Introduction to Cloud Computing

Release Learning

Gregor von Laszewski

September 14, 2014

CHAPTER ONE

ABOUT

The purpose of this Web page is to provide developers and students with a very simple guide on how to get started to program in the cloud. The material is based on practical experience that we gained form interacting with undergraduate students starting from the freshman level to graduate students working on a PhD. Naturally, this means that some material may seem to you very simple and at other times we find that some material may be missing or may have changed over time.

It is a pleasure if you can improve this material and potentially contribute your own section to it. You can coordinate your contribution by contacting Gregor von Laszewski (laszewski@gmail.com).

1.1 Independent Study at IU

Gregor is also an Adjunct Associate Professor at Indiana University in the Computer Science Department and thus can supervise students to take independent studies. Independent studies provide an excellent opportunity to engage more intensely in projects related to cloud computing. Independent studies may not just include cloud computing activities but can also be done on other topics related to computer science. You can als propose your own topic and we can discuss if it is suitable. Almost all programming is done in Python, and Javascript. We accept a suitable small number of projects in Java if necessary as part of the topic. The topics are not listed by priority. One or more topic may be covered in the independent study. We will discuss this in a meeting. The amount of work for a 3 hour independent study is the same for a very demanding class in the departments. Expect 15-20 hours of work per week. Those with significant programming experience will have an easier time.

1.1.1 Cloud Topics

- Develop a cloud portal in django replicate significant functionality of our flask portal)
  - must use bootstrap
  - must use jinja2 in addition to djangos rendering engine
  - must interface with our mongodb (not needed for authentication)
  - must provide a user mashup between the user database from django and our own user database (containing contact information)
- Develop a user management system
- Improve the CLoudmesh Portal with Javascript
- Manage 10000 virtual machines
- Develop a PaaS launcher
- Develop an HPC interface

- Develop improvements to the cloud shell

1.1.2 Data Topics

- Develop a program that solves name ambiguity in bibliographic data by investigating a social network graph
- Develop a scalable distributed mongo db for our publication data, conduct performance comparisons for searches

1.2 Student Employment at IU

An independent study is also a precursor to gaining employment as a student with him at Indiana University with us. Students without significant programming experience will not be considered. Preference is given to those that have strong background in python and Javascript. If you do not have such knowledge, we expect that you gain it as part of your independent study. In some cases exceptions are made, this may include students that have programmed significantly in other projects.

Employment is mostly done on an hourly rate. During the time of employment we also recommend that students take an independent study with Gregor. IN some cases this is mandatory to be considered. Thus plan your independent studies carefully.

1.3 Contributing to the Manual

If you have a good chapter that you like to integrate into this manual, please contact Gregor von Laszewski (laszewski@gmail.com).

Suggestions include:

- Development Ecosystem
  - git
  - virtualenv
  - introduction to python
  - introduction to Javascript
  - flask
  - django (we have some material to start with)
  - explain how to create commands in cm with cmd3
- Cloud
  - get an account on FutureGrid (possibly just a link)
  - use nova client from the command shell
  - python example on how to manage vms with cloud mesh API & shell
  - IaaS Intro (pick your IaaS)
  - PaaS Intro (pick your Platform)

CHAPTER TWO

CONTACT

Gregor von Laszewski laszewski@gmail.com

2.1 Contributors

- Gregor von Laszewski (laszewski@gmail.com)
- Fugang Wang
- Hyungro Lee
- Mark Xiao
- Aravindha Varadharaju

CHAPTER

THREE

IPYTHON

3.1 IPython

IPython is a python command shell with notebook features that can be accessed through a browser. Throughout the material presented here we will be using IPython to present some of the demonstrations. This also allows you to try out the various excersises easily. We present here just a small set of features and recommend you to visit the IPython manaul for more information

3.1.1 Command Execution

To execute a shell command you can specify the ! at the beginning of a line

```
!echo "hallo"
```

```
hallo
```

3.1.2 Environment Variables

Environment variables can be accessed with $$

```
!echo "$$EDITOR"
```

```
emacs
```

Variables can be accessed by assigning values with = and by using them in { }

3.1.3 Variables

```
a="Hallo"
```

```
!echo "{a}"
```

```
Hallo
```

3.1.4 Surpressing output

Output can be surpressed while using the %%capture command. These commands are called magic functions in IPython. Many more magic functions are documented in the IPython manual

```
%%capture
!echo "You can not see me"
```

```
!echo "You can see me"
```

```
You can see me
```

CHAPTER

FOUR

PARALLEL SHELL

Traditionally system administrators and developers using parallel computing need tools to manage a significant number of machines. One of the requirements is to execute a command in parallel on many machines and gather its output. There are many tools that can achieve this task. We focus here on the introduction of the following tools:

1. pdsh - a parallel distributed shell
2. fabric - python framework to execute commands on remote computers
3. Cloudmesh Sequential and Parallel python functions for executing repeatedly commands with caching

4.1 Parallel Distributed Shell (pdsh)

The parallel distributed shell (pdsh) is a shell command line program that allows the execution of commands not just on one computer but on a list of computers.

An online version of the manual pages is located at

- http://linux.die.net/man/1/pdsh

An important feature is that the list of hosts can be specified in a convenient form that is also kwon as hostlists. This format allows you to define a list of hosts based on some abbreviation. For example the string:

```
host[0-3]
```

will create a host list containing the hosts:

```
host0, host1, host2, host3
```

Furthermore, substitutions for the user and the hostname to login to the remote machine while leveraging ssh config files make this tool real easy to use. One such example:

```
pdsh -R exec -w host[0-3] ssh -x -l %u %h hostname
```

executes the command hostname on all specified machines.

Todo

At this time we are not aware that pdsh is installed by default on india. check with the systems group and have them provide a documentation on how we activate it.

4.2 Fabric

Fabric is a Python command-line tool and library for assisting system administration tasks related to the execution of command via ssh. It includes the ability to execute commands on the local machine, but also on a remote machine. Due to the integration with Python function definitions, it has also somewhat the ability to write "target" like specifications that we may know from makefiles. However it is more sophisticated as we can use the full feature richness of python.

