# Table of Contents

# TABLE OF CONTENTS

## Complete instructions to create a CRIC vm using puppet

Instructions to install cric on a vm.

1. Check your permissions ( http://configtraining.web.cern.ch/configtraining/introduction/permissions.html ) in order to create a vm using puppet
2. SSH into a `aiadm.cern.ch` machine with your cern username and run the following commands (replacing `{hostname}` with the desired hostname) to create the vm in the 'cric' cluster:

   ```
   eval $(ai-rc 'cric')
   ai-bs --cc7 --foreman-environment qa --foreman-hostgroup cric/cms --landb-responsible cric
   ```

   In ~5 mins the VM should be ready (you can check it on the openstack portal after selecting the right project in the top menu bar).
3. SSH into your new VM, copy the template apache config file and edit it to replace your `{hostname}` and `{user-group}` in it (lines 3 and 74)

   ```
   cp /etc/httpd/conf.d/cric.conf.template /etc/httpd/conf.d/cric.conf
   vim /etc/httpd/conf.d/cric.conf
   ```
4. Create the necessary CRIC directories

   ```
   mkdir -p /data/cric/workspace/
   mkdir -p /var/log/cric/
   mkdir -p /data/cric/cric.conf/web/
   mkdir -p /data/cric/cric.conf/static/
   ```
5. Initialize and activate you virutalenv

   ```
   virtualenv /data/cric/workspace/cric
   source /data/cric/workspace/cric/bin/activate
   ```
6. Fetch the code from git (replace {user} with your gitlab username):

   ```
   git clone http://{user}@gitlab.cern.ch/cric/cric.git /data/cric/git/cric
   ```
7. Export location for "settings_local.py" and CMS-specific settings

   ```
   export PYTHONPATH=/data/cric/cric.conf/web/:${PYTHONPATH}
   export DJANGO_SETTINGS_MODULE='cric.web.cms.settings'
   ```
8. Configure your /data/cric/cric.conf/web/settings_local.py which contains sensitive project configuration data (password, hostnames, etc): The minimum you need to do is fill in a new SECRET_KEY, your gitlab token and point to your certificates on the four relevant places.

   ```
   """
   Local (host dependent) sensitive settings
   """
   import os
   import socket
   BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
   conf = 'DEVEL'
   DEBUG = True
   config = {
       'settings_path': 'cric.web.cms.settings',
       'VO_NAME': 'cms',
       'VIRTUALENV_DIR': os.path.join(os.path.dirname(BASE_DIR), 'workspace/cric'),
       'SECRET_KEY': '',
   ```

```
    'DATABASES': {
    'default': {
          'ENGINE': 'django.db.backends.sqlite3',
          'NAME': os.path.join(BASE_DIR, 'cricdb.sqlite3'),
        }
    },
    'PYTHON_PATH': [os.path.dirname(os.path.abspath(__file__)), ],
    'ALLOWED_HOSTS': ['localhost', '127.0.0.1', '[::1]', socket.getfqdn()],
    'VOMS_CERT': 'path/to/file.crt.pem',
    'VOMS_KEY': 'path/to/file.key.pem',
    'TMPDIR': '/data/cric/tmp',
    'CMS_CERT': 'path/to/file.crt.pem',
    'CMS_KEY': 'path/to/file.key.pem',
    'GITLAB_TOKEN': '',
    'MIGRATIONS_STORE_MODULE': 'xmigrations',
    'MIGRATIONS_STORE_PATH': '/data/cric/',
    'STATIC_ROOT': '/data/cric/cric.conf/static',
    # 'SESSION_COOKIE_SECURE': True,
    # 'CSRF_COOKIE_SECURE': True,
    'SSOAUTH_LOGIN_URL': 'https://%(host)s/Shibboleth.sso/?target=%(redirect_url)s',
    'SSOAUTH_LOGOUT_URL': 'https://login.cern.ch/adfs/ls/?wa=wsignout1.0&returnurl=',
}
settings_path = config.get('settings_path')
VIRTUALENV_DIR = config.get('VIRTUALENV_DIR')
```

9. Install gcc:

```
yum install -y gcc
```

10. Install Cric:

```
cd /data/cric/git/cric
python setup.py develop
```

11. Install cric's dependencies:

```
pip install -r requirements.txt #there might be problems with pycurl depending on what cur
```

12. Use migrations to propagate Model changes into DB schema

```
mkdir /data/cric/git/cric/lib/cric/web/cms/xmigrations
touch /data/cric/git/cric/lib/cric/web/cms/xmigrations/__init__.py
python -m cric.web.cms.manage makemigrations --all
python -m cric.web.cms.manage migrate --run-syncdb
```

