MiG User Scripts

Last updated 07-10-2009 16:25

Contents

1 Introduction

MiG provides two user interfaces for all user interaction with MiG: The non-intrusive web interface most appropriate when handling a limited number of jobs and files, and a set of scripts that are more suitable for big projects with too many jobs and files to handle manually.

This document focuses on the MiG scripts, and includes examples of the basic usage.

2 Prerequisites

In order to access MiG you need a MiG key/certificate pair: please contact us if you haven't already been handed one.

The user scripts rely heavily on curl for communication with the MiG servers. Any recent version of curl should work without problems, as long as ssl/tls support is included. Old curl versions prior to 7.10, didn't support the *–insecure* flag, which is used if no MiG CA certificate is available. Similarly the *–create-dirs* flag wasn't available back then. Any problems related to those two flags not being available can be circumvented by using a simple wrapper that just filters out the flags, since they are already implicitly set in those old versions. Please contact us for more details if needed.

Currently the scripts are available in two scripting languages - *python* and *sh*. The python scripts provide platform independent script access to MiG, but may require the user to install python if not already available. The sh scripts on the other hand are un*x only, but seldom require anything to be installed as an sh-interpreter is normally readily available. The sh-scripts are intended to be *pure* sh, but our testing is somewhat limited to bash and zsh, so we may have missed some bash'isms. Again, please feel free to report any problems.

3 Obtaining the scripts

All MiG scripts are provided on a *on demand* basis through a generator cgi script. To obtain the latest version, simply point your browser to user-scripts . It is possible to pass options to the generator in order to change

the generated script languages and default application paths. Go to the usage-help to get help on using these options. After running the generator successfully, the output will include links to the location of the generated scripts. Simply select the zip link to download a zip archive of the scripts. Unpack the archive and you're ready to start using the scripts.

4 Configuration

You receive information about the address of the MiG server that you should use along with the key/certificate pair. In order to use the scripts you need to create a configuration file including the server address and paths to your key and certificate. The default configuration is location is ~/.mig/miguser.conf, but you can override it with the -c flag to all user scripts. After saving the certificate and key in ~/.mig/cert.pem and ~/.mig/key.pem an example configuration could look like:

```
migserver https://MIG-SERVER
certfile /home/LOGIN/.mig/cert.pem
keyfile /home/LOGIN/.mig/key.pem
```

With that configuration each mig user script execution will ask for your passphrase. If you wish to avoid typing the passphrase you can add a password line to your configuration:

```
password MYPASSPHRASE
```

IMPORTANT NOTICE: this will allow anybody with read access to the configuration file to read your password!! Please make sure to protect the configuration file with suitable read permission if you choose to include the password.

Additionally it is possible to verify the identity of the MiG server by providing a CA certificate path in the configuration:

```
cacertfile /home/LOGIN/.mig/MiG-CA.crt
```

Please contact us if you haven't been given a copy of the CA certificate. The CA certificate file may be called either MiG-CA.crt or cacert.pem depending on the source.

5 Using the scripts

First of all change directory to the unpacked MiG-user-scripts directory. In general all MiG scripts include a short usage help which is shown if the script is called with the *-h* flag or if incorrect syntax is used. In the following sections each of the MiG user scripts are shown in action. Each section contains examples using both the sh and python version. The two versions work in a very similar way under the hood, so the example output can be expected to be identical or very similar. Thus only one output example is shown for each operation.

5.1 submit a job

This example uses a very simple job described in the file hellogrid.mRSL:

```
cat hellogrid.mRSL
::EXECUTE::
echo "Hello grid"
```

After creating the job description file it can be submitted:

- python

  ```
  python migsubmit.py hellogrid.mRSL
  ```

- sh

  ```
  ./migsubmit.sh hellogrid.mRSL
  ```

Example output:

```
0

2452_11_15_2005__15_54_23_mig-1.imada.sdu.dk.0 is the job id assigned.
```

Now the job can be referenced by the unique string 2452_11_15_2005__15_54_23_mig-1.imada.sdu.dk.0 in status and cancel requests.