The web page of Fabric which includes several examples and tutorials is located at:

- http://www.fabfile.org

Similar to the previous command we like to start the command hostname on a number of machines. To install fabric you simply say:

```
pip install fabric
```

in your virtualenv. Next, we define a file called fabfile.py with the following contents:

```
from fabric.api import run

def hostname():
    run('hostname')
```

Next let us run this command on the local computer and test it out:

```
fab -H localhost hostname
```

This will execute the function defined with the name hostname and print it via the run command. To execute the command on multiple hosts, you can simply specify them as part of the -H argument:

```
fab -H host0,host1,host2,host3 hostname
```

4.3 Cloudmesh Parallel API

The previous commands are all developed with a single user in mind, i.e. a single user executes the command. However in the age of cloud computing what would happen if thousands of users were to execute the same task, or even when ten users execute the same task, but the task would take considerable compute time to calculate? The answer is obvious, we would waste valuable compute resources as we do not take into consideration that the same task may be run by multiple people. To overcome this challenge we have started a simple demonstration program in our cloudmesh repository to partially address this issue.

To do so we are at this time we are only focussing on the consecutive execution of a command in a particular time period. Instead of executing the command over and over, we will simply return the result from a result cache. The cache has a specific time to life in which no new results are created but the result is read from the cache. New requests are cached.

We recognize that the example we provide is not a complete solution to our problem, but a step in the right direction. I also has the advantage of being relatively simple and introducing you to a number of tools and concepts that will become important when dealing with parallelism in the cloud.

4.3.1 Requirements

1. A computer with python 2.7
2. Using a python virtualenv
3. Having downloaded the cloudmesh code with git clone as discussed elsewhere

4. Having installed the cloudmesh code and libraries as discussed elsewhere

4.3.2 Code

The code is located in the directory:

```
cloudmesh_examples/example_mongo
```

within the cloudmesh code you have cloned from github. The code contains two python functions called Sequential and Parallel, that allow users to run commands either sequentially or in parallel on a number of hosts. The hosts can be specified in a yaml file located in:

```
~/.cloudmesh/cloudmesh_hpc.yaml
```

An example would be:

```
meta:
  yaml_version: 3.0
  kind: hpc
cloudmesh:
    hpc:
        alamo:
            cm_host: alamo.futuregrid.org
            cm_type: hpc
            username: albert
        india:
            cm_host: india.futuregrid.org
            cm_type: hpc
            username: albert
        sierra:
            cm_host: sierra.futuregrid.org
            cm_type: hpc
            username: albert
        bigred:
            cm_host: bigred2.uits.iu.edu
            cm_type: hpc
            username: albert
```

This file is used to specify the username for each host and define the host names. In case you want to run commands on the hosts you can do this with the following python program.

The first command executes the task sequentially over the array given in the first parameter. The second one executes it in parallel. Instead of just presenting you with a bare bones program we present you with some additional features that are worth noting and may come in handy in future. This includes the availability of a named stopwatch and the ability to read configuration parameters easily from a yaml file. Sometimes it is also nice to have very visible debug messages that we create with a banner function. Results are often more readable when using the python pprint function instead of just the print function. This is especially true when we print data-structures such as arrays and dicts. Next we will present the program and explain a selected number of features by commenting them in the code. We assume you know by now elementary python.

```python
from cloudmesh_task.tasks import cm_ssh
from cloudmesh_task.parallel import Parallel, Sequential
from cloudmesh.util.stopwatch import StopWatch
from cloudmesh_common.util import banner
from pprint import pprint
from cloudmesh.config.cm_config import cm_config
from cloudmesh.config.ConfigDict import ConfigDict
import sys
```

```
# read the information from the yaml file into a dict called config
config = ConfigDict(filename="~/.cloudmesh/cloudmesh_hpc.yaml")["cloudmesh"]["hpc"]

# a function to extract from the config file the username from all
# hostnames in the array hosts
def  get_credentials(hosts):
    credential = {}
    for host in hosts:
        credential[host] = config[host]['username']
    return credential

# find all hostnames from the config file
hosts = config.keys()

# find all credentials (username, hostname) from the hosts in the
#  config file
credentials = get_credentials(hosts)


# create a stop watch
watch = StopWatch()

# execute is a python function. It is either Parallel or Sequential
# * modify
#    for execute in [Sequential]:
#    for execute in [Parallel]:

for execute in [Sequential, Parallel]:

    # get the name of the function
    name = execute.__name__

    # print the name of the function and start the timer
    banner(name)
    watch.start(name)

    # execute the function and return the result in a dict
    result = execute(credentials, cm_ssh, command="qstat")

    # stop the timer and print the result dict
    watch.stop(name)
    pprint(result)

    # only print the output from the command we executed
    banner("PRINT")
    for host in result:
        print result[host]["output"]

# print the timers
for timer in watch.keys():
    print timer, watch.get(timer), "s"
```

Bug: Before you start the command, you have to start a new window and say fab fab manage.mongo in the cloudmesh directory where your fabfiles are located. This will give something like:

```
$ fab manage.mongo
[localhost] local: make -f cloudmesh/management/Makefile mongo
```

```
mongod --noauth --dbpath . --port 27777
all output going to: /usr/local/var/log/mongodb/mongo.log
```

To run the command you will need to start the caching backend services. to do so we created a simple program cm-task.py that will be used to start and stop the services:

```
./cm-tasks.py menu

Queue Management
================

    1 - all start
    2 - all stop
    3 - rabbit start
    4 - celery start
    5 - rabbit stop
    6 - celery stop
    7 - mongo start
    q - quit

Select between 1 - 7:
```

Now select the number:

```
1 - all start
```

This will bring up the necessary services and look similar to:

```
 -------------- celery@host.local v3.1.13 (Cipater)
---- **** -----
--- * ***  * -- Darwin-13.3.0-x86_64-i386-64bit
-- * - **** ---
- ** ---------- [config]
- ** ---------- .> app:         cloudmesh_task:0x10365bcd0
- ** ---------- .> transport:   amqp://guest:**@localhost:5672//
- ** ---------- .> results:     amqp
- *** --- * --- .> concurrency: 10 (prefork)
-- ******* ----
--- ***** ----- [queues]
 -------------- .> celery           exchange=celery(direct) key=celery


[tasks]
  . cloudmesh_task.tasks.cm_ssh

[2014-08-19 15:46:24,060: INFO/MainProcess] Connected to amqp://guest:**@localhost:5672//
[2014-08-19 15:46:24,071: INFO/MainProcess] mingle: searching for neighbors
[2014-08-19 15:46:25,098: INFO/MainProcess] mingle: sync with 10 nodes
[2014-08-19 15:46:25,099: INFO/MainProcess] mingle: sync complete
[2014-08-19 15:46:25,109: WARNING/MainProcess] celery@host.local ready.
[2014-08-19 15:46:28,352: INFO/MainProcess] Events of group {task} enabled by remote.
```