13. Populate the database

Note: There is an ongoing ticket in GGUS discussing an issue that is crucial for CRIC to be able to fetch data from GOCDB. If at the time of CRIC installation, the ticket has not been resolved appropriately, then please obtain a cache file for GOCDB data (send an email to : cric-devs@cernNOSPAMPLEASE.ch requesting a copy of gocdb_services.xml) and cp the file received to /data/cric/tmp/

```
mkdir -p /data/cric/temp
## fetch sites & services from GOCDB/OIM source
python -u lib/cric/apps/cric_crons/crons/manager.py --cron OIMGOCDBInfoLoaderCron -c start create
## init regional centers
python -u lib/cric/apps/cric_crons/crons/manager.py --cron GStatLoaderCron -c start -o collectreg
## collect pledges
python -u lib/cric/apps/cric_crons/crons/manager.py --cron GStatLoaderCron -c start -o collectple
python -u lib/cric/apps/cric_crons/crons/manager.py --cron GStatLoaderCron -c start -o collectple
python -u lib/cric/apps/cric_crons/crons/manager.py --cron GStatLoaderCron -c start -o collectple
python -u lib/cric/apps/cric_crons/crons/manager.py --cron GStatLoaderCron -c start -o collectple
### collect Downtimes from GOCDB for last 90 days for ALL the sites (skip_unknown_sites=False) ev
### and push data via REST API
python -u lib/cric/apps/cric_crons/crons/manager.py --cron GOCDBLoaderCron -c start createjson=Tr
## collect (HTCondor) CE & (local) Queues from OSG source
python -u lib/cric/apps/cric_crons/crons/manager.py --cron OSGLoaderCron -c start createjson=True
## collect Site details, (HTCondor) CE and (local) Queues from BDII
python -u lib/cric/apps/cric_crons/crons/manager.py --cron BDIILoaderCron -c start createjson=Tru
```

```
# collect SiteDB information:
python -u lib/cric/apps/cric_crons/crons/manager.py --cron SITEDBInfoLoaderCron -c start createjs
# collect Glidein entries from github
python -u lib/cric/apps/cric_crons/crons/manager.py --cron GithubLoaderCron -c start createjson=T
# collect squids information from gitlab-hosted site configuration files
# Note: This collector uses the GitLab api which requires a token to run. Add your token at setti
# Instruction to get the token: https://docs.gitlab.com/ce/user/profile/personal_access_tokens.ht
python -u lib/cric/apps/cric_crons/crons/manager.py --cron GITLABLoaderCron -c start createjson=T
# collect Storage informations from vofeed
python -u lib/cric/apps/cric_crons/crons/manager.py --cron VofeedLoaderCron -c start createjson=T
```

1. Collect the static files

```
cd /data/cric/cric.conf/static/
python -m cric.web.cms.manage collectstatic
```

2. Change the owner of the cric root folder (/data/) to the user you specified in the apache configuration:

```
chown -R {user} /data/
```

3. Enable automatic startup of shibboleth daemon then restart Apache and Shibboleth:

```
/sbin/chkconfig --levels 345 shibd on
/bin/systemctl restart shibd
/bin/systemctl restart httpd
```

4. Register your application for Shibboleth by filling this form:

- insert `{hostname}` into the field "Application Name"
- insert `https://{hostname}.cern.ch/Shibboleth.sso/ADFS` into the field "Application Uri"
- insert `https://your_VM.cern.ch/` into the field "Application Homepage"
- fill the application description
- click the button "Send Registration Request"

# Full procedure to setup your CRIC "dev" environment on a CERN VM

## 1.Create your VM on

1. Log in to CERN OpenStack
2. Create a CC7 Extra VM on the IT WLCG operations project. Full instructions can be found here. Should you not be able, let us know: in the meantime, the personal project is ok.
3. Log into your VM from terminal
4. When you develop your django application you better not be "root". You should instead use your CERN user account:
   - Execute the following locmap command for all needed services:

```
locmap --configure sudo
locmap --configure sendmail
locmap --configure ntp
locmap --configure gpg
locmap --configure ssh
locmap --configure lpadmin
locmap --configure kerberos
locmap --configure afs
```

   - Add your cern user account by executing

```
addusercern <yourCERNuseraccount>
```

   - Grant privileged access to yourself, by executing the command visudo:

```
# User privilege specification add user <yourCERNuseraccount>
root ALL=(ALL) ALL
<yourCERNuseraccount> ALL=(ALL) ALL
```

## 2.Installation and configuration of Apache/SSL/WSGI/Shibboleth & VOMS client-side libraries

1. Instructions reported below are based on the cern tutorial for Centos7. In case of problem you can also contact the IT support page.
2. Register your application for Shibboleth by filling this form:
   - insert your_VM into the field "Application Name"
   - insert https://your_VM.cern.ch/Shibboleth.sso/ADFS into the field "Application Uri"
   - insert https://your_VM.cern.ch/ into the field "Application Homepage"
   - click the button "Send Registration Request"
3. Install **Apache 4.*** and needed dependencies on the machine:

   ```
   yum -y install httpd
   ```
4. Install SSL and WSGI on the machine

   ```
   yum -y install openssl mod_ssl mod_wsgi
   ```
5. Install also development libraries for openssl and nss (provided by CERN repo):

   ```
   yum -y install openssl-devel nss-devel gcc
   ```
6. Install Shibboleth (which includes mod_shib) and some other related packages on the machine:

