umd-verification Documentation

Release 1.0

Pablo Orviz

Dec 14, 2018

Contents

Contents:

CHAPTER 1

Overview

1.1 Getting Started

`umd-verification` tool uses Python's Fabric library.

There is no need to build or install the application, just download the source code and interact with the tool through *fab* commands.

Note that in order to execute *fab* commands, your current path has to be the **root path of the repository** i.e. where *fabfile.py* exists.

1.2 Basic Usage

1.2.1 Listing available deployments

```
$ fab -l
Available commands:

argus                ARGUS server deployment.
argus-ees            ARGUS EES daemon deployment.
bdii-site            Site BDII deployment.
(..)
```

1.2.2 Running a deployment

Once selected the most suitable product verification (*commands* in Fabric) from the command-listing output above, one can trigger the deployment following the format:

```
$ fab <command>:<arg1>=<value1>,<arg2>=<value2>,..
```

The available runtime arguments are explained in *Runtime args* section.

Note that the only mandaatory parameter that is required at runtime is *umd_release*.

1.3 Creating a new verification

umd/products/ directory contains the *.py* files where all the available (see *Listing available deployments*) deployments are defined.

In order to create a verification for a new product, one has to instantiate *base.Deploy* class providing a given set of arguments (see the full list at *Instantiation args*):

```
from umd import base

argus = base.Deploy(
    name="argus",
    doc="ARGUS server deployment.",
    metapkg="emi-argus")
```

Fabric takes then as available commands every instance of this class found the product's directory. The command identifier is the value of *name* argument, while *doc* will contain the description of this command. This is actually the information displayed when listing commands (see *Listing available deployments*).

Note that in the case of adding a new *.py* file under *umd/products* directory, this new module has to be included in *fabfile.py* in order for Fabric to find the new command/s. Following the example above, we should add

```
from umd.products.argus import *
```

to *fabfile.py* in case that our brand new Python file is called *argus.py*.

1.4 Test execution

After a successful deployment, the last step usually involves testing that the current deployment actually works. Testing phase corresponds to EGI's `QC_FUNC_1` and `QC_FUNC_2` steps.

Test definition is placed in *etc/qc_specific.yaml*. The format of each entry is:

```
<id>:
    <qc_func_1|qc_func_2>:
        - test: <path_to_directory_or_executable_file>
          description: <test_description_string>
          user: <user_running_the_executables>
          args: <executable_arguments>
```

Things to note:

- Tests are included in the *bin/* directory within the repository. The currently available tests are described in *Testing*.
- Path (*test* parameter) can either point to a directory or to a particular executable file. In the former case all the executable files found in that directory will be executed.
- Using *args* only make sense in case of defining file paths (not directory paths).
- Environment variables can be passed to the tests at runtime (see `qcenv-*` argument at *Runtime args*).

CHAPTER 2

Argument passing

UMD product verification can be customized by providing arguments at different stages. The current available arguments and the way to pass them to the tool are explained below:

2.1 Runtime args

Runtime arguments are given through *fab* argument list. Currently supported runtime arguments are:

umd_release UMD release to be triggered.

- **Available options:**

  3 UMD-3 release.

  4 UMD-4 release.

- Default value: No default value, this parameter **is required** to be provided at runtime if `cmd_release` is not used.

cmd_release CMD release to be triggered.

- **Available options:**

  0 CMD-0 release.

- Default value: No default value, this parameter **is required** to be provided at runtime if `umd_release` is not used.

repository_url Repository path with the verification content.

- In YUM-based systems the URL MUST point to where `repodata` directory is located.
- Multiple values are allowed by prefixing with *repository_url*.

```
fab repository_url=<URL1>,repository_url_2=<URL2>,repository_url_
↪other=<URL3>,..
```

- Arguments passed with equal names will overwrite the value.

repository_file URL pointing to a valid repository file (.list, .repo).

- Multiple values are allowed by prefixing with *repository_file*.

```
fab repository_file=<URL1>,repository_file_2=<URL2>,repository_file_
↪other=<URL3>,..
```

- Arguments passed with equal names will overwrite the value.

igtf_repo Repository for the IGTF release.

- Value must contain a URL pointing to a valid repository file.
- **Required** value located in the default configuration file (see *Static args*).

yaim_path Path pointing to YAIM configuration files.

- Default value: `etc/yaim/`.

log_path Path to store logs produced during the execution.

- Default value: `/var/tmp/umd-verification`.

qcenv_* Pass environment variables needed by the QC specific checks.

- The name of the environment variable to be exported is the name given after the underscore '_' symbol. Accordingly, the variable's value is the *fab* argument's value.

```
fab qcenv_FOO=bar,..
```

This example will set `FOO=bar` in the testing environment.

qc_step Run a given set of Quality Criteria steps.