After this you can start the program repeatedly with:

```
$ python prg.py
```

We are committing some of the output but at the end ist should look something like:

```
# ######################################################################
# PRINT
# ######################################################################
Tue Aug 19 15:48:29 EDT 2014
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
1589570.i136               sub18248.sub     aaaa                   0 Q delta
1589589.i136               sub15366.sub     aaaa                   0 Q delta
1589669.i136               sub12428.sub     aaaa                   0 Q delta
1795838.i136               twisterJob       bbbbbb                 0 Q batch
1872981.i136               sub9593.sub      aaaa                   0 Q delta
1904453.i136               sub2114.sub      aaaa                   0 Q delta
1904930.i136               dimer_in_sol_ph7 cccccccc        883:55:5 R batch
1904931.i136               dimer_in_sol_ph5 cccccccc        902:18:4 R batch
1904957.i136               suffix           dddddddd        360:36:1 R echo
1904961.i136               dimer_in_sol_ph7 cccccccc               0 H batch
1904963.i136               dimer_in_sol_ph5 cccccccc               0 H batch
1904993.i136               blast            eeee            15:08:00 R bravo
1904995.i136               blast            eeee            14:33:19 R bravo
1905016.i136               papi-inca        aaaa                   0 Q bravo
1905021.i136               vampir-inca      aaaa                   0 Q bravo
1905044.i136               papi-inca        aaaa                   0 Q bravo
1905057.i136               STDIN            ffffffff        00:10:17 R delta
1905062.i136               ...Script.i21500 gggggg          00:00:11 R batch

Sequential 12.12866169 s
Parallel    0.00446796417236 s
```

Please note that we have replaced the real usernames.

When you execute this command you will notice That the parallel execution time is much faster. In this case it was within the TTL and thus read the cache value from the cache. Executing the command again within the TTL will give you also for the sequential time a real short value:

```
Sequential 0.00726103782654 s
Parallel   0.000990867614746 s
```

It is not surprising the parallel result is even faster than the sequential one as the information gathering even from reading it out from the cache is done in parallel and no resource congestion exists at the scale we use for our example.

Let us now compare the true time between sequential and parallel execution. Simply modify the code in the * line and replace the loop accordingly:

```
Sequential 12.681866169 s
Parallel    6.51530909538 s
```

Thus we see two interesting performance improvements

First, the performance improvement for running the queries in parallel. Second, the improvement of retrieving the results from a cache. The later is important if we have many user on the system executing the same command.

The lesson we learn is that clouds must make use of execution parallelism as well as addressing reuse of repeated results.

4.4 Exercises

1. Is pdsh installed? Where
2. Return the hostname of the machines sierra, india and foxtrot via the fabric command

3. Execute the command qstat with fabric on sierra and india. If you have an account on bigred2, please try it also there
4. Run the cloudmesh Sequential and parallel program. Modify your cloudmesh file accordingly
5. Advanced: compare the performance of the cache backend between Mongodb and the use of RabbitMQ while switching RabbitMQ out with Redis in the Celery code.
6. Advanced: provide a documentation on how to run celery for this example on Redis.

CHAPTER

FIVE

CLOUDMESH

5.1 Cloudmesh Overview

We have provided some documentation about cloudmesh at http://cloudmesh.github.io/index.html.

Cloudmesh is an important component to deliver a software-defined system – encompassing virtualized and bare-metal infrastructure, networks, application, systems and platform software – with a unifying goal of providing Cloud Testbeds as a Service (CTaaS). Cloudmesh federates a number of resources from academia and industry. This includes existing FutureGrid infrastructure (4704 cores used by more than 355 projects), Amazon Web Services, Azure, HP Cloud, Karlsruhe using various technologies.

An high level architectural image is provided at

Figure 5.1: **Figure:** *High level architecture of Cloudmesh*

The three layers of the Cloudmesh architecture include a Cloudmesh Management Framework for monitoring and operations, user and project management, experiment planning and deployment of services needed by an experiment, provisioning and execution environments to be deployed on resources to (or interfaced with) enable experiment management, and resources.

Before continuing we recommend that you look at the following sections on the cloudmesh web page:

- http://cloudmesh.github.io/cloudmesh.html
- http://cloudmesh.github.io/cloud.html
- http://cloudmesh.github.io/rain.html
- http://cloudmesh.github.io/hpc.html

In this document we like to focus on the actual implementation of cloudmesh and how the various components are structured. For this purpose we have redrawn the Figure we pointed out earlier while focussing on a number of subcomponents that we will be looking more closely into.

Typo: there are two baremetal boxes in cloudmesh cor. one should be HPC

This diagram highlights some important information which we describe next

5.1.1 Cloudmesh Main Components

The main cloudmesh component include:

- Cloudmesh Install Management: which allows to easily install cloudmesh on a given operating system. This can also include a virtual machine.

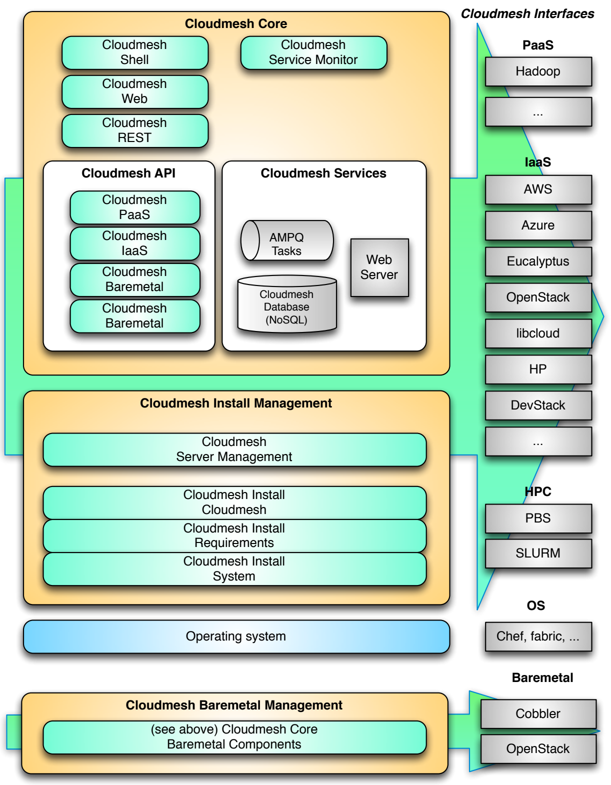


Figure 5.2: **Figure:** *High level architecture of Cloudmesh*

- Cloudmesh Baremetal Management (currently part of core): which allows the deployment of an os via bare metal through cloudmesh. The important differentiation to other systems is that users can be authorized to conduct bare metal provision based on service policy descriptions. The user may not be the administrator of the machine.
- Cloudmesh Core: which contains the major components to interface with external subsystems to conduct bare matel provisioning, interact with IaaS such as virtual machines, HPC queues, or the bare metal provisioning services.

