

Software Installation

Last review date

2009-09-15

Reviewer

Marco Bencivenni
Enrico Fattibene



Table of Contents

[Software Installation](#)

[Common Setup](#)

[Sandbox Method](#)

[Parachute Method](#)

[Shared Area Method](#)

Software Installation

Most jobs require some application- specific software installed on the machine being used. There are three methods for installing application- specific software on the grid: the "Sandbox" method, the "Parachute" method, and the "Shared Area" method. Each method has its own advantages and disadvantages.

Common Setup

To demonstrate these methods, we will create a toy software system consisting of one shared library and one executable. We will implement a single function `say_hello()` that will write a greeting to the standard output.

Create the header file (named `hello.h`) with the definition of this function; this file is used by both the executable and shared library.

```
/* hello.h */
void say_hello(char* name);
```

Next create the file containing the implementation of this function; name this file `libhello.c`.

```
/* libhello.c */

#include "hello.h"
#include <stdio.h>

void say_hello(char* echo_string) {
    printf("Hello %s\n", echo_string);
}
```

Finally, create the main function that will call this function; copy the following into `say_hello.c`.

```
/* say_hello.c */

#include "hello.h"

int main(int argc, char *argv[]) {
    if (argc==2) {
        say_hello(argv[1]);
    } else {
        printf("Usage: Hello echo-string\n");
    }
    return 0;
}
```

Now build the sharable library and executable. The following assumes that you are using the GNU compiler on a linux- based operating system. (You should prepare binaries for a RedHat compatible operating system as those are the only ones currently available on the grid infrastructure.)

```
$ gcc -shared -o libhello.so libhello.c
$ gcc -o say_hello -L. -lhello say_hello.c
```

After these commands you should have two new files: `say_hello` and `libhello.so`. You can verify that the executable works with the following commands:

```
$ export LD_LIBRARY_PATH=`pwd`
$ ./say_hello HappyGridUser
```

This should print "Hello HappyGridUser" to the console. If you are not using a bourne shell (or compatible) you will need to change the first command to correspond to your shell.

Although in this case the number of files is small, in practice there are often a large number of files in the software package you create and want to use on the grid. We will bundle all of the necessary files into a single tarball.

```
$ mkdir hello
$ mv say_hello libhello.so hello/
$ tar zcf hello.tgz hello/
```

We will then transfer the tarball and unpack and use it on the remote resource. Verify that the file `hello.tgz` has been created and that it contains the necessary files.

Sandbox Method

The Sandbox Method is the simplest of the three techniques for software installation; it is available to all members of a Virtual Organization. All of the necessary software will be transferred with the job itself. This mechanism stages the software to the WMS server for *each* job. Thus, this method is *only* appropriate when the total amount of code is small (tens of megabytes, maximum) and the total number of simultaneous running jobs is small.

Preparing the Job

Prepare a script (`hello_sandbox.sh`) that will launch our executable.

```
#!/bin/sh
tar zxf hello.tgz
export LD_LIBRARY_PATH=`pwd`/hello
export PATH=`pwd`/hello:${PATH}
./hello/say_hello HappyGridUser
```

Prepare a job description called `hello_sandbox.jdl` like the following.

```
Executable      = "hello_sandbox.sh";
StdOutput       = "std.out";
StdError        = "std.err";
InputSandbox    = {"hello_sandbox.sh", "hello.tgz"};
OutputSandbox   = {"std.out", "std.err"};
Requirements    = other.GlueHostOperatingSystemName=="Scientific Linux"
#               && other.GlueHostArchitecturePlatformType=="i686"
;
```

Since we have created precompiled executables and libraries, it is important to specify the target operating system and architecture for the job. In this case, "Scientific Linux" and "i386", respectively, have been specified. These properties (especially the architecture) are not uniformly published; so unfortunately, this may overly restrict the number of matching resources. You will need to balance the restrictions against the number of jobs that could end up on inappropriate resources. The architecture restriction is commented out because the intel architecture (or compatible) is nearly universally deployed on the grid infrastructure.

Running the Job

To run the job, initialize a proxy and then use the usual WMS commands: `glite-wms-job-submit`, `glite-wms-job-status`, and `glite-wms-job-output`. The script sent along with the job will untar the `hello.tgz` tarball, set the required load path, and then run the `hello` executable. After recovering the output, you should see that `std.out` contains the following:

```
Hello HappyGridUser
```

and that the file `std.err` is empty.

Summary

This example used the standard WMS sandbox mechanism to transfer a software package to a worker node and to use that package from the job. This mechanism has the benefit of being simple for the user. However, because the sandbox files are transferred twice and because the cache space on the WMS is limited, this mechanism should only be used for small packages (tens of megabytes, maximum) and for small numbers of jobs. Larger packages or many jobs that use the same software should try the "Parachute" or "Shared Area" techniques described below.

