technology, status, capacity, access(es), quota-used
**Compute Unit Information**

name, contact(s) admin/tech, status, list-of-compute-elements, associated-storage-units, squids, stage-out(s)

**Storage Unit Information**

name, contact(s) admin/tech, status, list-of-storage-elements, associated-compute-units, PhEDEx-nodename, links-to-other-storage-units, quota

**Compute Queue Information**

max-walltime, max-disk, max-memory, DNs-accepted, scheduling/eviction

**Example Use Cases**

Many CMS applications, especially for monitoring, etc. need to know the sites that provide CMS support.

- List of sites (CMS name, tier level)
  - Data information of a site (PhEDEx name, SE endpoints, parent Tier)
  - Computing information of a site (Processing name, CE endpoints/queue)
  - Site details (country/location, site contacts)
  - pledged/installed/available CPU
  - pledged/installed/available storage

All sites (PhEDEx nodename instance) translate logical file names into an URI/physical file name, i.e. protocol, nodename, filepath, and filename, for the sites (processing site instance).

- Logical-filename to physical-filename mapping

CMS uses Glide-In WMS. The glide-in factories around the world, need to be configured. CRIC has the queue information, so their configuration could be extracted from CRIC (plus CMS experiment specific extension).

SiteDB is currently used to connect compute resources to storage resources and vice versa.

- processing sitename to PhEDEx nodename mapping

**Dynamic Information**

Downtime information is used by shifters in case of a site error and by programs that evaluate sites. Shifters want to see current downtimes for any site that supports CMS, production operators and also individual users want to see the downtimes of the day and/or of the week for sites supporting CMS. For site evaluation programs need an API to get the list of downtimes for sites that support CMS within a time window.

- query input: start time, end time
- query output: site name, resource element, start time, end time, time the downtime was scheduled, reason for the downtime

An important note, downtimes within a time window includes downtimes that started before the time window (and ended after the time window).

- Used space (both current and hystorical, link to dashboard?)
- Used CPU (both current and hystorical, link to dashboard?)
- Site Readiness (based on custom CMS metric)
SiteDB APIs:

documentation

- from site/resource unit:
  - sites
  - site-names
  - site-resources
  - site-responsibilities
- people: not needed but taken from CERN HR database instead
  - people
- membership: role, group, list-of-usernames
  - whoami
  - roles
  - groups
  - group-responsibilities

Mutable information

- List of sites (CMS name, tier level)
- SITECONF management (configuration files holding storage and local batch details)
- siteconf repository

Things to discuss

- What about WMAgent, glideinWMS, in general the submission infrastructure...
- What about CRAB

First Step (2016-May-13)

The goal of the first step is to populate minimal information for the CMS Tier-0 and Tier-1 sites and extract downtime information for the grid services used at those sites. This requires

- first implementation of the CMS site data structures and interface to allow defining a CMS site (with storage and compute units) and relate grid CEs and SEs to the site (units).
- caching of downtime information from grid middleware/infrastructure organizations and updating/maintaining the cache;
- an API to access the information as described above.

Once data structures and interface to define a CMS site are ready, CRIC and CMS team members will define the CMS Tier-0 and Tier-1 sites in CRIC. CMS will then update some scripts to use the CRIC API for downtime information.

Feedback for first prototype

<table>
<thead>
<tr>
<th>Action</th>
<th>Status</th>
<th>Input needed</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment Site Definition</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove cloud concept as it doesn’t exist in CMS</td>
<td>DONE</td>
<td>Alexey</td>
<td>Cloud removed</td>
</tr>
<tr>
<td>In Experiment site field, do not include by default the GOCDB/OIM name</td>
<td>DONE</td>
<td>Alexey</td>
<td>removed default suggestion</td>
</tr>
<tr>
<td>Is it possible to remove sites? How?</td>
<td>yes, but not from WebUI</td>
<td>Alexey</td>
<td>object can be DISABLED to become automatically hidden in</td>
</tr>
</tbody>
</table>