   ```
   yum -y install shibboleth liblog4shib1 xmltooling-schemas opensaml-schemas
   ```
7. Install the LCG CA bundle:
   - Add EGI repo and LCG repo to the /etc/yum.repos.d/ directory:

     ```
     cd /etc/yum.repos.d/
     #install wget if you do not have it, by issuing: yum -y install wget
     wget http://repository.egi.eu/sw/production/cas/1/current/repo-files/EGI-trustanchor
     wget http://repository.egi.eu/sw/production/cas/1/current/repo-files/lcg-trustanchor
     ```
   - Create /etc/yum.repos.d/carepo.repo and insert:

     ```
     [carepo]
     name="IGTF CA Repository"
     baseurl=http://linuxsoft.cern.ch/mirror/repository.egi.eu/sw/production/cas/1/curren
     gpgkey=http://repository.egi.eu/sw/production/cas/1/GPG-KEY-EUGridPMA-RPM-3
     gpgcheck=1
     enabled=1
     ```
   - Then execute:

     ```
     yum-config-manager --enable carepo
     yum install lcg-CA
     ```
8. Create a host certificate at CERN Certification Authority,
9. Create certificate and unencrypted private keys:

   ```
   openssl pkcs12 -in hostCert.p12 -clcerts -nokeys -out hostcert.pem
   openssl pkcs12 -in hostCert.p12 -nocerts -nodes -out hostkey.pem
   ```

   and put hostcert.pem and hostkey.pem in /etc/grid-security
10. The SELinux policy has not been implemented for Shibboleth 2 therefore SELinux must be changed to run in permissive mode on your system for Single Sign On to work. For this please edit /etc/sysconfig/selinux file, and replace the line:

    ```
    SELINUX=enforcing
    ```

by

```
SELINUX=permissive
```

Next reboot your system or run:

```
/usr/sbin/setenforce Permissive
```

for the change to take effect. ( **WARNING: Notice that even if SELinux is set to permessive, it is still possible that for executing wsgi.py. So in case you want Apache to be able to run your wsgi.py you would need to add execution rights to the directory containing wsgi.py and relative super-directories of your django project** ).

11. Open ports for HTTP and HTTPS using firewalld and then reload:

```
sudo firewall-cmd --permanent --add-port=80/tcp
sudo firewall-cmd --permanent --add-port=443/tcp
sudo firewall-cmd --zone=public --add-port=8000/tcp #to allow other people to look at your
sudo firewall-cmd --reload
```

### Basic Shibboleth configuration for CERN SSO

- Copy the following files inside /etc/shibboleth/ (you can find them also here Configuration for CERN SSO in CentOS7). ( **WARNING: better to remove the pre-installed versions of shibboleth2.xml and attribute-map.xml inside /etc/shibboleth/, you should get the most recent version of those files in your machine**):
  - shibboleth2.xml (main shibboleth configuration file customized for CERN SSO).
  - ADFS-metadata.xml (ADFS configuration customized for CERN SSO)
  - attribute-map.xml (ADFS attribute mapping)
  - wsignout.gif
- Edit /etc/shibboleth/shibboleth2.xml:
  - set up the listener host (default setting of localhost should be used in most cases):

    ```
    <TCPListener address="127.0.0.1" port="1600" acl="127.0.0.1"/>
    ```
  - replace ALL 5 occurences of somehost.cern.ch, by your system hostname
- In a development environment, enable debug logging for shibd by editing /etc/shibboleth/shibd.logger:

```
log4j.rootCategory=DEBUG, shibd_log
```

  - Remember: the Shibboleth log is at /var/log/shibboleth/shibd.log/ and is under logrotate.
- Time to check the shibboleth apache configuration:
  - in /etc/httpd/conf.modules.d/ add file 00-shibd.conf:

    ```
    # Required for Shibboleth
    LoadModule mod_shib /usr/lib64/shibboleth/mod_shib_24.so
    ```
  - Inside /etc/httpd/conf.d/shib.conf, comment out the line LoadModule mod_shib /usr/lib64/shibboleth/mod_shib_24.so.
- Enable automatic startup of shibboleth daemon then restart Apache and Shibboleth:

```
/sbin/chkconfig --levels 345 shibd on
/bin/systemctl restart shibd
/bin/systemctl restart httpd
```

### Test if Apache configuration is OK

- In the VM, check out if mod_ssl and mod_shib are listed in the loaded modules list returned by:

```
httpd -M #to see the installed modules
apachectl configtest #to see if the apache configuration is faulty
/bin/systemctl status httpd #to see the status of apache
```

### Test if Shibboleth configuration is OK

- From the VM, run:

```
curl -k https://127.0.0.1/Shibboleth.sso/Status | xmllint --format -
```

- Access https://yourhostname/secure using a web browser: this is the fake location we protect for testing by default. It would be the URL of the application you want to protect. You should be redirected to your IdP (in our case it's "CERN SSO"), asked to login, and then see a "404" or "This page could not be displayed" or "this server could not verify..." error. That's because there is no real webpage at /var/www/html/secure/index.html, but you can create one. Go to https://yourhostname/Shibboleth.sso/Session with the same web browser. You should see basic login information.