Parachute Method

For the Parachute Method, we copy the software to a storage element on the grid avoiding the WMS sandbox caching. Each job begins by retrieving the software and installing it.

Registering Software a Storage Element

The first step of this process is the creation of the software package, such as was done above with the `hello.tgz` tarball. The next step is to copy this tarball into grid storage. We use the standard `lcg-cr` command to do this. You must replace the VO name ("`vo.lal.in2p3.fr`" here) with your own.

```
$ lcg-cr --vo vo.lal.in2p3.fr \  
-l lfn:/grid/vo.lal.in2p3.fr/hello.tgz \  
file:`pwd`/hello.tgz
```

```
guid:8d4538cd-a672-4430-821c-cea246f2d9ba
```

Each job will then retrieve this tarball, unpack it, and then run the job.

Preparing the Job

Now that the tarball has been uploaded to a storage element, prepare a script that will launch our executable called `hello_parachute.sh`. Remember to replace the VO name with your own.

```
#!/bin/sh  
lcg-cp --vo vo.lal.in2p3.fr \  
lfn:/grid/vo.lal.in2p3.fr/hello.tgz \  
file:`pwd`/hello.tgz  
  
tar xzf hello.tgz  
export LD_LIBRARY_PATH=`pwd`/hello  
export PATH=`pwd`/hello:${PATH}  
./hello/say_hello HappyGridUser
```

Notice that the first action is to make a local copy of the tarball. After that the tarball is expanded and the executable is run.

Prepare a job description called `hello_parachute.jdl` like the following.

```
Executable      = "hello_parachute.sh";  
StdOutput       = "std.out";  
StdError        = "std.err";  
InputSandbox    = {"hello_parachute.sh"};  
OutputSandbox   = {"std.out", "std.err"};  
Requirements    = other.GlueHostOperatingSystemName=="Scientific Linux"  
#               && other.GlueHostArchitecturePlatformType=="i686"  
;
```

This is identical to the JDL file for the Sandbox Method, except that the script name has changed.

Running the Job

To run the job, initialize a proxy and then use the usual WMS commands: `glite-wms-job-submit`, `glite-wms-job-status`, and `glite-wms-job-output`. The script sent along with the job will download and untar the `hello.tgz` tarball, set the required load path, and then run the `hello` executable. After recovering the output, you should see that `std.out` contains the following:

```
Hello HappyGridUser
```

and that the file `std.err` is empty.

Summary

This example used the Parachute Method for software installation. This method manually caches the required software package in a grid storage element; each job then retrieves the software, unpacks it, and uses it. This is much more efficient than the Sandbox Method when several jobs need to use the same software package. This method can be used by any member of a Virtual Organization. The only inconvenience of this method is that each job must spend some time downloading and installing the software.

Shared Area Method

Site administrators provide a shared area for pre-installing application-specific software for each supported VO. This area can be read by any member of a given VO, but can only be written by the Software Manager(s) of the VO. This method avoids the overheads with the repeated download and installation of software packages associated with the parachute method. It requires, however, coordination within the VO and effort to manage the software installation at multiple sites. It is important to note that the VO's Software Manager, *not the site administrator*, is responsible for the installation, testing, and maintenance of the software within the VO's shared area.

There are several phases when using the Shared Area Method: 1) the VO Software Manager installs the software on selected sites, 2) the Software Manager creates a runtime environment tag, and 3) users run jobs using the pre-installed software.

Preparing the Installation Job

We will use the tarball that has been uploaded to the storage element for the Parachute Method above. Prepare a script that will install our software on a selected site called `hello_shared_install.sh`. Remember to replace the VO name with your own.

```
#!/bin/sh

# change to shared software area
cd $VO_VO_LAL_IN2P3_FR_SW_DIR

# recover a copy of the software
lcg-cp --vo vo.lal.in2p3.fr \
  lfn:/grid/vo.lal.in2p3.fr/hello.tgz \
  file:`pwd`/hello.tgz

# unroll it, clean up
tar xzf hello.tgz
rm -f hello.tgz
```

The location of the shared area may be different on different sites. The location can be reliably located with an environment variable of the form: `VO_<voname>_SW_DIR`, where the `voname` is replaced by the name of the VO. *Because the value must be a valid shell variable, you must replace any periods or hyphens by an underscore.* For example, the variable name for the VO "vo.lal.in2p3.fr" is "VO_VO_LAL_IN2P3_FR_SW_DIR".

Prepare a job description called `hello_shared_install.jdl` like the following.

```
Executable = "hello_shared_install.sh";
StdOutput  = "std.out";
StdError   = "std.err";
```

```
InputSandbox = {"hello_shared_install.sh"};
OutputSandbox = {"std.out", "std.err"};
Requirements = RegExp(".*grid10.lal.in2p3.fr.*", other.GlueCEUniqueID);
```

In this case, you will want to identify one or more computing elements that provide resources for your VO and that meet your software requirements. In the "Requirements" statement, you will want to target specifically those resources. (You can also use the `-r` option for `glite-wms-job-submit`.)