SiteDB APIs:
<table>
<thead>
<tr>
<th>Question</th>
<th>Yes/No</th>
<th>Alexey</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email contact list is comma separated?</td>
<td>yes</td>
<td></td>
<td>Current prototype, whatever format/structure can be implemented.</td>
</tr>
<tr>
<td>Email contact is not visible in the ExperimentSite view</td>
<td>done</td>
<td></td>
<td>Added</td>
</tr>
<tr>
<td>Does CMS need a more structured list of email contacts? Like DataManager, PhEDEx contact, site Admin, site Executive?</td>
<td>Yes, at least we should be able to differentiate site executives from site admins</td>
<td>CMS</td>
<td>Only CE type is needed</td>
</tr>
<tr>
<td>Resources already exist as part of the central CRIC, correct? We only need to attach them to the ComputeUnits?</td>
<td>yes</td>
<td></td>
<td>Alexey</td>
</tr>
<tr>
<td>Does CMS need Batch System type information?</td>
<td>NO</td>
<td>CMS</td>
<td>Only CE type is needed</td>
</tr>
<tr>
<td>Protocol needed in the endpoint</td>
<td>yes</td>
<td>Alexey</td>
<td>In current prototype the scheme value is mandatory. In ATLAS we improved a lot SE and its protocol declaration. This form will be updated, but for prototype I think it's OK?</td>
</tr>
<tr>
<td>Check input data button doesn't seem to work</td>
<td>it works</td>
<td></td>
<td>Alexey</td>
</tr>
<tr>
<td>What about StorageUnit, do you want to create this in a next step? Storage services are not defined in the devel instance</td>
<td>yes</td>
<td>Alexey</td>
<td>SE services are defined (auto collected from GOCDB.OIM), but in state DISABLED. StorageUnit implementation is considered as next step</td>
</tr>
<tr>
<td>The free text for State, is it a good idea? What does it mean?</td>
<td>?</td>
<td>Alexey</td>
<td>The state indicates if object is ACTIVE (means being used by the system) or DISABLED (means need to be deleted); probably the question is related to the status field.</td>
</tr>
<tr>
<td>Since the VOFeed doesn't include the queue name, I was selecting queues based on the fact that its name was containing cms, otherwise, it's impossible to known which queues are for CMS. Could CMS confirm this? Maybe using Factory config file?</td>
<td>Queue info is indeed in Factory config file</td>
<td>CMS</td>
<td>No need to enter real data to validate prototype. Ideally, CRIC should be able to produce the Factory config file in the future</td>
</tr>
</tbody>
</table>

### Installing CRIC prototype for CMS

The instructions below describe the steps needed to install the CRIC prototype for CMS.

---

Feedback for first prototype
WLCGConfigurationEvaluation < LCG < TWiki

Note that the instructions below were tested in a CC7 VM with python 2.7.

Note in any case that if you are planning to use an SL6 VM, make sure package useraddcern is installed and that the user that is going to run the installation steps is added to the VM by running useradd username. It is recommended to choose the image SL6 CERN server.

You will need host certificates in the VM where CRIC is going to be installed.

Preliminary steps

- Make sure you have an account on gitlab and that your ssh public key is uploaded in your profile.
- Deploy a VM and make sure it contains the following rpms and their necessary dependencies (use yum to install them, in most cases are available in the CERN repos, otherwise, the link to the used rpm is included):
  - `git`
  - `python`
  - Note that in CC7, it’s used python 2.7. When using python 2.7, the script `/data/gitcms/manage.py` and all the `setup.cfg` files present in all agis directories under `/data/gitcms` (these files are available from CRIC codebase, as explained in the instructions below) need to be modified to call python instead of python2.6. In `/data/gitcms/agis_django.web.httpd/templates.cfg` and `/data/gitcms/agis_django.api.httpd/templates.cfg`, the paths need to be modified to include `/usr/lib/python2.7/site-packages/` in all the occurrences.
  - `rpm-build`
  - `Django 1.5.5` (For CC7, Django version available in repo is OK)
  - `python-simplejson`
  - `python-ldap`
  - `cx_Oracle SL6` `cx_Oracle CC7`
  - `python-gdata` (only in CC7 installation)
  - `httpd`
  - `mod_ssl`
  - `mod_wsgi`
  - `CERN CA certificates`
  - Disable the SELinux module (run `setenforce 0` as root)
  - Make sure the hostnames where CRIC web and API are going to run are registered in the DNS

