Table of Contents

Deployment of CRAB TaskWorker

Complete: 🟩	Go to SWGuideCrab

All questions about TaskWorker and deploying it on private machine should go to hn-cms-crabDevelopment@cernNOSPAMPLEASE.ch

For production and pre-production: Local service account used to deploy, run and operate the service is `crab3`.

Introduction

This twiki explains how to deploy the CRAB server backend (a.k.a. the CRAB TaskWorker or TaskManager). It will guide you through the steps required to:

1. Get a virtual machine (VM) with the right architecture from the CERN OpenStack Cloud Infrastructure.
2. Install the required software on the machine.
3. Configure the machine.

Note: Legend of colors for the examples:

```
Commands to execute
```

```
Output sample of the executed commands
```

```
Configuration files
```

```
Other files
```

Get and install a virtual machine

See Deployment of CRAB REST Interface / Get and install a virtual machine. Make sure to go through the `Machine preparation` steps if you do not plan to install the REST interface on this machine.

Prepare directories and host cert to be used by the TaskWorker running as the service user

```
#sudo mkdir /data/certs  # This should have been done already by the Deploy script when installin
sudo mkdir /data/srv/TaskManager /data/certs/creds /data/srv/tmp
sudo chmod 700 /data/certs /data/certs/creds
sudo touch /data/srv/condor_config
#sudo cp -p /etc/grid-security/host{cert,key}.pem /data/certs # This should have been done alread
```

For production and pre-production installations:

```
sudo chown crab3:zh /data/srv/TaskManager /data/certs /data/certs/creds /data/srv/tmp
sudo chown crab3:zh /data/certs/host{cert,key}.pem
```

For private installations:

```
sudo chown `whoami`:zh /data/srv/TaskManager /data/certs /data/certs/creds /data/srv/tmp
sudo chown `whoami`:zh /data/certs/host{cert,key}.pem
```

Setup a service certificate for interacting with CMSWEB

A service certificate with is needed with `DN=/DC=ch/DC=cern/OU=computers/CN=tw/vocms052.cern.ch` (or whatever the correct host name is). As of 2017, this is provided yearly by James Letts, who takes care of this for all Submission Infrastructure related machines, but if needed, anybody can do it using the procedure indicated here: https://twiki.cern.ch/twiki/bin/view/CMSPublic/CompOpsGlideinWMSCerts

The service certificate must be registered in VO CMS and in SiteDB. The certificate and private key should be in `/data/certs/servicecert.pem` and `/data/certs/servicekey.pem` respectively, and should be readable only by the service user.

For production and pre-production also do:

```
sudo chown crab3:zh /data/certs/service{cert,key}.pem
```

For private installations:

```
sudo chown `whoami`:zh /data/certs/service{cert,key}.pem
```

The proxy is created for 8 days (192 hours), because this is the maximum allowed duration of the VO CMS extension. Thus, the proxy has to be renewed every 7 days (at least). You can do it manually (executing the last six commands) or you can set up an automatic renewal procedure like is being done in production and pre-production.

Install Docker daemon

Install the Docker daemon with the following commands

```
sudo yum install -y yum-utils \
  device-mapper-persistent-data \
  lvm2
sudo yum-config-manager \
  --add-repo \
  https://download.docker.com/linux/centos/docker-ce.repo
sudo yum install docker-ce
```

Then start the daemon with

```
sudo systemctl start docker
```

You can check if everything worked correctly with:

```
sudo docker ps
```

```
ONTAINER ID        IMAGE                COMMAND              CREATED              STATUS
```

Optionally you can add your user to the docker group, this will allow you to skip sudo command when using Docker cli

```
sudo usermod -aG docker your-user
```

Testing the service certificate

The certificate can be tested via command line `curl`

check VOMS:

```
curl --cert /data/certs/servicecert.pem --key /data/certs/servicekey.pem https://voms2.cern.ch:84
```

output is a longish HTML page, ugly but readable, anyhow if it works it will have something like this close to the end

```
      Your certificate:
      <ul class="certificate-info">
          <li>
            is currently linked to the following membership:
            <a href="/voms/cms/user/load.action;jsessionid=em8vmjdl2krp1k8o14htoe05z?userId=166">
              JAMES LETTS (166)
            </a>
          </li>
```

check CMSWEB:

```
curl --cert /data/certs/servicecert.pem --key /data/certs/servicekey.pem https://cmsweb.cern.ch/a
```

output will be self-explanatory

Alternatively, and in particular for private installations, you can install your own short lived proxy (starting from your own account):

```
voms-proxy-init --voms cms --valid 192:00
sudo cp /tmp/x509up_u$UID /data/certs/servicecert.pem
sudo cp /tmp/x509up_u$UID /data/certs/servicekey.pem
sudo chmod 600 /data/certs/servicecert.pem
sudo chmod 400 /data/certs/servicekey.pem
```