5.1.2 Cloudmesh Install Management

- Conducted in three phases
- **Phase 1:** prepare the system. Te OS may have missing packages. the program will install all missing packages for cloudmesh. This step requires typically sudo permissions.
- **Phase 2:** Install the python requirements. As cloudmesh is mostly developed in python, this step installs all necessary python libraries.
- **Phase 3:** Install the cloudmesh programs. This Step installs the cloudmesh program in the python library. (We use virtual env for our development)

5.1.3 Cloudmesh Service Management

After all the software is installed, we can start up the various cloudmesh services. This includes

- **Cloudmesh Database**: A NOSQL database in which we record which virtual machines run on which IaaS. This allows us to have a federated view of the heterogeneous clouds.
- **Cloudmesh Web Service**: Provides a Graphical user interface to manage virtual machines and HPC tasks
- **Cloudmesh Task Service**: As cloudmesh is a multi user systems many tasks need to be handles in parallel. To achieve this we are using an AMPQ queue and coordinate the execution of managing multiple virtual machines for multiple users.

5.1.4 Cloudmesh Use Mode

Base on the rich service model in Cloudmesh we are able to start cloudmesh either in a

- standalone mode or in a
- hosted mode for multiple users.

For development purposes most users will want to run the cloudmesh services in standalone mode. This enables you to test out cloudmesh without interfering with other users. It also allows you to be completely responsible for your credentials without relaying them through a third party.

5.1.5 Cloudmesh Baremetal

Cloudmesh contains an interface to cobbler for its bare metal services. However the important feature is that it also contains a very small abstraction interface to bare metal provisioning. This will allow us to integrate with other bare metal provisioners and enable for example the use of OpenStack Ironic once it is deployed for example on FutureGrid.

5.1.6 Cloudmesh HPC

Cloudmesh provides an easy tou use API and GUI to HPC queues. It allows simple display of queues. As FutureGrid shares on some systems the queue manager it also seperates the queues appropriately and displays them accordingly. The API returns the job and queue information as python dicts.

5.1.7 Cloudmesh IaaS

Cloudmesh contains an abstration to interface with arbitrary IaaS farmeworks this includes

- Azure
- AWS
- OpenStack
- Eucalyptus
- clouds that can be accessed through libcloud

The important differentiation to other frameworks is that it is not just capable of interfacing with libcloud to a remote cloud but it is possible to provide interfaces while using the native protocol. This has greatly helped in debugging real clouds as for example some features are not properly exposed through libcloud or EC2 compatible mechanisms. It also protected us from several changes that took place during the various versions of OpenStack. Our OpenSTack library interfaces directly with the OpenStack REST services

5.1.8 Cloudmesh Web

While other clouds focus on their own infrastructure, Cloudmesh provides a user interface with federation capabilities to display and interact with heterogeneous clouds. In addition information between these clouds is not hidden behind a compatibility library such as libcloud or a cloud standard, but uses instead the natively available information. This allows developers to interact and inspect information on a different level than just being able to start and stop virtual machines. Interfaces to HPC queues are also available. The Web services interfaces with the Task and Database Services.

5.1.9 Cloudmesh Shell

For experiment management it os often not sufficient to just provide a GUI interface but to be able to script how virtual machines are coordinated. This can be done with our cloudmesh shell that similar to matlab has its own shell environment, but can also be simply be called as a command on a regular Linux terminal. The Cloudmesh Shell services interfaces with the Task and Database Services.

5.1.10 Cloudmesh State

As the Shell and the Web Service interact with the Database and the Task Services, the status after a refresh is synchronized between them. This means if I start a virtual machine with the command shell I can see it in the Web after a refresh and vice versa. Default values are shared and the interaction between Web and Shell is seamless.

The emphasize is here on managing multiple machines and the start of a VM can be done with a single click in the Web or with a single command without parameters in the shell. This is in contrast to other frameworks that do not make use of extensive default management for repetitive interactive experiments.

5.1.11 Tutorials

We have provided a number of tutorials through IPython notebooks that you can follow. A setup guide is available that documents the installation of cloudmesh through a single curl call. The tutorials will show you each of the three different interfaces including:

- the Python API
- the Web GUI via the Web browser
- the command shell

The examples focus on displaying information and managing virtual machines.

A list of all notebooks is available. The list will be expanded, and we would be happy if you contribute to them with your own suggestions. If you think we need to show a particular feature, please let us know. We wil try to add it.

5.1.12 Development and Transition to FutureSystems

- Due to the transition of FutureGrid to FutureSystems, we have limited our tutorial activities in regards to baremetal provisioning
- We are focussing our current efforts on the development of our PaaS launcher that interfaces with chef and ssh to deploy platforms on other resources.

5.2 Cloudmesh Setup

Cloudmesh can be setup to on your local environment in single user mode, or as a hosted service for multi-tenancy use. Based on your needs you may decide which way to set it up.

For development and class users we recommend the local setup.

5.2.1 Quick Start on your desktop

Warning: this tutorial is for the new FutureSystems infrastructure. However at this time we still use FutureGrid. Please replace all occurrences of FutureSystems with FutureGrid.

This quickstart is designed for Ubuntu 14.04 and OSX.

We recommend that you use virtualenv to provide an isolated environment for cloudmesh. We assume you create one called ENV and activate it:

```
virtualenv ~/ENV
$ source ~/ENV/bin/activate
```

First you need to download the code from github. We assume you have git installed:

```
$ git clone https://github.com/cloudmesh/cloudmesh.git
```

Next, you need to install a number of required packages with the following commands:

```
$ cd cloudmesh
$ sudo ./install system
$ ./install requirements
```

Note: on OSX you can omit the sudo.