### Install VOMS client-side

- **WARNING**:
  - VOMS client-side libraries are required only if one uses the VOMS authentication app.
  - the following procedure was discovered with great difficulty due to sw out of maintenance or even buggy, or hard to find (e.g. voms sw main page reports misleadingly EMI3 as a required repo for voms-clients packages on SL6 while UMD-4 is the right repo, besides we are working on Centos7 not on SL6). Moreover there is some uncertainty about the future of VOMS and even the ways of querying it (Rest APIs have been developed but not yet in production at CERN), this solution works for the time being.

- **voms-admin-client** (voms-admin command), required for voms authentication:

```
#1. Install simplejson if needed (on venv, while CC7 has the RPM from epel)
pip install simplejson
#2. download and install PyXML from sources:
wget https://github.com/actmd/PyXML/archive/0.8.4.tar.gz -O PyXML-0.8.4.tar.gz
pip install PyXML-0.8.4.tar.gz #manually fix the syntax error i.e. as = ParsedAxisSpecifie
#3. Install ZSI, allowing pre-releases:
pip install ZSI --pre
#4. Download and install voms-admin-client:
wget https://github.com/italiangrid/voms-admin-client/archive/2_0_19.tar.gz -O voms-admin-
pip install voms-admin-client-2_0_19.tar.gz
```

- **voms-clients** (for voms-proxy-init command, in case your CRIC will use proxy certificates, otherwise skip it):

```
#From EPEL repository
yum -y install bouncycastle voms-clients-java
```

## 3. CRIC ENVIRONMENT, INSTALLATION & SETUP

### Python environment

- Install required libraries:
  1. Make also sure that git is installed in your VM. If not run:

     ```
     yum -y install git
     ```
  2. Install pip, virtualenv, and setuptools:

```
yum -y install python2-pip python-virtualenv python-setuptools
```

3. Install development libraries for python, libcurl and LDAP client:

```
yum -y install python-devel openldap-devel libcurl-devel
```

4. install Django and libraries pycurl, python-ldap, and pycountry:

```
pip install "django<2" #Django 2 requires python3
##OR:
#pip install Django==1.11

pip install pycurl
##WARNING: if it doesn?t work then clean everything by uninstalling pycurl,
##exporting the library (try e.g. "openssl" or "nss" you can check it by issuing "cu
##then trying to install the package again
#pip uninstall pycurl
#export PYCURL_SSL_LIBRARY=nss
#pip install pycurl

pip install python-ldap
pip install pycountry
```

- Go to https://gitlab.cern.ch and follow the instructions to register to the service and have a SSH key (key-tape should be ecdsa) associated to it:

```
ssh-keygen -t ecdsa -C "avedaee@pic.es"
#ssh -i id_ecdsa -T git@gitlab.cern.ch
```

- Switch to CERN user: exit and re-enter your VM by using your CERN user account

### CRIC environment&installation

- The general instructions can be found in the DEPLOY file of the main CRIC repository ( **WARNING: In what follows we consider a specific deployment suitable for CMS**).
  1. Fetch sources and initialize workspace environment

```
mkdir -p /data/cric/workspace/
mkdir -p /var/log/cric/
mkdir -p /data/cric/cric.conf/web/

## initialize own virtualenv environment
virtualenv /data/cric/workspace/cric
source /data/cric/workspace/cric/bin/activate  # activate it

## fetch sources into any directory, /data/cric/git/cric as an example:
git clone ssh://git@gitlab.cern.ch:7999/cric/cric.git /data/cric/git/cric

## export location for "settings_local.py" and CMS-specific settings
export PYTHONPATH=/data/cric/cric.conf/web/:${PYTHONPATH}
export DJANGO_SETTINGS_MODULE='cric.web.cms.settings'
```

  2. Configure your /data/cric/cric.conf/web/settings_local.py which contains sensitive project configuration data (password, hostnames, etc):

```
import os
import socket

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
conf = 'DEVEL'
DEBUG = True

config = {
    'settings_path': 'cric.web.cms.settings',
    'VIRTUALENV_DIR': '/data/cric/workspace/cric',
    'SECRET_KEY': 'qk-4!b*p^++@=+p-@(@icuz^t7_&3-9zyx(^mg!',   #change this value th
    'DATABASES': {
```

```
        'default': {
            'ENGINE': 'django.db.backends.sqlite3',
            'NAME': '/data/cric/cric.conf/cricdb.sqlite3',
        }
    },
    'PYTHON_PATH': ['/data/cric/workspace/cric/lib/python2.7/site-packages', ], #'/h
    'ALLOWED_HOSTS': ['localhost', '127.0.0.1', '[::1]', socket.getfqdn()],
    'MIGRATIONS_STORE_MODULE': 'xmigrations',
    'MIGRATIONS_STORE_PATH': "/data/cric/git/cric/lib/cric/web/cms",
    'STATIC_ROOT': os.path.join(BASE_DIR, 'static'),
    'TMPDIR': os.environ.get('TMPDIR', os.environ.get('TMP', '/tmp'))
}
settings_path = config.get('settings_path')
VIRTUALENV_DIR = config.get('VIRTUALENV_DIR')
```

