User-friendly access to Grid and Cloud resources for scientific computing

Dr. Alexander Richards

Imperial College Sci., Tech. & Med. UK
(IC)

18th – 19th January 2016

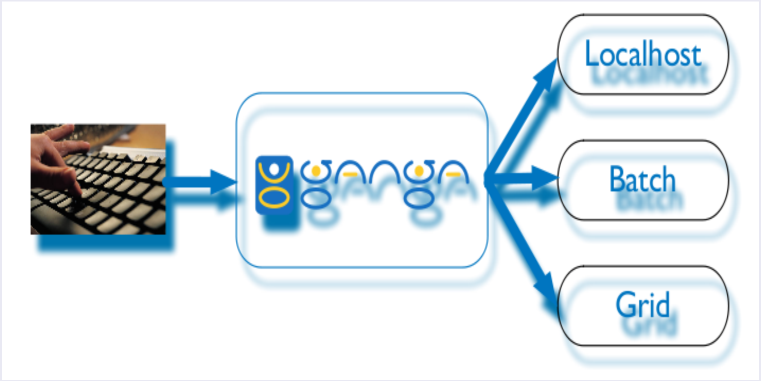
Cloud Services for Synchronisation and Sharing (CS3)
ETH Zurich

Introduction

"Ganga is an easy-to-use frontend for job definition and management, implemented in Python."

The Ganga Mantra

Configure once, run anywhere.



How can Ganga help?

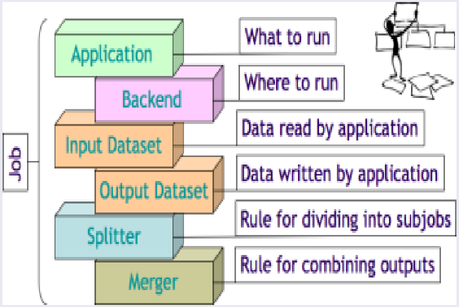
Job Life Cycle

Build $\rightarrow$ Configure $\rightarrow$ Split $\rightarrow$ Submit $\rightarrow$ Monitor $\rightarrow$ Merge

- Ganga easily allows for software checkout and building
- Ganga provides a user-friendly configuration environment
- Jobs can be split, upon submission, to form multiple 'subjobs'
- A built-in monitoring thread polls jobs for status
- Ganga provides a convienient framework in which to merge jobs once complete (or indeed other post-run activities)

What makes up a Ganga Job?

The Ganga 'Job' object



Applications and Backends

Most Important Objects

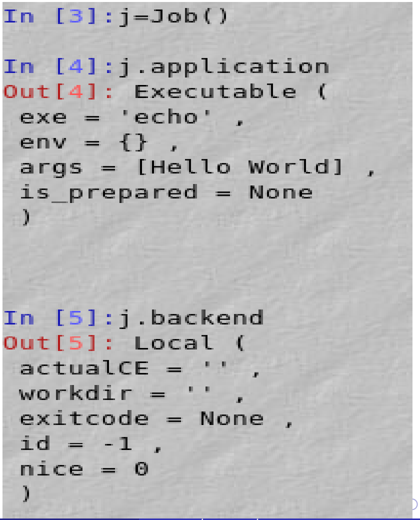
Applications

- Executable
- Root
- Several experiment specific

Backends

- Local
- Batch (LSF, PBS, SGE)
- Grid (DIRAC, Panda)

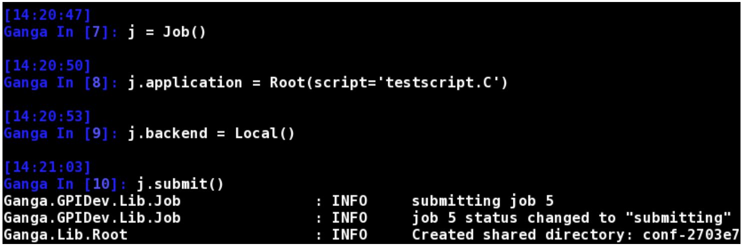
In action



```
In [3]:j=Job()

In [4]:j.application
Out[4]: Executable (
 exe = 'echo' ,
 env = {} ,
 args = [Hello World] ,
 is_prepared = None
 )


In [5]:j.backend
Out[5]: Local (
 actualCE = '' ,
 workdir = '' ,
 exitcode = None ,
 id = -1 ,
 nice = 0
 )
```