To get access to IaaS cloud platforms, you need to create locally a new user that has access to various clouds. This can be done with:

```
$ ./install new
```

The next steps will deploy the cloudmesh code into the virtualenv library path:

```
$ ./install cloudmesh
```

Note: This step is optional but highly recommended for users.

In case you have accounts on the IU machines you can also obtain pre-configured cloud rc files from them. To test if you have an account and have set it up correctly, please login to the machine india:

```
$ ssh [username]@india.futuresystems.org
```

If this does not work, you may not have uploaded your public key to FutureSystems portal at

- https://portal.futuresystems.org/my/ssh-keys

Once this step is completed, you can create the configuration files as follows:

```
$ cm-iu user fetch
$ cm-iu user create
```

You may also need to specify a project id as follows:

```
$ cm project default [project id]
```

[project id] is your default project id e.g. fg82

Other resources may require you also to manage keys.

At this time we like you to edit some information about yourself in the cloudmesh.yaml file. Choose your favorite editor:

```
$ emacs ~/.cloudmesh/cloudmesh.yaml
```

Change the values TBD that you find here with values that describe you.

As you will need at one point to login into virtual machines you will need a key that cloudmesh can use do to so. We assume you have a public key generated in your .ssh directory in the file:

```
~/.ssh/id_rsa.pub
```

If you do not have such a key, you can generate it with:

```
ssh-keygen -t rsa -C $USER-key
```

We add this key to the cloudmesh database with:

```
cm "key add --keyname=gvonlasz-key ~/.ssh/id_rsa.pub"
```

The next steps will deploy the cloudmesh databases:

```
$ fab mongo.reset
```

To start the cloudmesh services use:

```
$ fab server.start
```

Now you can test the services by visiting the web interface at http://127.0.0.1:5000. We have a convenient shortcut for this by typing:

```
$ fab server.view
```

Alternatively you can use the cloudmesh shell by invoking the cm command via a terminal:

```
$ cm

======================================================
/ ___| | ___  _   _  __| |_ __ ___   ___  ___| |__
| |   | |/ _ \| | | |/ _` | '_ ` _ \ / _ \/ __| '_ \
| |___| | (_) | |_| | (_| | | | | | |  __/\__ \ | | |
\____|_|\___/ \__,_|\__,_|_| |_| |_|\___||___/_| |_|
======================================================
Cloudmesh Shell

cm> cloud
+--------------------------+----------+
| cloud                    | active   |
+==========================+==========+
| alamo                    |          |
+--------------------------+----------+
| aws                      |          |
+--------------------------+----------+
| azure                    |          |
+--------------------------+----------+
| dreamhost                |          |
+--------------------------+----------+
| hp                       |          |
+--------------------------+----------+
| hp_east                  |          |
+--------------------------+----------+
| india_eucalyptus         |          |
+--------------------------+----------+
| india_openstack_havana   |          |
+--------------------------+----------+
| sierra_eucalyptus        |          |
+--------------------------+----------+
| sierra                   |          |
+--------------------------+----------+

cm> cloud on india
...
cloud 'india' activated.

cm> flavor india --refresh
...
Refresh time: 0.190665006638
Store time: 0.0578060150146
+--------+------+--------------+---------+-------+--------+----------------------+
| CLOUD  |   id | name         |   vcpus |   ram |   disk | cm_refresh           |
|--------+------+--------------+---------+-------+--------+----------------------|
| india |    1 | m1.tiny      |       1 |   512 |      0 | 2014-08-26T01-15-20Z |
| india |    3 | m1.medium    |       2 |  4096 |     40 | 2014-08-26T01-15-20Z |
| india |    2 | m1.small     |       1 |  2048 |     20 | 2014-08-26T01-15-20Z |
| india |    4 | m1.large     |       4 |  8192 |     40 | 2014-08-26T01-15-20Z |
| india |    7 | m1.memmedium |       1 |  4096 |     20 | 2014-08-26T01-15-20Z |
| india |    6 | m1.memlarge  |       1 |  8192 |     20 | 2014-08-26T01-15-20Z |
+--------+------+--------------+---------+-------+--------+----------------------+
```

Commands without description

This script assumes that you have a key in:

```
~/.ssh/id_rsa.pub
```

Which will be used to log into the VMs and the machines. This key must be uploaded to the FutureSystems portal.

```
git clone https://github.com/cloudmesh/cloudmesh.git
virtualenv ~/ENV
source ~/ENV/bin/activate
cd cloudmesh
sudo ./install system
./install requirements
./install new
./install cloudmesh
cm-iu user fetch
cm-iu user create
fab mongo.reset
fab server.start
cm cloud list
cm cloud on india
cm flavor india --refresh
```

One line install with curl

This script can also be executed while getting it from our convenient instalation script repository. For ubuntu you can use:

```
$ curl -sSL https://cloudmesh.github.io/get/ubuntu/ | username=[your Futuresystems portal id] sh
```

It will install cloudmesh in the directory where you started it from and place it in the directory:

```
cloudmesh
```

It creates also a directory called ~/github/cloudmesh and then cds into this directory to conduct the installation from there. Furthermore, as you can see this script also creates a virtual env under the name ~/ENV

If you do not like these names or have a conflict with the names, please download the script and modify accordingly.

5.2.2 Creating the yaml file

You must have installed cloudmesh as discussed in ??? and run:

```
./install/new
```

This will create a ~/.cloudmesh directory with some basic yaml files that you will need to modify.

Adding FutureGrid Openstack clouds on sierra and india to the yaml file

For FutureGrid we have additionally provided a script that automatically creates some yaml files from the installation. In future FutureGrid will provide directly a yaml for cloudmesh so that this step is unnecessary. Before you can execute this command you maust make sure that you can log into india and sierra via ssh. Once you have verified this for example with:

```
ssh USERNAME@india.futuregrid.org hostname
ssh USERNAME@sierra.futuregrid.org hostname
```

Now create the yaml file while fetching some information from the remote machines:

```
./install rc fetch
```

First it will ask you which username you have on FutureGrid. The name may be different from your current local machine name. Please enter your name when you see:

```
Please enter your portal user id [default: flat]:
```

After this you can update the yaml files with the data fetched from the india and sierra with the command:

```
./install rc fill
```

The reason why we have separated the commands and not just created one command is to provide you with the ability to double check overwriting possibly an existing rc file.

Adding FutureGrid OpenStack Clouds on alamo and hotel to the yaml file

We do not recommend adding these machines as they use the FG portal and password. However if you do so, we have placeholders in the yaml file for these clouds. In case you can not find them, simply copy the one from india and make appropriate corrections.