3. Install CRIC sources to local environment (development installation)

```
cd /data/cric/git/cric
python setup.py develop #this enables you to make changes to the sources itself insi

## use migrations to propagate Model changes into DB schema
mkdir /data/cric/git/cric/lib/cric/web/cms/xmigrations
touch /data/cric/git/cric/lib/cric/web/cms/xmigrations/__init__.py
python -m cric.web.cms.manage makemigrations --all
python -m cric.web.cms.manage migrate --run-syncdb

## command to run dev server at http://localhost:1111
#python -m cric.web.cms.manage runserver 1111
## to be visible outside local machine, e.g. http://xxhost.cern.ch:5555
#python -m cric.web.cms.manage runserver xxhost.cern.ch:5555
```

4. DB population by crons

```
## fetch sites & services from GOCDB/OIM source
python -u lib/cric/apps/cric_crons/crons/manager.py --cron OIMGOCDBInfoLoaderCron -c
## init regional centers
python -u lib/cric/apps/cric_crons/crons/manager.py --cron GStatLoaderCron -c start
## collect pledges
python -u lib/cric/apps/cric_crons/crons/manager.py --cron GStatLoaderCron -c start
python -u lib/cric/apps/cric_crons/crons/manager.py --cron GStatLoaderCron -c start
python -u lib/cric/apps/cric_crons/crons/manager.py --cron GStatLoaderCron -c start
python -u lib/cric/apps/cric_crons/crons/manager.py --cron GStatLoaderCron -c start
## collect Downtimes from GOCDB for last 90 days for ALL the sites (skip_unknown_sit
## and push data via REST API
python -u lib/cric/apps/cric_crons/crons/manager.py --cron GOCDBLoaderCron -c start
## collect (HTCondor) CE  & (local) Queues from OSG source
python -u lib/cric/apps/cric_crons/crons/manager.py --cron OSGLoaderCron -c start cr
## collect Site details, (HTCondor) CE and (local) Queues from BDII
python -u lib/cric/apps/cric_crons/crons/manager.py --cron BDIILoaderCron -c start c

## collect SiteDB information (CMS Specific collector):
python -u lib/cric/apps/cric_crons/crons/manager.py --cron SITEDBInfoLoaderCron -c s
python -u lib/cric/apps/cric_crons/crons/manager.py --cron SITEDBInfoLoaderCron -c s
## collect Glidein entries from github (CMS Specific collector):
python -u lib/cric/apps/cric_crons/crons/manager.py --cron GithubLoaderCron -c start
## collect squids information from gitlab-hosted site configuration files (CMS Speci
## Note: This collector uses the GitLab api which requires a token to run. Add your
## Instruction to get the token: https://docs.gitlab.com/ce/user/profile/personal_ac
python -u lib/cric/apps/cric_crons/crons/manager.py --cron GITLABLoaderCron -c start
```

5. Collect the static files

```
mkdir -p /data/cric/cric.conf/static/
cd /data/cric/cric.conf/static/
python -m cric.web.cms.manage collectstatic
```

### Apache configuration for CRIC

1. This is just a template (the least you can do, is to replace the keywords yourhost and yourself accordingly) and should be placed in a apache config file e.g. /etc/httpd/conf.d/cric.conf:

```
<VirtualHost *:80>
    # Redirect to HTTPS
    Redirect permanent / https://yourhost.cern.ch/
    # This secures the server from being used as a forward (third party) proxy server
    ProxyRequests Off
    ProxyPreserveHost On
    ProxyVia Block

    ### for Computer.Security @ CERN
    RewriteEngine on
    RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)
    RewriteRule .* - [F]
    # Reject request when more than 5 ranges in the Range: header.
    # CVE-2011-3192
    RewriteCond %{HTTP:range} !(^bytes=[^,]+(,[^,]+){0,4}$|^$)
</VirtualHost>

<VirtualHost *:443>
    # Enable SSL and define some host-specific settings
    SSLEngine               on
    # Allow only TLSv1.2
    SSLProtocol             all -SSLv3 -TLSv1 -TLSv1.1
    # On request from CERN security:
    # OpenSSL Ciphers: using this spec, TLSv1.2:!aNULL:!eNULL:!EXP:!DES:!RC4:!MD5:!PSK:!DH
    # but, if possible, preferring ECDSA over RSA.
    SSLCipherSuite          ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDH
    SSLHonorCipherOrder     on
    # Explicitly disable compression on the SSL level (avoid CRIME attack).
    SSLCompression          off
    SSLCertificateFile      /etc/grid-security/hostcert.pem
    SSLCertificateKeyFile   /etc/grid-security/hostkey.pem

    # Enable HSTS (mod_headers is required) (15768000 seconds = 6 months)
    Header always set Strict-Transport-Security "max-age=300; includeSubDomains"


    # This secures the server from being used as a forward (third party) proxy server
    ProxyRequests Off
    ProxyPreserveHost On
    ProxyVia Block

    ### for Computer.Security @ CERN
    RewriteEngine on
    RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)
    RewriteRule .* - [F]
    # Reject request when more than 5 ranges in the Range: header.
    # CVE-2011-3192
    RewriteCond %{HTTP:range} !(^bytes=[^,]+(,[^,]+){0,4}$|^$)

    ### SHIBBOLETH configuration
    #
    # Ensures handler will be accessible.
    #
    <Location /Shibboleth.sso>
        SetHandler shib
        AuthType None
        Require all granted
    </Location>

    <Location /accounts/sso>
        SSLRequireSSL
```

```
        AuthType shibboleth

        #THIS:
        Require shib-session
        ShibRequestSetting requireSession On
        ShibRequestSetting exportAssertion Off
        #OR:
        #ShibRequireSession On
        #Require valid-user
    </Location>
    ### WSGI configuration
    WSGIApplicationGroup %{GLOBAL}
    WSGIDaemonProcess cric user=yourself group=zp processes=5 maximum-requests=10 threads=
    WSGIProcessGroup cric
    WSGIScriptAlias / "/data/cric/git/cric/lib/cric/web/cms/wsgi.py" process-group=cric
    <Directory "/data/cric/git/cric/lib/cric/web/cms">
        <Files "wsgi.py">
            Require all granted
        </Files>
    </Directory>

    Alias /static0/             /data/cric/cric.conf/static/
    <Directory /data/cric/cric.conf/static/>
        Require all granted
    </Directory>

    LogLevel debug
    ErrorLog "logs/wsgi_error.log"
    CustomLog "logs/wsgi_access.log" combined

</VirtualHost>
```