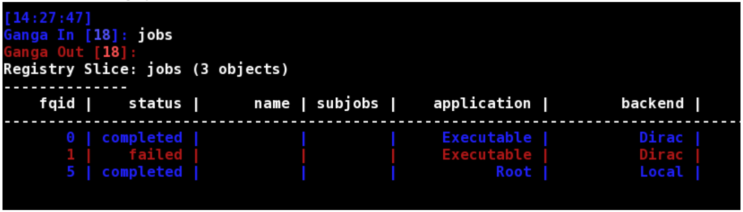
Using Ganga - Typical Usage

- Configuring/Submitting jobs



```
[14:20:47]
Ganga In [7]: j = Job()

[14:20:50]
Ganga In [8]: j.application = Root(script='testscript.C')

[14:20:53]
Ganga In [9]: j.backend = Local()

[14:21:03]
Ganga In [10]: j.submit()
Ganga.GPIDev.Lib.Job                    : INFO     submitting job 5
Ganga.GPIDev.Lib.Job                    : INFO     job 5 status changed to "submitting"
Ganga.Lib.Root                          : INFO     Created shared directory: conf-2703e7
```

- Monitoring jobs



```
[14:27:47]
Ganga In [18]: jobs
Ganga Out [18]:
Registry Slice: jobs (3 objects)
--------------
    fqid |    status |      name | subjobs |    application |        backend |
---------------------------------------------------------------------------------
       0 | completed |           |         |     Executable |          Dirac |
       1 |    failed |           |         |     Executable |          Dirac |
       5 | completed |           |         |           Root |          Local |
```

Handling Files

GangaFiles

- LocalFile
- MassStorageFile
- DiracFile
- Cloud-based files

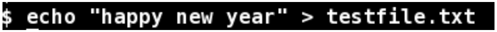
Out of the box Ganga supports file types for:

- Files stored on the local machine.
- Files stored on a mass storage system such as LCG EOS.
- Files stored on a Dirac storage element.

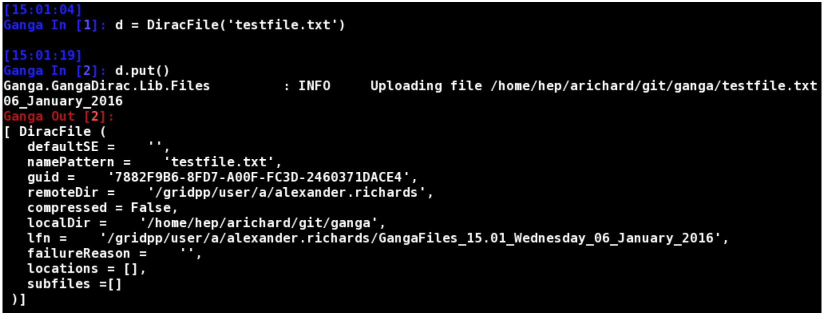
- These have a unified API
- The ability to do uploading/downloading on the worker node or fallback to doing it on the client
- We have now started expanding to cover cloud-based files

DiracFile Example

- GangaFile objects support the *PUT*/*GET* API.
- This makes it trivial to store/retrieve a file for example on a DIRAC SE.



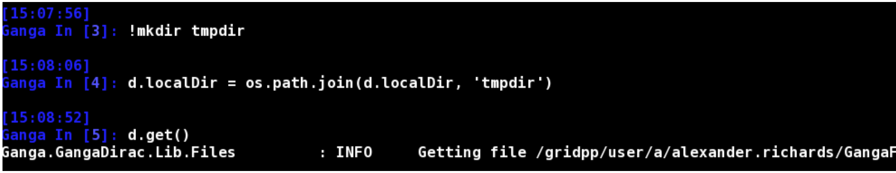
```
$ echo "happy new year" > testfile.txt
```



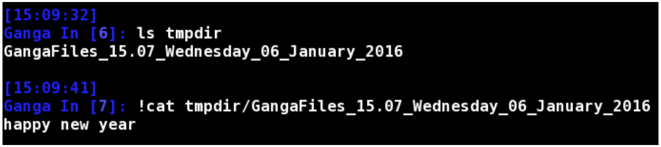
```
[15:01:04]
Ganga In [1]: d = DiracFile('testfile.txt')

[15:01:19]
Ganga In [2]: d.put()
Ganga.GangaDirac.Lib.Files         : INFO     Uploading file /home/hep/arichard/git/ganga/testfile.txt
06_January_2016
Ganga Out [2]:
[ DiracFile (
   defaultSE =    '',
   namePattern =    'testfile.txt',
   guid =    '7882F9B6-8FD7-A00F-FC3D-2460371DACE4',
   remoteDir =    '/gridpp/user/a/alexander.richards',
   compressed = False,
   localDir =    '/home/hep/arichard/git/ganga',
   lfn =    '/gridpp/user/a/alexander.richards/GangaFiles_15.01_Wednesday_06_January_2016',
   failureReason =    '',
   locations = [],
   subfiles =[]
 )]
```