Adding HP cloud to the yaml file

The cloud offered from HP is an Openstack cloud and contains the ability to conduct project and user based billing. As this cloud is Openstack it behaves much the same as the once defined on India and Sierra. There may be differences based on the version.

HP provides an interface to their cloud through horizon. The documentation for it can be found at:

- http://docs.hpcloud.com/hpcloudconsole

To use the cloud you have to first create an account with HP, which will charge you real money for using their cloud. Make sure you understand what costs will be charged before you request thousands of virtual machines. Naturally this is valid for any other commercial cloud also. The console for the HP cloud is available at:

- http://www.hpcloud.com/console

Which will bring you to their horizon interface:

- https://horizon.hpcloud.com

You can add your username and password into the cloudmesh.yaml in the .cloudmesh directory. It is that simple. However, presently the data is stored in cleartext which we will change in future. Thus if your would like to run cloudmesh we currently recommend running it on your own local machine. Make sure that the access to the yaml file is properly secured.

Adding AWS to the yaml file

Amazon EC2 Cloud requires Secret Access Keys to use Amazon Web Services (AWS). To configure Amazon EC2 on Cloudmesh, you need to provide the Secret Access Keys of your account. Amazon allows only to download the credentials via their web page, you need to go to the Security Credentials page to get the credentials. You may use your existing AWS account or create a new AWS account. The Access Key is a pair of Access Key ID and Secret Access Key and these values should be replaced with *EC2_ACCESS_KEY* and *EC2_SECRET_KEY* fields in the yaml file. Cloudmesh identifies *cm_type: aws* as Amazon Web Services in the yaml file, you update the *aws* section with your security credentials. Note that Amazon offers commercial services, the access key identification and the secret key should be kept in a safe place to avoid any unexpected usage from someone who you didn't authorize.

Adding Azure to the yaml file

Microsoft Windows Azure offers security credentials per a valid subscription on a user account. Based on the subscription id, chargeable usage is going to be applied to your bill. To authenticate requests to Azure, you need to configure your credentials for Cloudmesh. The following step-by-step tutorial explains the configuration of Azure credentials on Cloudmesh.

To connect Azure Virtual Machines to Cloudmesh, you need to provide Azure credentials to authenticate requrests in the yaml file. You can find the credentials in the

- Azure Management Portal

which is a web interface to manage your account and Azure Virtual Machines. Also, you can find credentials by downloading the subscription file (.publishsettings) here:

- http://go.microsoft.com/fwlink/?LinkId=254432.

Once you download the file, you may need to import your subscription Id and valid X.509 certificate from the file with the help of the Azure cross-platform command line interface. More information about the Azure CLI can be found in the Manual/article about the

- Azure Cross-Platform Command-Line Interface.

The Azure credentials require that the X.509 certificate is placed in the *.cloudmesh* directory. The *subscriptionid* field should be filled with your Azure subscription id. The valid X.509 certificate file (.pem) must also be stored in the *.cloudmesh* directory. We store it under the name:

```
$HOME/.cloudmesh/azure_managementCertificate.pem
```

Cloudmesh yaml file has an example non valid entry that you can change with your settings. it can be easily identified whle looking for the keyword azure in the *cloudmesh.yaml* file As Azure is a commercial service it is important that you properly secure the .cloudmesh directory and its yaml files.

Todo

describe briefly how to do that, use chmod go-rw ... chmod u+r ...

Azure Quickstart

Azure account If you do not have an Azure account you can obtain one from Microsoft. Microsoft provides a free-trial for new account applicants. The Windows Azure site is located at

- https://manage.windowsazure.com

Download credentials Form ther you can download the:

```
.publishsettings
```

Install Azure CLI Next you will need to install the Azure CLI. This is documented at

- http://azure.microsoft.com/en-us/documentation/articles/xplat-cli/

Here you find install instructions for Linux but also a link to an OSX installer.

Once the client is installed you can download the credentials

Import Credentials via Azure CLI

```
$ azure account download
$ azure account import <.publishsettings file path>
```

Download Subscription File (.publishsettings)

- http://go.microsoft.com/fwlink/?LinkId=254432

Place X.509 certificate on Cloudmesh

```
 $ cp -p ~/.azure/managementCertificate.pem ~/.cloudmesh/azure_managementCertificate.pem

Only the owner with read and write permission e.g. -rw-------
```

Todo

explicitly describe how to do that , do not use the 755 or other numbers use chmod go- ....

Replace Subscription ID

```
$ azure service cert list
```

provides your subscription id that just imported from the .publishsettings file.

Now, you are ready to use Azure Virtual Machines on Cloudmesh.

Test Azure Virtual Machine TBD

Adding devstack to the yaml file (TBD)

DevStack offers an easy method to try out Openstack on your machine or in a virtual machine (VM). DevStack provides a setup guide and configuration here: Configuration.

Adding dreamhost to the yaml file

Dreamhost provides an Openstack cloud that can be accessed through the dreamhost panel at:

- https://panel.dreamhost.com/index.cgi

The Horizon interface is located at

- https://dashboard.dreamcompute.com

If you are a customer of dreamhost, use your username and password that was send to you.

To use cloudmesh, please add this username and password in the placeholder for dreamhost.

5.2.3 Quickstart for an Openstack VM

Note: This setup is primarily used for testing, but it can also be useful for classes using OpenStack, when the call participants have access to an OpenStack cloud.

Setting up Cloudmesh on a VM is an especially convenient way during development and testing. To do so, you can follow the steps to run cloudmesh in a VM running Ubuntu 14.04 on FutureSystems *India* OpenStack. The instructions have been tested on a small instance and the whole process could take about half an hour before you can access the running server.

Requirements

We assume that you have set up an account on FutureSystems and are able to log into the machine with the name india.

If you use a different cloud, you can adapt the instructions accordingly.

Starting the VM

First, you have to start a VM on the cloud and assign it a public IP.

This can be done in multiple ways, using the command line, vagrant, or the horizon GUI. Let us assume you have set it up via the horizon GUI or the novaclient command line. This is described in the following document:

More information about horizon on FutureSystems is available at our manual page.