Contact Analysis Ops for new instances

In addition to GSI authentication, CRAB3 operators whitelist the hostnames allowed to submit jobs for CRAB3. If this is a brand-new hostname, you will need to contact CRAB operator(s) to add this TaskWorker to the whitelist.

TaskWorker: build container image and prepare configuration

Build container image

First of all clone the docker repo for the TW:

```
git clone https://gitlab.cern.ch/crab3/crab_docker.git
cd crab_docker/TaskWorker
```

Optionally you can customize the TaskWorkerConfig.py in order to prepare the image woth the correct configuration. Please note that, in any case, you'll still be able to change it at runtime. The parameters that you might have to change are:

Parameter	Type	Explanation
`TaskWorker.name`	string	A name that identifies this TaskWorker. For example the host name.
`TaskWorker.resturl`	string	host name for the `frontend` (e.g. `cmsweb.cern.ch`, `cmsweb-testbed.cern.ch`, `myprivateRESThost.cern.ch`). This parameter is incorrectly named "url" in the configu

		https://github.com/dmwm/CRABServer/blob/master/src/python/TaskWorker/MasterWo
`MyProxy.serverdn`	string	Grid host certificate DN as given by the output of `openssl x509 -noout -subject -i /data/certs/hostcert.pem &vert; cut -d ' ' -f2`

Before running the TaskWorker, double check all the parameters. If you don't know what parameters to use, please contact hn-cms-crabDevelopment@cernNOSPAMPLEASE.ch
The latest versions of all the configuration files for the TaskWorker instances in Prod & PreProd & Dev are uploaded in the same git repository as the CRAB3Rest config:
https://gitlab.cern.ch/crab3/CRAB3ServerConfig/tree/master/Taskworker/config

When you are ready, everything is setup to build the TaskWorker image locally with:

```
TW_VERSION=3.3.1909.rc1
docker build . -t mytaskworker:$TW_VERSION --build-arg TW_VERSION=$TW_VERSION
```

At the end, you should obtain a situation like:

```
docker images

REPOSITORY                                                 TAG                  IMAGE ID            CR
mytaskworker                                               $TW_VERSION         0e5bc4349856        4 d
```

Download a pre-built container image

TODO

Work with instructions in Dockerfile

https://docs.docker.com/engine/reference/builder/

Run TaskWorker with Docker

Start and stop the container

Decide and create a folder for storing tw logs (e.g. /data/myuser/logs).

To start the container:

```
docker  run --name mydev-tw -d  -ti --privileged --net host \
                 -v /etc/grid-security/:/etc/grid-security/ \
                 -v /data/certs/:/data/certs/  \
                 -v /data/myuser/logs:/data/srv/TaskManager/logs \
                 -v /etc/vomses:/etc/vomses \
                 mytaskworker:$TW_VERSION
```

If you want to bind a custom configuration file you can add the following option:

```
-v `pwd`/TaskWorkerConfig.py:/data/srv/TaskManager/current/TaskWorkerConfig.py \
```

Now check that the process is correctly running with:

```
docker ps | grep mydev-tw

b6bfc7da1620        mytaskworker:<myversion>    "/bin/sh -c 'source  "    14 seconds ago      Up 13
```