DiracFile Example

- GangaFile objects support the *PUT*/*GET* API.
- This makes it trivial to store/retrieve a file for example on a DIRAC SE.



```
[15:07:56]
Ganga In [3]: !mkdir tmpdir

[15:08:06]
Ganga In [4]: d.localDir = os.path.join(d.localDir, 'tmpdir')

[15:08:52]
Ganga In [5]: d.get()
Ganga.GangaDirac.Lib.Files          : INFO     Getting file /gridpp/user/a/alexander.richards/GangaF
```



```
[15:09:32]
Ganga In [6]: ls tmpdir
GangaFiles_15.07_Wednesday_06_January_2016

[15:09:41]
Ganga In [7]: !cat tmpdir/GangaFiles_15.07_Wednesday_06_January_2016
happy new year
```

Case for using cloud storage

What one might want to achieve:

- Ease of use for both the storage of the data and the retrieval.
  - Including accessability across multiple platforms
- Possibility to share large files safely and easily.
- Integration with existing software solutions for cloud storage.
- No new authentication mechanisms.
- Secure (i.e. not transmitting and storing new tokens in plain text).

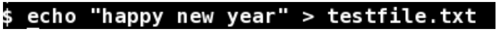
WebDAVFile

- WebDAV (Distributed Authoring and Versioning) allows remote content authoring
- Becoming available for many cloud-based storage solutions
  - Amazon - S3
  - CERN - CERNBox
  - Imperial - Box ?
  - Dropbox - although not directly (dropdav)
- Implemented in Ganga as WebDAVFile
- can use basic authentication or SSL key/cert
- Using python tinydav under the hood

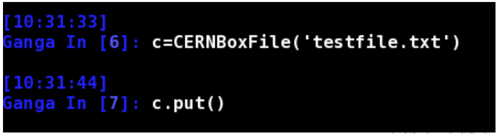
CERNBoxFile

The CERNBox server a useful testbed for HEP Ganga cloud file usage. (more tomorrow)

- Larger space to start with (100GB).
- Available for each CERN user.
- Potential for syncing with larger experiment specific storage (EOS).
- As it's a special case within HEP we have a CERNBoxFile which builds upon WebDAVFile using X.509 authentication
- As with all GangaFiles, can be used standalone or attached to the output/inputfiles attribute of a job

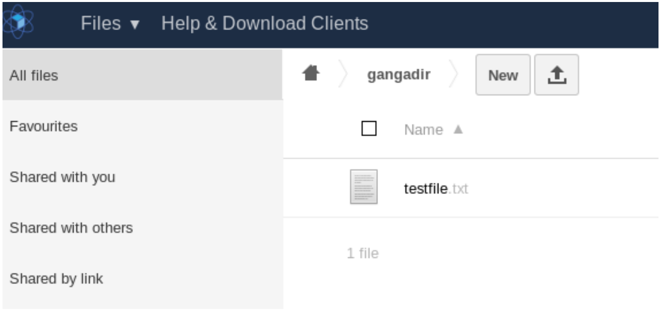


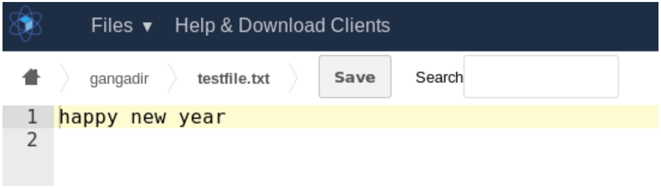
```
$ echo "happy new year" > testfile.txt
```



```
[10:31:33]
Ganga In [6]: c=CERNBoxFile('testfile.txt')

[10:31:44]
Ganga In [7]: c.put()
```

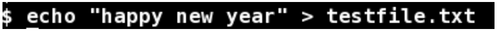
WebDAVFile/CERNBoxFile



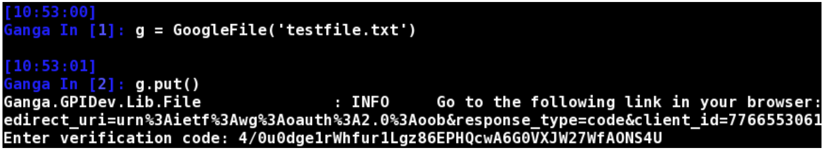


```
happy new year
```

GoogleFile

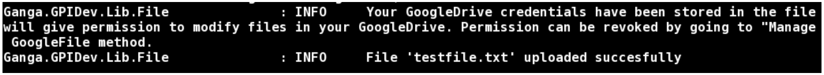


```
$ echo "happy new year" > testfile.txt
```