Download CRIC codebase:

- `ssh` to your VM with the account in gitlab corresponding to the uploaded ssh key
- `mkdir /data/gitcms/`
- `cd /data/gitcms`
- `git clone ssh://git@gitlab.cern.ch:7999/agis/agis.git`
- `git checkout -b cms origin/cms`

Build CRIC packages

- `mkdir /data/dist/
  - python2.6 manage.py `--dist-dir=/data/dist/` --release=999 --bdist_rpm`

Deploy API + WebUI rpms:

- `cd /data/dist/
  - sudo rpm -i agis-api-1.3.5-999.noarch.rpm agis-common-1.3.6-999.noarch.rpm agis_django-api-1.5.1-999.noarch.rpm agis_django-apps-1.5.1-999.noarch.rpm

Installing CRIC prototype for CMS
Normally API and WebUI packages are deployed separately in production. If both API and WebUI are on same machine, apache configuration needs to be patched:

- cd /data/gitcms
- vi agis_django.web.httpd/config/httpd/agis-web.conf.template
- Comment first line:
  - WSGIPythonPath /etc/agis-web
- Uncomment 2nd, 9th, 10th, 31st and 32nd lines
  - #WSGISocketPrefix run/wsgi
  - #WSGIDaemonProcess agis-web processes=2 threads=10 display-name=agis-web
  - python-path=/etc/agis-web
  - #WSGIProcessGroup agis-web
- vi agis_django.api.httpd/config/httpd/agis-api.conf.template
  - Do the same for API httpd configuration: comment 1st line and uncomment 2nd, 10th, 11th, 31st and 32nd lines in same way as for WebUI

Configure RPMs containing apache settings:

- cd /data/gitcms/agis_django.web.httpd
- change T_ServerName to the proper value in templates.cfg:
  - T_ServerName = crictest1.cern.ch
- Then build the RPM with the corresponding release
  - python setup.py bdist_rpm --release=crictest1
- As a result, a new rpm will appear:
  - dist/agis_django-web-httpd-0.3.2-cric_cms_dev.noarch.rpm
- Copy it to dist or/and simply install:
  - cp -p dist/agis_django-web-httpd-0.3.2-cric_cms_dev.noarch.rpm; sudo rpm -i dist/agis_django-web-httpd-0.3.2-cric_cms_dev.noarch.rpm
- Do the same with API apache settings stored in agis_django.api.httpd package
  - cd ../agis_django.api.httpd/
  - vi templates.cfg and change T_ServerName to crictest1-api.cern.ch (Note that even if webui and api are in the same machine, the api hostname must be different in the settings)
  - python setup.py bdist_rpm --release=crictest1-api
  - cp -p dist/agis_django-api-httpd-0.3.2-cric_cms_api_dev.noarch.rpm /data/anisyonk/dist/;
    sudo rpm -i dist/agis_django-api-httpd-0.3.2-cric_cms_api_dev.noarch.rpm

Configure settings:

- Fix hostnames, DB settings, initial STAFF user DN, etc settings if needed in settings_essentials.py and store them under:
  - /etc/agis-web/settings_essential.py (template) File used for the deployment tests below:

```python
current_config = 'DEVEL'
DEBUG = True
import os
config_presets = {
    'DEVEL': {
        'DATABASES': {
            'default': {
                'ENGINE': 'django.db.backends.sqlite3',
                # 'NAME': 'agisdb',
```
WLCGConfigurationEvaluation < LCG < TWiki

'NAME': '/var/spool/agis/agisdb.sqlite',
'USER': 'agis',
'PASSWORD': 'password',
'USER_CREATE': 'agis',
'PASSWORD_CREATE': 'password',
'HOST': '127.0.0.1',
'PORT': '123',
}

# 'DEFAULT_DB_SCHEMA': 'ATLAS_AGIS',

## 'SSL_AUTH_FORCE_ENV': { # used for DEBUG
##    'SSL_CLIENT_S_DN': 'xx', # used to simulate SSL when DISABLE_SSL is set
##    'SSL_CLIENT_VERIFY': 'SUCCESS',
##},