You should be able to see the logs in the folder you specified:

```
tail -f /data/myuser/logs/twlog.txt

2019-10-06 12:56:14,835:INFO:Worker,129:Process 2 is starting. PID 8627
2019-10-06 12:56:14,831:INFO:Worker,129:Process 1 is starting. PID 8626
2019-10-06 12:56:14,828:INFO:Worker,185:Started 2 slaves
2019-10-06 12:56:14,977:INFO:MasterWorker,243:Finished failing QUEUED tasks (total 0)
2019-10-06 12:56:15,185:INFO:Worker,226:Injecting work 0: TaskWorker.Actions.Recurring.RemovetmpD
2019-10-06 12:56:15,185:INFO:Worker,226:Injecting work 1: TaskWorker.Actions.Recurring.BanDestina
2019-10-06 12:56:15,191:INFO:MasterWorker,286:Master Worker status:
2019-10-06 12:56:15,191:INFO:MasterWorker,287: - free slaves: 0
2019-10-06 12:56:15,191:INFO:MasterWorker,288: - acquired tasks: 2
2019-10-06 12:56:15,191:INFO:MasterWorker,289: - tasks pending in queue: 0
2019-10-06 12:56:15,187:INFO:BaseRecurringAction,34:Executing {'tm_taskname': 'TaskWorker.Actions
2019-10-06 12:56:15,188:INFO:RemovetmpDir,18:Checking for directory older than 0 days..
2019-10-06 12:56:15,188:INFO:BaseRecurringAction,34:Executing {'tm_taskname': 'TaskWorker.Actions
```

To stop the container:

```
docker kill mydev-tw
```

You can also remove it from the saved state with:

```
docker rm mydev-tw
```

Login into the running container

To start a bash session inside the running container:

```
sudo docker exec -ti mydev-tw bash
```

You are now inside the container, as you can see with a simple ls command. Please note that the hostname might be identical to yours on the working machine, this is expected and need to keep your host certificate working.

Start and stop TW from inside the running container

To stop the running TW after you login as above you should use the preconfigured script:

```
sh stop.sh
```

Which executes the following steps:

Show Hide
first identify the pid of the master process:

```
ps faux crab3
crab3    29690  5181  0 11:06 pts/1    R+      0:00  \_ ps f -fu crab3
crab3    13459     1  4 Jul28 ?        Rl     46:03 python /data/srv/TaskManager/current/slc6_amd6
crab3    13466 13459  0 Jul28 ?        Sl      0:08  \_ python /data/srv/TaskManager/current/slc6_
crab3    13467 13459  0 Jul28 ?        Sl      0:20  \_ python /data/srv/TaskManager/current/slc6_
crab3    13469 13459  0 Jul28 ?        Sl      0:51  \_ python /data/srv/TaskManager/current/slc6_
crab3    13471 13459  0 Jul28 ?        Sl      0:15  \_ python /data/srv/TaskManager/current/slc6_
crab3    13473 13459  0 Jul28 ?        Sl      0:19  \_ python /data/srv/TaskManager/current/slc6_
```

and then kill only the master process

```
kill 13459
```

You should wait until all slaves finish.

Importat NOTE : the above procedure will make each TW slave stop when it has completed the work it pulled in its queue (2 tasks at most), so that no tasks are left in QUEUED. Tasks in status QUEUED when TW restarts would be automatically failed. But at times it may happen that one slave is instead executing one of the recurring actions (see `TaskWorkerConfig.py`), in particular the proxy renewal action can last a long time. In this case it is better not to wait (hours ?) since the proxy renewer script can be safely killed at any time. You can monitor TW winding down progress using `ps fux` and/ior using `tail` on most recent `logs/processes/proc.c3id_*.txt` files. Generally speaking it is usually safe to kill any lingering TW slave after about 5min. If they did not complete in that time, something is wrong, but you should put in the elog the name and last lines of the log for the stuck process for investigation.

To restart the tw you have to use the preconfigured script:

```
# Start the TW
sh start.sh
```

Note that this script takes care of the environmnet as well, and can be used in 4 different ways:

```
sh start.sh help

There are 4 ways to run start.sh:
  start.sh             without any argument starts current TW instance
  start.sh private     starts the TW instance from /data/user/TaskWorker
  start.sh debug       runs private instance in debub mode. For hacking
  start.sh test        runs current instance in debug mode. For finding out
BEWARE: a misppelled argument is interpreted like no argument
```

Using your own CRABServer (and WMCore) repository

Clone your repositories

Fork the dmwm/CRABServer and dmwm/WMCore repositories on github to have them under your github username (you need to have a github account) and clone them.

Note: The instructions below suggest to put the cloned repositories into the `/data/user/` directory of your host. But if you will install the TaskWorker and the REST Interface on different hosts, then you should better use another place that is accessible from both hosts, e.g. AFS.