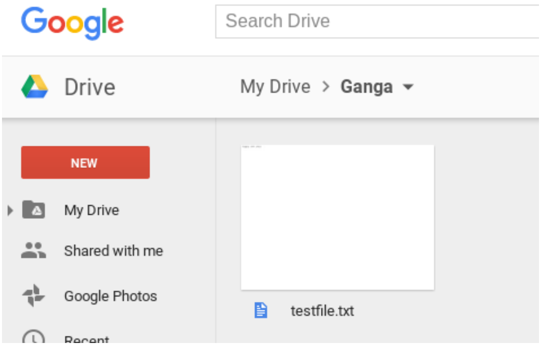
```
[10:53:00]
Ganga In [1]: g = GoogleFile('testfile.txt')

[10:53:01]
Ganga In [2]: g.put()
Ganga.GPIDev.Lib.File              : INFO     Go to the following link in your browser:
edirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&response_type=code&client_id=776655306l
Enter verification code: 4/0u0dge1rWhfur1Lgz86EPHQcwA6G0VXJW27WfAONS4U
```

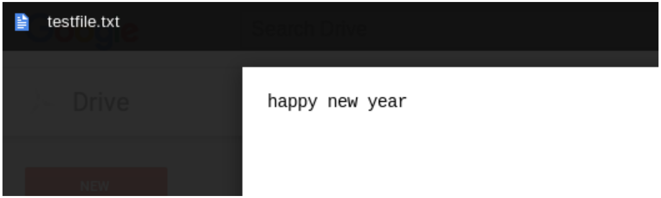
- Authentication set up via OAuth.
- Although Ganga makes it easy, requires separate authentication
- Only 15GB available.
  - Helpful to look at small files locally when job finished.
  - Not so useful for sharing large datafiles.
  - Storing large job output quickly clogs up quota



```
Ganga.GPIDev.Lib.File              : INFO     Your GoogleDrive credentials have been stored in the file
will give permission to modify files in your GoogleDrive. Permission can be revoked by going to "Manage
 GoogleFile method.
Ganga.GPIDev.Lib.File              : INFO     File 'testfile.txt' uploaded succesfully
```

GoogleFile





```
happy new year
```

Python Importing

- Plan to distribute Ganga via PyPI
- Easy installation

Example

```
$ pip install ganga
```

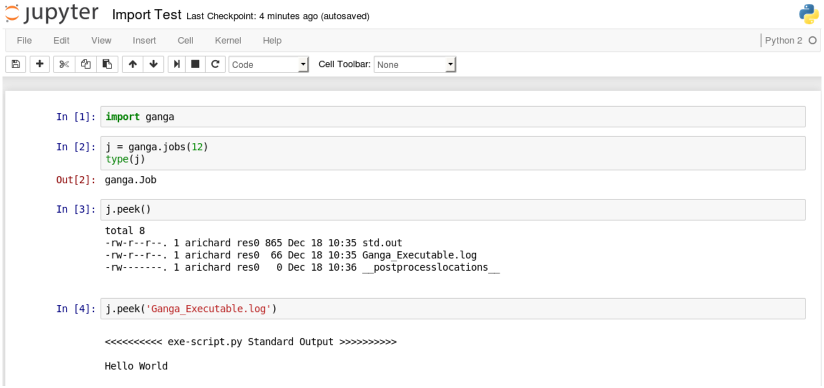
- A lot of work has been done recently to make the code base more modular.
- This will enable a user to directly import Ganga object into a python shell
- Great for scripting if you don't need all the monitoring machinery

Example

```
>>> import ganga
>>> j = ganga.Job()
>>> j.submit()
```

Jupyter

- CERN exploring providing access to analysis in ROOT through Jupyter
- With the python importing work above, using Jupyter is trivial