Running the Installation Job

To have write access to the shared area, you must be authorized as a Software Manager. Most VOs have a group identified with this role named "lcgadmin". If you are a member of that group, you can initialize a proxy using this role. For example:

```
$ voms-proxy-init --voms vo.lal.in2p3.fr:/vo.lal.in2p3.fr/Role=lcgadmin
```

of course substituting your VO and the proper Software Manager role for your VO.

Using the proxy with the role information, you can use the usual WMS commands (`glite-wms-job-submit`, `glite-wms-job-status`, and `glite-wms-job-output`) to run the job. You should check the output carefully for any errors.

Publishing a Tag

You will want to publish a "runtime environment tag" for the CE's that have your software installed. These tags allow the users to identify which computing resources have your software pre-installed. Each tag must have a prefix of the form: "VO- <voname>". The content after the prefix is free (but don't use whitespace or characters that are likely to cause problems). In this example, the created tag is: "VO-vo.lal.in2p3.fr-hello".

You (or your colleagues) may want to publish several tags for a given CE. It is important that you preserve existing tags, otherwise users will no longer think that the software identified by other tags is still installed. Like for the shared area itself, the VO Software Manager(s) is (are) responsible for maintaining the coherence between the list of tags and the installed software. The list of tags is in a file named `/opt/edg/var/info/<voname>/<voname>.list`.

Create the following script named `hello_shared_tag.sh`:

```
#!/bin/sh

# location of tag file and definition of tag
taglist=/opt/edg/var/info/vo.lal.in2p3.fr/vo.lal.in2p3.fr.list
tag=VO-vo.lal.in2p3.fr-hello

# publish software tag
touch ${taglist}
cp ${taglist} ${taglist}.bak
cat - ${taglist} <<EOF | sort -u > ${taglist}.new
${tag}
EOF
mv ${taglist}.new ${taglist}
```

This script must run directly on the computing element on which you have installed your software. The easiest way to do this is to run the script directly with the `globus-job-run` command: `globus-job-run`. You must use the proxy with the Software Manager role information embedded in it.

```
$ globus-job-run grid10.lal.in2p3.fr:2119/jobmanager-fork -s hello_shared_tag.sh
```

You must use the "fork" job manager that will run the job directly on the computing element itself. This is an interactive command so any output or errors will be streamed to the console.

If there were no errors, you should see your tag appear in the information system for the given computing element. You can check this with the following command:

```
$ ldapsearch -LLL -H ldap://topbdii.grif.fr:2170 \
-x -b 'mds-vo-name=local,o=grid' \
```

```
'(GlueHostApplicationSoftwareRunTimeEnvironment=VO-vo.lal.in2p3.fr-hello)' \
GlueChunkKey
```

Note that it may take 10 minutes or so for the tag to be published and to fully propagate through the information system. Choose your favorite entry point to the information system and be sure to replace the tag with the one appropriate to your VO. This command will list the resources with the given tag; check that the resource with your newly installed software is listed.

Running a User Job

Prepare a script (`hello_shared_run.sh`) that will launch our executable.

```
#!/bin/sh

# location of software
hellodir=${VO_VO_LAL_IN2P3_FR_SW_DIR}/hello

# setup the environment: PATH and LD_LIBRARY_PATH
if [ -z ${PATH} ]; then
PATH="${hellodir}"
else
PATH="${hellodir}:${PATH}"
fi
export PATH

if [ -z ${LD_LIBRARY_PATH} ]; then
LD_LIBRARY_PATH="${hellodir}"
else
LD_LIBRARY_PATH="${hellodir}:${LD_LIBRARY_PATH}"
fi
export LD_LIBRARY_PATH

# run the software
say_hello HappyGridUser
```

This script sets up the user's environment to make the "hello" package visible. It will then simply run the `hello` executable. Prepare a job description called `hello_shared_run.jdl` like the following.

```
Executable      = "hello_shared_run.sh";
StdOutput       = "std.out";
StdError        = "std.err";
InputSandbox    = {"hello_shared_run.sh"};
OutputSandbox   = {"std.out", "std.err"};
Requirements    = Member("VO-vo.lal.in2p3.fr-hello",
                        other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

Note that the job requirements line now uses the published tag to identify suitable resources. Using `glite-wms-job-list-match` should show the resources with the software installed. The job should run normally with the usual `glite-wms-*` commands. A normal proxy without the Software Manager role can use the installed software.

Summary

This example used the Shared Area Method for software installation. This requires that a Software Manager for a VO pre- install software on a set of computing elements. After the software has been installed, other members of the VO can access this software easily. This simplifies the normal user jobs, but requires more effort on the part of the VO to manage its software.