'UNSECURED_HOST': 'http://criccc7.cern.ch',
'SECURED_HOST': 'https://criccc7.cern.ch',
ALLOWED_HOSTS: ['.cern.ch', 'criccc7'],

'TMPDIR': os.environ.get('TMPDIR', os.environ.get('TMP', '/tmp')),

# 'TMPDIR': '/var/spool/agis/tmp',

'PYTHON_PATH': ['local/agis/trunk/base'],

'VOMS_CERT': '/etc/agis-web/agisbot.usercert.pem',
'VOMS_KEY': '/etc/agis-web/agisbot.userkey.pem',

'DISABLE_SSL': False,
'DISABLE_AUTH': False,

'LOGGER_FILENAME': '/var/log/agis/agis_Server.web.log',

# 'DUMP_JSON_DIR': os.path.join(os.environ.get('TMPDIR', os.environ.get('TMP', '/tmp')), 'json_data'),

'MANAGERS': ( ('AGIS Team', 'atlas-adc-agis@cern.ch'), ),  # list of emails for any (admin) notifications

'WEBUI_NOTIFY': ('malandes@cern.ch', ),               # emails for changes notifications made through WebUI

'IS_CMS': True,
'IS_CRIC': True,

'FIRST_STAFF_USER_DN': '/DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=malandes/CN=644124/CN=Maria Alandes Pradillo',

'SECRET_KEY': 'g0(9*y7e9j$NjH8h%m4hg%u6%#326c298hg$10%uJNblnv', # Make this unique, and

'AIP_HOST': 'criccc7-api.cern.ch'  # hostname of API node

config = config_presets.get(current_config)

if not config:
    raise Exception("Failed to obtain config data! Config data for configuration preset '%s' not

    • /etc/agis-api/settings_essential.py (template)

    current_config = 'DEVEL'
    DEBUG = True

    import os

    config_presets = {

        'DEVEL': {

        }
    }

Installing CRIC prototype for CMS
'DATABASES': {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        # 'NAME': 'agis',
        'NAME': '/var/spool/agis/agisdb.sqlite',
        'USER': 'agis',
        'PASSWORD': 'password',
        'USER_CREATE': 'agis',
        'PASSWORD_CREATE': 'password',
        'HOST': '127.0.0.1',
        'PORT': '123',
    }
},

##'SSLAUTH_FORCE_ENV': {  ### used for DEBUG
##    'SSL_CLIENT_S_DN': 'xx', # used to simulate SSL when DISABLE_SSL is set
##    'SSL_CLIENT_VERIFY': 'SUCCESS',
##},

'UNSECURED_HOST': 'http://criccc7-api.cern.ch',
'SECURED_HOST': 'https://criccc7-api.cern.ch',
ALLOWED_HOSTS: ['.cern.ch', 'criccc7-api'],
'TMPDIR': os.environ.get('TMPDIR', os.environ.get('TMP', '/tmp')),
'PYTHON_PATH': ['local/agis/trunk/base'],
'DISABLE_SSL': False,
'DISABLE_AUTH': False,
'LOGGER_FILENAME': os.path.join(os.environ.get('TMP', '/tmp'), '.agis.api.log'),
'CRON_LOGGER_FILENAME': os.path.join(os.environ.get('TMP', '/tmp'), '.agis.api.cron.log'),
'CACHE_JSON_DIR': os.path.join(os.environ.get('TMPDIR', os.environ.get('TMP', '/tmp')), 'jsoncache'),

# 'IS_CRIC': True,
'IS_CMS': True,

# Make this unique, and don't share it with anybody.
'SECRET_KEY': 'g0(9*y7e9j$NjH84h^m4hg%!u&#3266c2zntb$i0*%ujNbl83x'

}
}

config = config_presets.get(current_config)

if not config:
    raise Exception("Failed to obtain config data! Config data for configuration preset '%s' not")

    • If using sqlite3 backend, please, make sure the right location is defined for TMPDIR,
    LOGGER_FILENAME, DUMP_JSON_DIR and NAME. i.e. 'NAME':
    '/var/spool/agis/agisdb.sqlite'