- CRABServer: https://github.com/dmwm/CRABServer --- Fork on github ---> https://github.com/<your-github-username>/CRABServer

```
cd /data/user/
git clone https://github.com/<your-github-username>/CRABServer
cd CRABServer
git remote -v

origin   https://github.com/<your-github-username>/CRABServer (fetch)
origin   https://github.com/<your-github-username>/CRABServer (push)

git remote add upstream https://github.com/dmwm/CRABServer
git remote -v

origin   https://github.com/<your-github-username>/CRABServer (fetch)
origin   https://github.com/<your-github-username>/CRABServer (push)
upstream   https://github.com/dmwm/CRABServer (fetch)
upstream   https://github.com/dmwm/CRABServer (push)
```

- WMCore: https://github.com/dmwm/WMCore --- Fork on github --->

https://github.com/<your-github-username>/WMCore

```
cd /data/user/
git clone https://github.com/<your-github-username>/WMCore
cd WMCore
git remote -v

origin   https://github.com/<your-github-username>/WMCore (fetch)
origin   https://github.com/<your-github-username>/WMCore (push)

git remote add upstream https://github.com/dmwm/WMCore
git remote -v

origin   https://github.com/<your-github-username>/WMCore (fetch)
origin   https://github.com/<your-github-username>/WMCore (push)
upstream   https://github.com/dmwm/WMCore (fetch)
upstream   https://github.com/dmwm/WMCore (push)
```

Each `crabserver` release uses a given version of WMCore as specified in https://github.com/cms-sw/cmsdist/blob/comp_gcc493/crabtaskworker.spec Unless you will use an old tag of CRABServer, you should use the given WMCore tag:

```
git checkout <tag> # e.g. 1.0.5.pre5
```

Mount your folders inside the container

Run your TaskWorker as shown above but adding the binding for your folders that contain your repos

```
sudo docker  run --name mydev-tw -d  -ti --privileged --net host \
                -v /etc/grid-security/:/etc/grid-security/  \
                -v /data/certs/:/data/certs/ \
                -v /data/myuser/logs:/data/srv/TaskManager/logs \
                -v /etc/vomses:/etc/vomses \
                -v /data/myuser/crab_docker/TaskWorker/TaskWorkerConfig.py:/data/srv/TaskManager
                -v /data/myuser/CRABServer:/data/user/CRABServer \
                -v /data/myuser/WMCore:/data/user/WMCore \
                mytaskworker:<your version>
```

Modify the init.sh from inside the container

Once logged into the running TaskWorker container fix init script setting the `PYTHONPATH` to point to your cloned CRABServer and WMCore repositories.

Edit the file `/data/srv/TaskManager/current/slc6_amd64_gcc493/cms/crabtaskworker/*/etc/profile.d/init.sh`, comment out the lines that are changing the `PYTHONPATH` (should be the last two lines) and add a new line at the end of the script adding your repositories to the `PYTHONPATH` (here we assume the repositories are under `/data/user/`):

```
#[ ! -d /data/srv/TaskManager/3.3.16.rc2/slc6_amd64_gcc493/cms/crabtaskworker/3.3.16.rc2/${PYTHON
#[ ! -d /data/srv/TaskManager/3.3.16.rc2/slc6_amd64_gcc493/cms/crabtaskworker/3.3.16.rc2/x${PYTHO
export PYTHONPATH=/data/user/CRABServer/src/python:/data/user/WMCore/src/python:$PYTHONPATH
```

Create your own CRABServer archive

While logged into the running TaskWorker container, just use the set the desired CRABserver and WMcore version in the update.sh script. Then simply run it:

```
source update.sh
```

Restart the TW

To stop the running TW after you login as above you have to first identify the pid of the master process:

```
ps faux crab3
crab3    29690  5181  0 11:06 pts/1    R+     0:00  \_ ps f -fu crab3
crab3    13459     1  4 Jul28 ?        Rl    46:03 python /data/srv/TaskManager/current/slc6_amd6
crab3    13466 13459  0 Jul28 ?        Sl     0:08  \_ python /data/srv/TaskManager/current/slc6_
crab3    13467 13459  0 Jul28 ?        Sl     0:20  \_ python /data/srv/TaskManager/current/slc6_
crab3    13469 13459  0 Jul28 ?        Sl     0:51  \_ python /data/srv/TaskManager/current/slc6_
crab3    13471 13459  0 Jul28 ?        Sl     0:15  \_ python /data/srv/TaskManager/current/slc6_
crab3    13473 13459  0 Jul28 ?        Sl     0:19  \_ python /data/srv/TaskManager/current/slc6_
```

and then kill only the master process

```
kill 13459
```

wait until all slaves finish. To restart the tw you have to use the preconfigured scripts:

```
# Setup the environment
source env.sh
# Start the TW
sh start.sh
```

Save and push your image to docker registry

Push your image to

If you do not have a dockerHub account yet, create one at https://hub.docker.com/signup

After that, login with the docker cli

```
docker login
```

Now you have to tag your image locally:

```
docker tag mydev-tw <your Hub account>/taskworker:<your version>
```

And finally perform the upload

```
docker push <your Hub account>/taskworker:<your version>
```

Now your container is available from every machine for the download.