- Multiple values are allowed by prefixing with *qc_step*.

```
fab qc_step_1=QC_SEC,qc_step_2=QC_INFO,..
```

- Arguments passed with equal names will overwrite the value.

umdnsu_url URL (hostname:port) to interface with *umdnsu* service running in the SAM-Nagios instance.

hostcert Public key server certificate.

hostkey Private key server certificate.

dont_ask_cert_renewal Do not prompt for certificate renewal (when certificates already exist)

ca_version Special runtime argument for CA verifications. This value refers to the CA release version with '<major>.<minor>.<patch>' format.

enable_testing_repo Enable UMD or CMD testing repository.

enable_untested_repo Enable UMD or CMD untested repository.

params_file YAML file with extra parameters to be passed to the configuration management tool (Ansible, Puppet)

2.2 Static args

An additional way to provide the runtime arguments seen above is through the configuration file *etc/defaults.yaml*.

This file *must* exist since it is here where the *required* arguments are set. This is why it lives within the application codebase.

The format is YAML so the naming of the runtime arguments seen above differ a little. Currently supported runtime arguments (and their YAML formatted equivalent) are:

base:log_path `log_path` argument.

umd_release:<distro_version (e.g. redhat5)> `umd_release` argument.

igtf_repo:<distname (e.g. redhat)> `igtf_repo`.

yaim:path `yaim_path`.

nagios:umdnsu_url `umdnsu_url`.

2.3 Instantiation args

These arguments are used when defining a new deployment (`umd.base.Deploy` instance) in the product's directory *umd/products*. Currently supported instantiation arguments are:

name UMD product (aka Fabric command name).

- Type: `str`.
- Default value: empty string.

doc Docstring that will appear when typing *fab -l*.

- Type: `str`.
- Default value: empty string.

need_cert Whether installation type requires a signed cert.

- Type: `boolean`.
- Default value: `False`.
- Additional info: creates a dummy CA to issue public and private keys needed for the product to be deployed.

has_infomodel Whether the product publishes information about itself.

- Type: `boolean`.
- Default value: `False`.
- Additional info: launches QC_INFO_1 checks, so it's mandatory for the product publishing data (commonly through BDII).

cfgtool Configuration tool object.

- Type: `umd.base.configure.BaseConfig`.
- Default value: `None`.
- Additional info: contains an instance of any class that inherits from BaseConfig. Currently available: - `umd.base.configure.YaimConfig`

    nodetype YAIM nodetype to be configured.

    siteinfo File containing YAIM configuration variables.

  - `umd.base.configure.PuppetConfig`

manifest Main ".pp" with the configuration to be applied.

module_from_puppetforge list of modules to be installed (from PuppetForge).

module_from_repository module (repotype, repourl) tuples.

module_path Extra Puppet module locations.

qc_mon_capable

Whether extenal monitoring (aka SAM Nagios) can monitor the product.

- Type: `boolean`.
- Default value: `False`.

qc_specific_id

ID that match the list of QC-specific checks to be executed. The check definition must be included in *etc/qc_specific.yaml*.

- Type: `str`.
- Default value: `None`.

qc_step Specific step from the Quality Criteria to run.

- Type: `str`, `list`.
- Default value: empty list.

exceptions Documented exceptions for a given UMD product.

- Type: `dict`.
- Default value: empty dict.

CHAPTER 3

Testing

This page documents the tests included with the `umd-verification` tool. Note that the tool allows to execute whatever custom checks located in the system.

3.1 *bin/myproxy/client-test.sh*

Retrieves a proxy with VOMS extension from a MyProxy server.

- Accepted arguments: `retrieve`
- Currently supported VOs: *ops.vo.ibergrid.eu*, *dteam*
- Environment variables needed:
  - `MYPROXY_SERVER`
  - `MYPROXY_USER`
  - `MYPROXY_PASSWD`

  and the optional:
  - `VO`

  that defaults to `ops.vo.ibergrid.eu` VO.

  These variables need to be passed via runtime arguments (as specified in *Test execution*):

```
fab ui:qcenv_MYPROXY_SERVER=foo.example.org,qcenv_MYPROXY_USER=bar,qcenv_MYPROXY_
↪PASSWD=baz,..
```

- Requires a myproxy already stored in `MYPROXY_SERVER` with user and password credentials:

```
echo $MYPROXY_PASSWD | myproxy-init -S -l $MYPROXY_USER -s $MYPROXY_SERVER -m $VO
```

3.2 *bin/srm/client-test.sh*

Performs data management checks.

- Accepted arguments:
  - #1 Whether the endpoint is localhost or an external one.
    - Valid values: `localhost`, `storm`, `dpm`, `dcache`
  - #2 Client to be tested.
    - Valid values: `lcg-util`, `dcache-client`, `gfal2-python`, `gfal2-util`
- Environment variables:
  - `SRM_HOST` *optional*
  - `SRM_ENDPOINT` *optional*

  points to the SRM URL following the format *srm://<srm_host>/<srm_vo_path>*

```
fab ui:qcenv_SRM_ENDPOINT="srm://srm01.ifca.es:8444/srm/managerv2?SFN=/ops.vo.
↪ibergrid.eu"
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search