We summarize the following steps, however like to point out that the best source for this is our previously pointed out document.:

```
ssh <portalname>@india.futuresystems.org

india$ source ~/.futuregrid/openstack_havana/novarc
india$ nova keypair-add --pub-key ~/.ssh/id_rsa.pub $USER-india-key

india$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
india$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
india$ nova secgroup-add-rule default tcp 8888 8888 0.0.0.0/0
india$ nova secgroup-add-rule default tcp 5000 5000 0.0.0.0/0
india$ nova secgroup-list-rules default

india$ nova boot --flavor m1.small --image "futuregrid/ubuntu-14.04" --key_name $USER-india-key $USEF


india$ nova floating-ip-create

india$ export MYIP=`nova floating-ip-list |fgrep "None" | cut -d '|' -f2`
india$ nova add-floating-ip $USER-001 $MYIP
india$ nova show $USER-001
```

You should see a table similare like this:

```
+--------------------------------------+---------------------------------------------------------------
| Property                             | Value
+--------------------------------------+---------------------------------------------------------------
| status                               | ACTIVE
| updated                              | 2014-09-12T19:27:30Z
| OS-EXT-STS:task_state                | None
| private network                      | 168.39.1.34, 192.165.159.40
| key_name                             | USER-key
| image                                | futuregrid/ubuntu-14.04 (02cf1545-dd83-493a-986e-583d53ee372
| hostId                               | hsakjfhsdlkjfhsdlkjhflskjdhflkjsdhflkjshfpoeuiyrewuohfkljsdk
| OS-EXT-STS:vm_state                  | active
| OS-SRV-USG:launched_at               | 2014-09-12T19:27:30.000000
| flavor                               | m1.small (2)
| id                                   | 7e458cbd-d37d-443a-aa76-adc7fcad52ea
| security_groups                      | [{u'name': u'default'}]
| OS-SRV-USG:terminated_at             | None
| user_id                              | sjhkjsahflkjashfkljshfkdjsahfkjh
| name                                 | USER-001
| created                              | 2014-09-12T19:27:23Z
| tenant_id                            | abcd01234hfslkjhfdskjfhkjdshfkjs
| OS-DCF:diskConfig                    | MANUAL
| metadata                             | {}
| os-extended-volumes:volumes_attached | []
| accessIPv4                           |
| accessIPv6                           |
| progress                             | 0
| OS-EXT-STS:power_state               | 1
| OS-EXT-AZ:availability_zone          | nova
| config_drive                         |
+--------------------------------------+--------------------------------------------------------------
```

Looking at the status you will see if the VM is in ACTIVE state. Once this is the case you can login to it with:

```
india$ ssh -i ~/.ssh/id_rsa.pub ubuntu@$MYIP
```

Preparation of the VM

Next you have to update the operating system while logging into the VM:

```
sudo apt-get update
sudo apt-get install git
```

To obtain cloudmesh you need to clone it from git hub and change to the cloudmesh directory:

```
cd ~
git clone https://github.com/cloudmesh/cloudmesh.git
cd cloudmesh
```

The first thing you have to do is to fix some ip addresses on india with the command:

```
./bin/fix-india-routing.sh
```

Installation

To start the installation of cloudmesh we first need to install a number of packages with:

```
./install system
```

We also recommend that you run virtualenv in python which you can enable with:

```
cd ~
virtualenv  --no-site-packages ~/ENV
source ~/ENV/bin/activate
```

Now let us install cloudmesh into this virtualenv:

```
cd cloudmesh
./install requirements
./install new
```

The last command will create a number of yaml files in a folder:

```
~.cloudmesh
```

Next, install the cloudmesh server anad API with:

```
./install cloudmesh
```

Now we need to populate the cloudmesh.yaml file with your actual information. You can edit the file ‘‘~/.cloudmesh/cloudmesh.yaml‘ either with emacs or vi:

```
emacs ~/.cloudmesh/cloudmesh.yaml
```

or:

```
vi ~/.cloudmesh/cloudmesh.yaml
```

In this file, update your user profile, name, project data. Alternatively, if you already have yaml files on for example india.FutureSystems.org you can copy your local working yaml files from that machine to the virtual machine.

Yet another alternative is to use the functionality provided by cloudmesh:

```
cm-iu user fetch
cm-iu user create
```

This will fetch your cloud credentials from FutureSystems and populate them into the yaml config file. BEFORE you can do this, make sure you can log into the FutureSystems resources, e.g. india. You will need a private key present in the VM that the matching public ssh key had been registered to the FutureSystems. Additionally you may need to excetue the following beforehand to add your password protected key into the session:

```
eval `ssh-agent -s`
ssh-add
```

To run cloudmesh you will need to start a number of services that you can do with:

```
fab mongo.boot
```

In some cases you may see connection problems in the later step. In that case please execute this command one again so the tables and security settings are done properly.

Once the mongo is initiated properly it's time to update the user data with:

```
fab user.mongo
```

Before you start the server, you need to execute this so the server would be accessible from outside:

```
fab india.configure
```

And then start the server:

```
fab server.start
```

Then the cloudmesh service should be available via:

```
http://PUBLIC_IP_OF_THE_VM:5000
```

NOTE:

1. As you might be copying your yaml files into the cloud please secure the VM (following good security practice, including but not limited to proper ssh settings disallowing password authentication, securing the location of your private key as well as setting a passphrase, etc.). As this method targets the scenario for rapid dev and testing, it will be a good idea that shutting the vm down after using.
2. As the server is not secured by HTTPS, remember not to use your favorite password when you are asked to set a password for portal login.
3. This method is only intended for development and testing, and not recommended for real production use.

More information about more sophisticated install instructions are provided at

- http://cloudmesh.futuregrid.org/cloudmesh/developer.html#install-the-requirements

5.2.4 Setup Cloudmesh in an VirtualBox VM with Vagrant for Testing

This tutorial provides as how to deploy Cloudmesh with Vagrant and VirtualBox. Official Ubuntu 14.04 Server LTS 64 bit and 32 bit are supported as base images of Vagrant.

Download cloudmesh

```
git clone https://github.com/cloudmesh/cloudmesh.git
cd cloudmesh
```

Install Vagrant and VirtualBox

This instructions are tested on Ubuntu 14.04.

```
sudo apt-get install vagrant
sudo add-apt-repository multiverse
sudo apt-get update
sudo apt-get install virtualbox
```

Install Veewee (Optional)

There are requirements prior installing Veewee.

```
gem install veewee
```

- On OS X Mavericks:

  ```
  $ ARCHFLAGS=-Wno-error=unused-command-line-argument-hard-error-in-future gem install veewee
  ```

Vagrant up

```
cd ~/cloudmesh/vagrant/example1
./run.sh
```

FutureGrid Portal ID

Provide your portal ID:

```
==================================
Futuregrid portal id? (def: )

==================================
```

Base Image

Select one of the base images:

```
==================================
Select base image to launch
==================================
1) Ubuntu Server 14.04 64bit
2) Ubuntu Server 14.04 32bit
Please choose an option:

Ubuntu Server 14.04 xxbit selected
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Checking if box 'ubuntu/trustyxx' is up to date...
```

Vagrant will be loaded.