2. Restart Apache and then you should be able to browse the CRIC main page at https://yourhost.cern.ch:

```
sudo systemctl restart httpd
```

# Security

## Apache and mod_ssl

In Apache, you should redirect all traffic to HTTPS, and make sure to use only strong cyphers and MAC algorithms. Run openssl to get them:

```
$ openssl ciphers -v 'TLSv1.2:!aNULL:!eNULL:!EXP:!RC4:!kRSA:!DH:!kECDH:+HIGH:-MEDIUM:-LOW'
ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AESGCM(256) Mac=AEAD
ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AESGCM(256) Mac=AEAD
ECDHE-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AES(256)  Mac=SHA384
ECDHE-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AES(256)  Mac=SHA384
ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AESGCM(128) Mac=AEAD
ECDHE-ECDSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AESGCM(128) Mac=AEAD
ECDHE-RSA-AES128-SHA256 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AES(128)  Mac=SHA256
ECDHE-ECDSA-AES128-SHA256 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AES(128)  Mac=SHA256
```

For improved security, prefer ECDSA over RSA authentication, i.e. the certificates carrying ECDSA keys should take precedence over the ones with RSA keys.

For an introduction to the OpenSSL and its interaction with Apache, see Mozilla SSL configuration generator. OpenSSL man page is also useful. You can also refer to The OpenSSL ccokbook for further details. This blog post is also interesting.

On the weakness of Diffie-Helmann ciphers, see this report.

Long story short, use this as a starting point:

```
<VirtualHost *:80>
# Redirect to HTTPS
Redirect permanent / https://yourhost.cern.ch/
# This secures the server from being used as a forward (third party) proxy server
ProxyRequests Off
ProxyPreserveHost On
ProxyVia Block

RewriteEngine on
RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)
RewriteRule .* - [F]
# Reject request when more than 5 ranges in the Range: header.
# CVE-2011-3192
RewriteCond %{HTTP:range} !(^bytes=[^,]+(,[^,]+){0,4}$|^$)
</VirtualHost>

<VirtualHost *:443>
# Enable SSL and define some host-specific settings
SSLEngine on
# Allow only TLSv1.2
SSLProtocol all -SSLv3 -TLSv1 -TLSv1.1
# On request from CERN security:
# OpenSSL Ciphers: using this spec, TLSv1.2:!aNULL:!eNULL:!EXP:!DES:!RC4:!MD5:!PSK:!DH:!IDEA:!kRS
# but, if possible, preferring ECDSA over RSA.
SSLCipherSuite ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA38
SSLHonorCipherOrder on
# Explicitly disable compression on the SSL level (avoid CRIME attack).
SSLCompression off
SSLCertificateFile /etc/grid-security/hostcert.pem
SSLCertificateKeyFile /etc/grid-security/hostkey.pem

# Enable HSTS (mod_headers is required)
Header always set Strict-Transport-Security "max-age=300; includeSubDomains"

# This secures the server from being used as a forward (third party) proxy server
ProxyRequests Off
ProxyPreserveHost On
ProxyVia Block

RewriteEngine on
RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)
RewriteRule .* - [F]
# Reject request when more than 5 ranges in the Range: header.
# CVE-2011-3192
RewriteCond %{HTTP:range} !(^bytes=[^,]+(,[^,]+){0,4}$|^$)
</VirtualHost>
```

## Database servers

For testing purpose only, in production we should use DBoD.