    • In CC7, due to changed access control settings in apache, the following needs to be added to
    /etc/httpd/conf.d/agis-web.contents and /etc/httpd/conf.d/agis-api.contents:

<Directory /usr/lib/python2.7/site-packages/agis_django/projs/web>
<Files wsgi.py>
    <IfVersion < 2.3 >
        Order allow,deny
        Allow from all
    </IfVersion>
    <IfVersion >= 2.3>
        Require all granted
    </IfVersion>
</Files>
</Directory>
And also in CC7, only in /etc/httpd/conf.d/agis-web.contains please add:

```html
•
<Directory ~ "/usr/lib/python2.7/site-packages/agis_django/projs/web/(static_external|static)">
  Require all granted
</Directory>

<Directory /usr/lib/python2.7/site-packages/django/contrib/admin/media/>
  Require all granted
</Directory>

<Directory /var/spool/agis/jsondir/>
  Require all granted
</Directory>
```

Those files should be readable for apache service user

- In ATLAS we use special AGIS service account atagadm and do everything AGIS related under its environment. But it could be any user, even apache -- just make sure that it has access to settings and log dirs, like /var/spool/agis/.
- Note that even if webui and api are in the same machine, the api hostname must be different in the settings as well!

Service configuration: Crons and DB population:

- DB model creation, DB bootstrap from CRONS
  - sudo -u atagadm -E bash (it could be done from any user but be sure this user has access to /etc/agis-web/settings_essential.py and that /etc/agis-web/ is added to PYTHONPATH env variable)
  - export PYTHONPATH=/etc/agis-web/
- Configuration of Robot certificate:
  - CRIC requires any robot or client certificate to communicate with external sources (fetch protected data) like GOCDB or OIM, as well as to populate its own database via REST API by crons (e.g. downtimes injections). The certificate can be generated by CERN CA.
  - Once the certificate is registered/created/obtained, it should be converted separately to two *.pem files containing the certificate itself and the private key.
  - Then *.pem files generated above should be moved to WebUI settings dir /etc/agis-web/ and the full path of these files should be configured in settings_essentials.py config variables. i.e.:
    ```python
    VOMS_CERT:'/etc/agis-web/agisbot.usercert.pem
    VOMS_KEY:'/etc/agis-web/agisbot.userkey.pem
    ```
  - The DN of the certificate needs to be registered in GOCDB/OIM to communicate via their API (To be confirmed)
- python2.6 -m agis_django.projs.web.manage syncdb # create DB models
- python2.6 -m agis_django.projs.web.manage syncdb # run it again for some post-DB initializations
- Crons to populate DB (Note that the list of installed crons are available from CRIC-DEV machine: atagadm@aiatlas056NOSPAMPLEASE.cern.ch)
  - export DJANGO_SETTINGS_MODULE=agis_django.projs.web
  - python2.6 -u -m agis_django.apps.agis_crons.crons.manager --cron OIMGOCDBInfoLoaderCron -c start createjson=True &>
  - python -u -m agis_django.apps.agis_crons.crons.manager --cron OIMGOCDBInfoLoaderCron -c start createjson=True
  - python -u -m agis_django.apps.agis_crons.crons.manager --cron=GStatLoaderCron -c start -o collectregcenters createjson=True
Downtimes:

CRIC Groups need to be configured and access granted to robot (agisbot) DN certificate for downtime injections:

- [http://cric-cms-dev.cern.ch/core/user/list/](http://cric-cms-dev.cern.ch/core/user/list/)
- [https://cric-cms-dev.cern.ch/core/user/edit/2/](https://cric-cms-dev.cern.ch/core/user/edit/2/)
- [https://cric-cms-dev.cern.ch/core/group/list/DOWNTIME-ADMINS/](https://cric-cms-dev.cern.ch/core/group/list/DOWNTIME-ADMINS/)
- To create new user group: [https://cric-cms-dev.cern.ch/core/group/create/](https://cric-cms-dev.cern.ch/core/group/create/)

Some useful links

- Link to the WLCG Information System task force twiki
- Description of the future use cases for the WLCG IS

-- JuliaAndreeva - 2016-03-14