5.2 show the status of a job

- python

  ```
  python migstatus.py 2452_11_15_2005__15_54_23_mig-1.imada.sdu.dk.0
  ```

- sh

  ```
  ./migstatus.sh 2452_11_15_2005__15_54_23_mig-1.imada.sdu.dk.0
  ```

Example output:

```
0


Status: EXECUTING
Received: Tue Nov 15 15:59:19 2005
Queued: Tue Nov 15 15:59:19 2005
Executing: Tue Nov 15 15:59:21 2005
```

In this case the job has moved to the EXECUTING state meaning that it has been sent to a resource for execution.

5.3 show the status of all submitted MiG jobs

- python

  ```
  python migstatus.py
  ```

- sh

  ```
  ./migstatus.sh
  ```

Example output:

```
0

Status: FINISHED
Verified: NO
Received: Tue Nov 15 15:59:19 2005
Queued: Tue Nov 15 15:59:19 2005
Executing: Tue Nov 15 15:59:21 2005
Finished: Tue Nov 15 15:59:55 2005
```

In this case the job is FINISHED executing so any results are now available in the home directory. The 'NO' in the Verified field of the job indicates that the job didn't include any templates to verify the output and exit values against.

5.4 cancel a job

- python

  ```
  python migcancel.py 2453_11_15_2005__15_59_19_mig-1.imada.sdu.dk.0
  ```

- sh

  ```
  ./migcancel.sh 2453_11_15_2005__15_59_19_mig-1.imada.sdu.dk.0
  ```

Example output:

```
1

Trying to cancel job with job
id:2453_11_15_2005__15_59_19_mig-1.imada.sdu.dk.0

You can only cancel jobs with QUEUED or RETRY status, status of job
you are trying to cancel: FINISHED
```

As seen above the cancel failed with exit code 1 since the job already finished. Furthermore the output indicates that only jobs that are queued can be cancelled. Executing jobs can not be cancelled either. Please refer to the kill section for information about stopping jobs during execution.

5.5 list the contents of your MiG home directory

- python

  ```
  python migls.py
  ```

- sh

  ```
  ./migls.sh
  ```

Example output:

```
0

.:
.
..
hellogrid.mRSL
2453_11_15_2005__15_59_19_mig-1.imada.sdu.dk.0.stdout
2453_11_15_2005__15_59_19_mig-1.imada.sdu.dk.0.status
```

The zero tells us that the operation succeeded and the rest are the actual contents of the MiG home directory. hellogrid.mRSL is the job description file we implicitly uploaded in the submit script. he other two files contain the stdout and exit codes of that job.

5.6 concatenate and print file contents

- python

  ```
  python migcat.py '*'
  ```

- sh

  ```
  ./migcat.sh '*'
  ```

Example output:

```
::EXECUTE::
echo "Hello grid"
"Hello grid"
echo "Hello grid" 0
```

The first two lines are from the job description, while the two last lines are from the stdout and status files respectively. In the status file it is indicated that the exit code of the echo command was 0.

5.7 download files

- python

  ```
  python migget.py '2453_11_15_2005__15_59_19_mig-1.imada.sdu.dk.0*' .
  ```

- sh

```
./migget.sh '2453_11_15_2005__15_59_19_mig-1.imada.sdu.dk.0*' .
```

Example output:

Now the two files with output and exit codes from the previous job are locally available. Please note that file operation like migget support wild cards in remote filenames, but that it may be necessary to escape or quote the names with wild cards in them to avoid local shell wild card expansion.

5.8 upload files

In this example we create a local file and upload it to the MiG home.

```
touch somefile
for i in `seq 1 100`; do
    echo "test line $i" >> somefile;
done
```

- python

  ```
  python migput.py somefile .
  ```

- sh

  ```
  ./migput.sh somefile .
  ```

Example output:

```
0
```

```
A 'normal' file was uploaded (/./somefile). It can now be used as an
inputfile in your .mRSL files.
```

5.9 print first lines of files

- python

  ```
  python mighead.py somefile
  ```

- sh

```
./mighead.sh somefile
```

Example output:

```
0

test line 1
test line 2
test line 3
test line 4
test line 5
test line 6
test line 7
test line 8
test line 9
test line 10
test line 11
test line 12
test line 13
test line 14
test line 15
test line 16
test line 17
test line 18
test line 19
test line 20
```

The output contains the first 20 lines of the uploaded file, somefile.