Push your image to CERN registry

https://gitlab.cern.ch/help/user/project/container_registry

Commit an image locally after manual changes inside the container

In the instructions above you push an image built with Dockerfile, but you can also save the status of your images after manual tweaks inside the container. **PLEASE TRY TO AVOID THIS** for cases other then temporary dev tests.

TODO

TW Docker installation

CRABDocker

TaskWorker bare metal instruction (legacy)

TaskWorker installation and configuration

Installation

Switch to the service user `crab3`:

```
sudo -u crab3 -i bash
```

Create a script (`/data/srv/TaskManager/install.sh`) to install the TaskWorker :

```
mkdir -p /data/srv/TaskWorker
cd /data/srv/TaskWorker
wget https://raw.githubusercontent.com/dmwm/CRABServer/master/src/script/Deployment/TaskWorker/in
```

Edit the faile and check/update the TaskWorker release number and the repository to pull rpm's from, e.g.:

```
export RELEASE=v3.200816
export REPO=comp  # or REPO=comp.belforte if pulling a private build
```

- Install the TaskWorker:

```
sh /data/srv/TaskManager/install.sh
```

Configuration

Copy configuration template and utility scripts from the newly installed TaskWorker

```
cp current/slc7_amd64_gcc630/cms/crabtaskworker/*/data/script/Deployment/TaskWorker/TaskWorkerCon
cp current/slc7_amd64_gcc630/cms/crabtaskworker/*/data/script/Deployment/TaskWorker/env.sh .
cp current/slc7_amd64_gcc630/cms/crabtaskworker/*/data/script/Deployment/TaskWorker/start.sh .
cp current/slc7_amd64_gcc630/cms/crabtaskworker/*/data/script/Deployment/TaskWorker/stop.sh .
cp current/slc7_amd64_gcc630/cms/crabtaskworker/*/data/script/Deployment/TaskWorker/CleanLogs.sh
```

Edit the `TaskWorkerConfig.py` file and customize it for this specific installation. The parameters that you have to change are:

Parameter	Type	Explanation
`TaskWorker.name`	string	A name that identifies this TaskWorker. For example the host name.
`TaskWorker.instance`	string	Indicates which REST instance (combination of CRABServer host and Oracle DB instance) to use

Pay attention to comments inside the `TaskWorkerConfig.py` file for more info on what/why to do

Before running the TaskWorker, double check all the parameters., then copy the file to the `current` directory

```
cp TaskWorkerConfig.py current/
```

Using your own CRABServer (and WMCore) repository

If you are a developer, most probably you want to use your own repositories.

1) Clone the repositories.

Fork the dmwm/CRABServer and dmwm/WMCore repositories on github to have them under your github username (you need to have a github account) and clone them.

Note: The instructions below suggest to put the cloned repositories into the `/data/user/` directory of your host. But if you will install the TaskWorker and the REST Interface on different hosts, then you should better use another place that is accessible from both hosts, e.g. AFS.

- CRABServer: https://github.com/dmwm/CRABServer --- Fork on github ---> https://github.com/<your-github-username>/CRABServer

```
cd /data/user/
git clone https://github.com/<your-github-username>/CRABServer
cd CRABServer
git remote -v

origin   https://github.com/<your-github-username>/CRABServer (fetch)
origin   https://github.com/<your-github-username>/CRABServer (push)

git remote add upstream https://github.com/dmwm/CRABServer
git remote -v

origin   https://github.com/<your-github-username>/CRABServer (fetch)
origin   https://github.com/<your-github-username>/CRABServer (push)
upstream   https://github.com/dmwm/CRABServer (fetch)
upstream   https://github.com/dmwm/CRABServer (push)
```