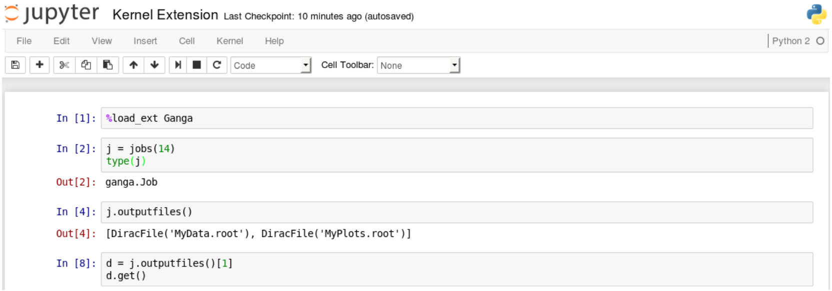

jupyter
Import Test Last Checkpoint: 4 minutes ago (autosaved)
File Edit View Insert Cell Kernel Help
Python 2
Code
Cell Toolbar: None

```
In [1]: import ganga

In [2]: j = ganga.jobs(12)
        type(j)

Out[2]: ganga.Job

In [3]: j.peek()

        total 8
        -rw-r--r--. 1 arichard res0 865 Dec 18 10:35 std.out
        -rw-r--r--. 1 arichard res0  66 Dec 18 10:35 Ganga_Executable.log
        -rw-------. 1 arichard res0   0 Dec 18 10:36 __postprocesslocations__

In [4]: j.peek('Ganga_Executable.log')

        <<<<<<<<<< exe-script.py Standard Output >>>>>>>>>>

        Hello World
```

- Owing to the generally asynchronous usage of Ganga we see the notebooks being most useful for documenting analysis.

Ganga & Jupyter

- Could also consider creating a Jupyter Kernel extension.
- This would allow us to pre-load into the user's namespace all the Ganga functions/classes that are imported in the interactive Ganga session.
- Essentially turning the notebook into a persisted Ganga session.

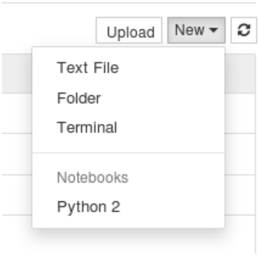


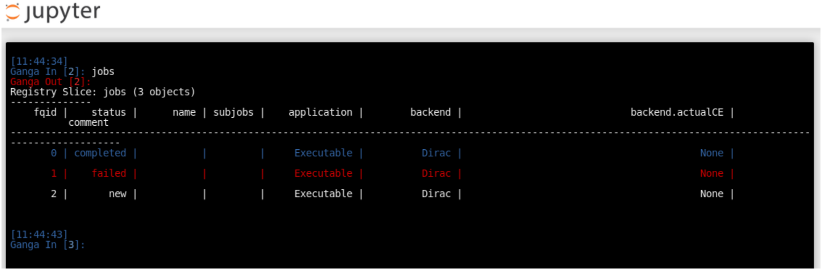
```
In [1]: %load_ext Ganga

In [2]: j = jobs(14)
        type(j)

Out[2]: ganga.Job

In [4]: j.outputfiles()

Out[4]: [DiracFile('MyData.root'), DiracFile('MyPlots.root')]

In [8]: d = j.outputfiles()[1]
        d.get()
```

- The loading of this kernel extension can be put into the users ipython_config.py

Ganga & Jupyter

- Could also consider a new Ganga terminal.
- This would open straight into the interactive Ganga session





```
[11:44:34]
Ganga In [2]: jobs
Ganga Out [2]:
Registry Slice: jobs (3 objects)
--------------
    fqid |    status |      name | subjobs |    application |        backend |                                 backend.actualCE |
        comment
-------------------------------------------------------------------------------------------------------------------------------------------
------------------
       0 | completed |           |         |     Executable |          Dirac |                                             None |
       1 |    failed |           |         |     Executable |          Dirac |                                             None |
       2 |       new |           |         |     Executable |          Dirac |                                             None |


[11:44:43]
Ganga In [3]:
```

Ganga & Jupyter

We envisage:

- The individual notebooks being used to perform and document analysis on completed jobs
- The Ganga 'terminal' being used for the creation/submission of jobs
- The Ganga 'terminal' running the monitoring of all jobs.

One could also consider:

- Moving the job creation/submission into a notebook, like one would a script.
- Owing to the asynchronous nature of the job life cycle, probably wouldn't want to combine such notebooks with analysis notebooks
- Using the terminal purely for the monitoring.

In summary...

- DIRAC + Ganga standard GridPP WMS/DMS job submission solution for smaller VO experiments in UK
- Starting to also support cloud-based files with unified API
- Ganga soon to be distributed via PyPI
  - pip install ganga
- Will be able to import and use Ganga objects straight into a python session.
- Looking at possibilities of integrating with Jupyter notebook