5.10 print last lines of files

- python

```
python migtail.py somefile
```

- sh

```
./migtail.sh somefile
```

Example output:

```
0

test line 81
test line 82
test line 83
test line 84
test line 85
test line 86
test line 87
test line 88
test line 89
test line 90
test line 91
test line 92
test line 93
test line 94
test line 95
test line 96
test line 97
test line 98
test line 99
test line 100
```

The output contains the last 20 lines of the uploaded file, somefile.

5.11 delete files

- python

  ```
  python migrm.py somefile
  ```

- sh

  ```
  ./migrm.sh somefile
  ```

Example output:

```
0
```

A subsequent migls will show that somefile is no longer available in the MiG home directory.

5.12 show online documentation

Run migdoc without arguments to get a list of available topics:

- python

  ```
  python migdoc.py
  ```

- sh

  ```
  ./migdoc.sh
  ```

Example output:

```
Documentation topics:
Resource configuration
Job description: mRSL
```

To see the documentation for one or more of the topics, run the script with some part of those topic titles as argument(s).

- python

  ```
  python migdoc.py mrsl conf
  ```

- sh

  ```
  ./migdoc.sh mrsl conf
  ```

Example output:

```
Documentation:
Job description: mRSL

ARCHITECTURE
Required: False
```

```
Type: string
Description: i386/.., 64bit
Value:
Example: i386

CPUCOUNT
Required: False
Type: int
Description: Number of CPU's the job requires on each node.
Value: 1
Example: 4

CPUTIME
Required: False
Type: int
Description: The time required to execute the job. The time is specified
in !!
Value: 0
Example: 60

[ ...  ]


Documentation:
Resource configuration

ARCHITECTURE
Required: True
Type: string
Description: The resource's architecture (eg. i386).
Value:
Example: i386

CPUCOUNT
Required: True
Type: int
Description: Number of CPU's on each node.
Value: 1
Example: 4

[ ... ]
```

5.13 flags

All the scripts support a few common flags in order to provide a more flexible operation. In addition to those flags some of the scripts support a number of extra flags only relevant to that particular script. All the available flags for a particular script are shown when the script is run with the *-h* flag:

- python

  ```
  python migls.py -h
  ```

- sh

  ```
  ./migls.sh -h
  ```

To use a configuration in a non-standard location use the *-c* flag:

- python

  ```
  python migls.py -c my-mig.conf
  ```

- sh

  ```
  ./migls.sh -c my-mig.conf
  ```

The *-s* flag can be used to override the *migserver* line in the configuration:

- python

  ```
  python migls.py -s "https://mig-2.imada.sdu.dk"
  ```

- sh

  ```
  ./migls.sh -s "https://mig-2.imada.sdu.dk"
  ```

Use the *-V* flag to find out which version of the scripts you have:

- python

  ```
  python migls.py -V
  ```

- sh

```
./migls.sh -V
```

Use the *-v* flag to turn on more verbose output:

- python

  ```
  python migls.py -v
  ```

- sh

  ```
  ./migls.sh -v
  ```

6 Changing key/certificate passphrase

You need openssl in order to change the passphrase on your key and certificates.

To create a copy of the key with a new passphrase:

```
openssl rsa -in myMiGkey.pem -des3 -out myMiGkey.pem.new
[enter old passphrase]
[enter new passphrase]
[verify new passphrase]
```

Now you can use the new key (and thus the new passphrase) along with the original pem certificate for the MiG scripts.

To create a copy of the P12 certificate with a new import password:

```
openssl pkcs12 -export -in myMiGcert.pem -inkey myMiGkey.pem -out myMiGcert.p12.new
[enter key passphrase]
[enter new export password]
[verify new export password]
```

Now you can import the new P12 certificate using the new password. In case you wish to change both the key and P12 passwords, you should replace the -inkey argument in the latter command with the path to the newly created key.

7 Feedback

Please send any feedback to the MiG mailing list at mig@imada.sdu.dk