1. Erase existing packages of older releases:

   ```
   # yum list mariadb*
   # yum erase mariadb-libs
   ```
2. Append a line (otherwise dependencies might resolve to the postgresql supplied by the base repository):

   ```
   exclude=mariadb*
   ```

in /etc/yum.repos.d/CentOS-Base.repo, [base] and [updates] sections

3. Follow the instructions in Installing MariaDB with yum
4. Make sure you also install the development library:

```
yum install MariaDB-devel
```

5. Enable MariaDB:

```
# systemctl enable mariadb
```

6. Setup the MariaDB server (no need to be sudo):

```
$ /usr/bin/mysql_secure_installation
```

7. Load time zone tables:

```
$ mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root -p mysql
$ sudo systemctl restart mariadb
```

8. Create a database instance for CRIC:

```
$ mysql -u root -p mysql
MariaDB [mysql]> CREATE DATABASE cric CHARACTER SET UTF8;
MariaDB [mysql]> GRANT ALL ON cric.* TO cric@localhost IDENTIFIED BY 'mysecretpassword';
MariaDB [mysql]> quit
```

9. In the virtual environment for your CRIC project, install mysqlclient (python client for MySQL /MariaDB):

```
(venv)$ pip install mysqlclient
```

10. Read carefully and understand all the possible issues in Django configuration with MySQL here
11. Use the following configuration for the DATABASES settings:

```
'default': {
    'ENGINE': 'django.db.backends.mysql',
    'OPTIONS': {
        'init_command': "SET sql_mode='STRICT_TRANS_TABLES'",
        'isolation_level': 'read committed',
    },
    'NAME': 'cric',
    'USER': 'cric',
    'PASSWORD': 'xxxxxxx',
    'HOST': 'localhost',
}
```

These instructions are inspired from here, here, here.

1. Install the CentOS7 PGDG RPM file. A PGDG file is available for each distribution/architecture/database version combination. Browse https://yum.postgresql.org/repopackages.php and find your correct RPM. For example, to install PostgreSQL 9.6 on CentOS 7 64-bit:

```
# yum install https://download.postgresql.org/pub/repos/yum/9.6/redhat/rhel-7-x86_64/pgdg-
```

2. Edit the line (otherwise dependencies might resolve to the postgresql supplied by the base repository):

```
exclude=mariadb*,postgresql*
```

in /etc/yum.repos.d/CentOS-Base.repo, [base] and [updates] sections

3. Install the needed packages

```
# yum install postgresql96 postgresql96-libs postgresql96-contrib postgresql96-server post
```

4. Setup the PostgreSQL server

```
# export PGSETUP_INITDB_OPTIONS="--pgdata=/var/lib/pgsql/9.6/data/ -E 'UTF-8' --lc-collate=
# /usr/pgsql-9.6/bin/postgresql96-setup initdb
```

5. Start PostgreSQL:

```
# systemctl start postgresql-9.6.service
```

6. Enable PostgreSQL:

```
# systemctl enable postgresql-9.6.service
```

7. Change the password for the Linux user postgres:

```
# passwd postgres
```

8. Switch to postgres user, and set a password:

```
# su postgres
$ psql
postgres=# ALTER USER postgres WITH PASSWORD 'xxxxxx';
postgres=# \q
$ exit
```

9. Edit /var/lib/pgsql/9.6/data/pg_hba.conf by adding on top of the corresponding sections:

```
host    all             all             127.0.0.1/32            md5
host    all             all             ::1/128                 md5
```

10. Restart the service:

```
# systemctl restart postgresql-9.6
```

11. Create a database instance for CRIC:

```
# psql -h localhost -U postgres
postgres=# CREATE DATABASE cric;
postgres=# CREATE USER cric WITH PASSWORD 'xxxxxx';
## make sure the encoding is set to UTF8, and that the isolation level is at least read co
postgres=# select current_setting('client_encoding');
postgres=# select current_setting('transaction_isolation');
## if not, change them:
postgres=# ALTER ROLE cric SET client_encoding TO 'utf8';
postgres=# ALTER ROLE cric SET default_transaction_isolation TO 'read committed';
postgres=# GRANT ALL PRIVILEGES ON DATABASE cric TO cric;
```

12. In the virtual environment for your CRIC project, install psycopg2 (python client for PostgreSQL):

```
(venv)$ export PATH=$PATH:/usr/pgsql-9.6/bin
(venv)$ pip install psycopg2
```

13. use the following configuration for the DATABASES settings:

```
'default': {
    'ENGINE': 'django.db.backends.postgresql',
    'NAME': 'cric',
    'USER': 'cric',
    'PASSWORD': 'xxxxxx',
    'HOST': 'localhost',
}
```

## Make the vm accessible outside CERN.

In order to make your vm accessible from outside CERN you must fill a New Firewall Opening Request. If you go to the details of your machine (on https://openstack.cern.ch/) on bottom of the first page there is a button to access the request form. After filling the form you need to wait for the network experts to process your request.