- WMCore: https://github.com/dmwm/WMCore --- Fork on github ---> https://github.com/<your-github-username>/WMCore

```
cd /data/user/
git clone https://github.com/<your-github-username>/WMCore
cd WMCore
git remote -v

origin   https://github.com/<your-github-username>/WMCore (fetch)
origin   https://github.com/<your-github-username>/WMCore (push)

git remote add upstream https://github.com/dmwm/WMCore
git remote -v

origin   https://github.com/<your-github-username>/WMCore (fetch)
origin   https://github.com/<your-github-username>/WMCore (push)
upstream   https://github.com/dmwm/WMCore (fetch)
upstream   https://github.com/dmwm/WMCore (push)
```

Each `crabserver` release uses a given version of WMCore as specified in https://github.com/cms-sw/cmsdist/blob/comp_gcc493/crabtaskworker.spec Unless you will use an old tag of CRABServer, you should use the given WMCore tag:

```
git checkout <tag> # e.g. 1.0.5.pre5
```

2) Configure the TaskWorker to use your repositories. ***This is not needed anymore since we have the options `private` and `debug` in `start.sh`, but is kept as reference info***

In the TaskWorker init script, fix the setting of the `PYTHONPATH` to point to your cloned CRABServer and WMCore repositories.

Edit the file `/data/srv/TaskManager/current/slc6_amd64_gcc493/cms/crabtaskworker/*/etc/profile.d/init.sh`, comment out the lines that are changing the `PYTHONPATH` (should be the last two lines) and add a new line at the end of the script adding your repositories to the `PYTHONPATH` (here we assume the repositories are under `/data/user/`):

```
#[ ! -d /data/srv/TaskManager/3.3.16.rc2/slc6_amd64_gcc493/cms/crabtaskworker/3.3.16.rc2/${PYTHON
#[ ! -d /data/srv/TaskManager/3.3.16.rc2/slc6_amd64_gcc493/cms/crabtaskworker/3.3.16.rc2/x${PYTHO
export PYTHONPATH=/data/user/CRABServer/src/python:/data/user/WMCore/src/python:$PYTHONPATH
```

Update the TaskWorker archives

ONLY for PRIVATE installations

Create a script (named e.g. `updateTMRuntime.sh`) with :

```
cd /data/user
cp CRABServer/src/script/Deployment/TaskWorker/updateTMRuntime.sh .
```

Run the script

- before starting the TaskWorker for the first time;
- every time you restart the TaskWorker after having done a change in the TaskWorker code.

Make sure to run it from the directory where it is locate:

```
cd /data/user
sh updateTMRuntime.sh
```

Missing `gsissh`

If `gsissh` is missing in the host machine, install it:

```
#'yum provides /usr/bin/gsissh' will tell that the package that provides /usr/bin/gsissh is gsi-o
sudo yum install gsi-openssh-clients-5.3p1-11.el6.x86_64
```

Missing `gfal-copy`

For a private installation, the `gfal-copy` command is likely to be missing in the VM, which means that the TaskWorker will have no ability to check for user's write permissions at a destination site. In that case, if a site where the user isn't authorized is selected, the TaskWorker will submit the task as usual, though it will later fail during stageout. To install `gfal-copy`, run the following command:

```
sudo yum install gfal2-util
```

Also as we never know which kind of transfer protocol a site provides/support is good to install all gfal protocol plugins:

```
sudo yum install gfal2-all
```

Start/stop the TaskWorker

- Start the service

```
sh /data/srv/TaskManager/start.sh
```

which will write `$MYTESTAREA/nohup.out`, empty if start is successful, with error messages if not.

- Stopping the service :

```
sh /data/srv/TaskManager/stop.sh
```

and wait until all slaves finish (can check with `ps fux`)

TaskWorker log files

The TaskWorker log files are in the `logs` subdirectory of the current directory from where the service is started. This subdirectory is created by the TaskWorker process if needed. If you started the service following the instructions above, the `logs` subdirectory should be in `$MYTESTAREA/logs/`. The main log file is `twlog.log`. There are also other log entries in the subdirectories `logs/processes` and `logs/tasks`. The `twlog.log` is automatically rotated every day by the service.

TaskWorker Banned sites

For sites issues check last validation twiki page Integration twiki

Install TaskWorker via Puppet

In case you want to go through the shortcut and avoid the long procedure explained above you may try to follow the automated installation process through puppet, which is explained here: CRABPuppet

---

This topic: CMSPublic > CMSCrabTaskWorker
Topic revision: r81 - 2020-08-16 - StefanoBelforte