# CRIC puppetizing

## What Puppet does

The developed puppet manifest basically follows the above instruction on manual installation. The project and the instruction are under development . **This why it is strongly recommended to to read the above instruction on manual installation before installing a puppetized virtual machine.**

The puppet manifest

- installs afs, sssd as puppet modules;
- installs httpd, mod_ssl as packages;
- installs openssl-devel, nss-devel as packages;
- sets Selinux to Permissive mode;
- installs and configures shibboleth puppet module;
- opens ports 80, 443 and 8000 using puppet module (iptables used, firewalld switched off);
- grants privileges access to berejnoi, aresh, diguida, andreeva;
- grants non-root access to berejnoi, aresh, diguida, cricserv,andreeva;
- grants access as root to all members of **cric-devs**;
- sets debugging mode in shibd.logger;
- replaces "LoadModule mod_shib /usr/lib64/shibboleth/mod_shib_24.so" from /etc/httpd/conf.d/shib.conf to /etc/httpd/conf.modules.d/00-shibd.conf;
- installs packages git, python2-pip, python-virtualenv, and mod_wsgi;
- creates directory /var/www/html/secure/;
- installs packages mod_wsgi, pycurl (using pip), Django 1.11, python-ldap (using pip), pycountry (using pip), python-devel, openldap-devel, libcurl-devel;
- puts files ssl.conf.template and cric.conf.template to the directory /etc/httpd/conf.d/ (see "Apache and mod_ssl" and "Apache configuration for CRIC");
- excludes packages mariadb* and postgresql* from /etc/yum.repos.d/CentOS-Base.repo;
- installs yumrepo for MariaDB;
- installs packages MariaDB -server, MariaDB -client, MariaDB -devel;
- starts mariabd service;
- supports httpd.service, shibd, mariabd in "running" and "enabled" state.

**MariaDB needs only for tests and will be removed from the production release.**

## What Puppet does not do

**The puppet manifest does not**

- install CRIC itself;
- configurate apache and it's ssl module for CRIC purposes (this configuration should be done and tested after the CRIC installation);
- configurate MariaDB (possible, but complicated and not useful) ;
- install PostgreSQL (the variant with using of PostgreSQL was rejected by developers and soon will be removed from the instruction);
- register your application for Shibboleth at CERN (can't);
- make the vm accessible outside CERN (can't).

As it is mentioned puppet does not configurate apache and it's ssl modules for CRIC, but it puts templates for ssl and cric configuration to files /etc/httpd/conf.d/ssl.conf.template and /etc/httpd/conf.d/cric.conf.template. These files are not tested and can be used only as starting points for CRIC configuration. Contents of these templates described in chapters "Apache configuration for CRIC" and "Apache and mod_ssl" of the

instruction on manual installation.

## Puppet repository

The puppet manifest is in the repository https://gitlab.cern.ch/ai/it-puppet-hostgroup-cric. **All members of cric-devs e-group have a root access to it.**

To clone the repository execute

```
git clone https://:@gitlab.cern.ch:8443/ai/it-puppet-hostgroup-cric.git
```

To push to the repository use

```
git pull --rebase origin qa         && git push origin qa
```

## How to create VM under puppet

To create a puppetized VM with the CIRC environment check your permissions ( http://configtraining.web.cern.ch/configtraining/introduction/permissions.html ) and follow the standard procedure described in http://configdocs.web.cern.ch/configdocs/nodes/create/.

To make a long story short just log in to aiadm.cern.ch (not from lxplus) and run a command:

```
ai-bs --cc7  --foreman-environment qa  --foreman-hostgroup  cric/cms   --landb-responsible cric-d
```

and in an hour the VM with CRIC environment will be ready.

## Some tests which should be done after VM installation

You can log in from lxplus **as root** to the installed VM and make some tests:

- insure that files exist:

```
cat /etc/httpd/conf.d/ssl.conf.template
cat /etc/httpd/conf.d/cric.conf.template
cat /etc/grid-security/hostcert.pem
cat /etc/grid-security/hostkey.pem
cat /etc/httpd/conf.modules.d/00-shib.conf
```

- try to log in to MaridDB:

```
 mysql -u root
```

- check status of services (must be running and enabled):

```
systemctl status httpd.service
systemctl status shibd
systemctl status mariadb
```

- check Shibboleth:

```
curl -k https://127.0.0.1/Shibboleth.sso/Status | xmllint --format -
```

- restart puppet agent:

```
puppet agent -t -v
```

## How to register your application for Shibboleth at CERN

If your Shibboleth works correctly, register your application for Shibboleth by filling the application register form (https://sso-management.web.cern.ch/SSO/RegisterApplication.aspx) . You should obtain an access from CERN to page https://hosname.cern.ch/secure/

## How to make the vm accessible outside CERN

In order to make your vm accessible from outside CERN you must fill a New Firewall Opening Request. If you go to the details of your machine (on https://openstack.cern.ch/) on bottom of the first page there is a button to access the request form. After filling the form you need to wait for the network experts to process your request.

AreshVedaee - 2017-04-11
SalvatoreDiGuida - 2017-02-02

---

This topic: LCG > InstructionsToCreateAVMRunningTheAuthentication
Topic revision: r73 - 2018-05-22 - AreshVedaee