Vagrant ssh

Cloudmesh installed on a root account. You need to switch user account to a root once you ssh to the VM.

```
vagrant ssh
sudo su -
```

Virtualenv and cm

Cloudmesh installed on Virtualenv. You need to enable the environment. `cm` Cloudmesh interactive shell is ready to use.

```
source ~/ENV/bin/activate
cd cloudmesh
cm
```

5.3 Cloudmesh cm

5.3.1 Cloudmesh `cm` Command

iPython Execution of shell commands

In this section we use one of the build in features of iPython. IPython provides various mechanisms to call programs within its shell. One of the ways to do so is to use the ! character at the beginning of a line to execute the command in the shell.

However, there are more convenient ways to eliminate the ! sign at the beginning of a line. One way is to use the alias command, another is to use the %rehasx command.

IPython Alias

With the alias command we simply define a new command with the name cm that we can call directly from IPython. Here we make sure that the parameters ar between “” so that they are properly set. Just execute the following lines.

```
alias cm cm %s
```

Now let us test the command and lets print the version of cloudmesh cm

```
cm version
```

```
1.0.4
```

Python %rehashx

In addition to the direct specification IPython has also a rehashx function, that loads the commands found in the $PATH variable so you can execute the without !.

```
%rehashx
```

```
cm version
```

```
1.0.4
```

We are using now one of the methods to call the cm commands in the following sections.

`cm` Command

The `cm` command has a number of options that are useful to pass a script or a command directly into cm. Please however not that in some cases the command must be quoted to avoid confusion between flags used for cm and flags used for its subcommands. Let us invoke the -h flag to see which options cm has.

```
cm -h
```

```
cm.

    Usage:
      cm [-q] help
      cm [-v] [-b] [--file=SCRIPT] [-i] [COMMAND ...]

    Arguments:
      COMMAND                  A command to be executed

    Options:
      --file=SCRIPT  -f  SCRIPT  Executes the script
      -i                 After start keep the shell interactive,
                         otherwise quit [default: False]
      -b                 surpress the printing of the banner [default: False]
```

Help

Now let us execute the help command to see what other functions are supported. As cm is based on cmd3 that you can find in pypi it inherits a number of commands from cmd3. However, more importantly it also obtains a number of commands from cm itself. To more easily distinguish the categories of the cloud related commands we introduced two of them called GUI commands and cloud commands.

```
cm help
```

```
Documented commands (type help <topic>):
========================================
EOF       edit      help       label   plugins  register        user
banner    exec      image      list    project  script          var
clear     exp       info       man     py       security_group  verbose
cloud     flavor    init       metric  q        storm           version
defaults  graphviz  inventory  open    quit     timer           vm
dot2      group     keys       pause   rain     use             web

Gui Commands
============
web

Cloud Commands
==============
cloud     group  inventory  rain            storm  keys
defaults  image  list       register        user   project
flavor    init   metric     security_group  vm
```

Starting the Web Browser

To start the browser, simply type the command

Listing Clouds

```
cm cloud list
```

```
+----------+----------+
| cloud    | active   |
+==========+==========+
| alamo    |          |
+----------+----------+
| aws      |          |
+----------+----------+
| azure    |          |
+----------+----------+
| hp       |          |
+----------+----------+
| hp_east  |          |
+----------+----------+
| india    | True     |
+----------+----------+
| sierra   | True     |
+----------+----------+
```

Let us inspect the parameters. To limit the output we just display the first 10 lines of the help/man page. We see the –column option in the list command.

```
cm help cloud | head -n 10
```

```
::

    Usage:
        cloud [list] [--column=COLUMN]
        cloud info [CLOUD|--all]
        cloud alias NAME [CLOUD]
        cloud select [CLOUD]
        cloud on [CLOUD]
        cloud off [CLOUD]
```

...

For more information, read the help page. It essentially allows us to display some more useful information beyond to just document the active clouds. Let us also display the label. This is done with the following command.

```
cm "cloud list --column=active,label"
```

```
+----------+----------+------------+
| cloud    | active   | label      |
+==========+==========+============+
| alamo    |          | alamo      |
+----------+----------+------------+
| aws      |          | aws        |
+----------+----------+------------+
| azure    |          | waz        |
+----------+----------+------------+
| hp       |          | hpos       |
+----------+----------+------------+
| hp_east  |          | hpeos      |
+----------+----------+------------+
| india    | True     | ios_havana |
+----------+----------+------------+
| sierra   | True     | sos        |
+----------+----------+------------+
```

Let us now demonstrate a common error by not using proper quoting. This occurs when you use option flags with the command. Here our current parser is unable to distinguish between the options passed to cm and the options as used in the cm command. You see the usage message that we do not have a –column in the cm command. To avoid thie use the " " as previously shown.

```
cm cloud list --column=active,label

Usage:
      cm [-q] help
      cm [-v] [-b] [--file=SCRIPT] [-i] [COMMAND ...]
```

5.3.2 `project` Command in Cloudmesh `cm`

The `project` command provides a list of default, active, or completed project on a given user. You can update the project information with this command.

Note that all your `project` command executions update your yaml file (cloudmesh.yaml) and mongo db (user, defaults) both.

IPython Alias

With the alias command we simply define a new command with the name cm that we can call directly from IPython. Here we make sure that the parameters ar between "" so that they are properly set. Just execute the following lines.

```
alias cm cm %s
```

We are using now one of the methods to call the cm commands in the following sections.

`project info` Command

The `project` command has a number of options that are useful to manage a list of projects. The `info` option provides a current information on your account.

```
cm project info

Project Information
-------------------

   default: fg20
  projects: fg20, fg82
 completed: fg130
```

Set a default project

`project default [project name]` allows you to update a default project to your account.

```
cm project default fg82

CM ... site/cloudmesh-1.0-py2.7.egg/cloudmesh_cmd3/plugins/cm_shell_project.pyc:58:   INFO - sets the
CM ... site/cloudmesh-1.0-py2.7.egg/cloudmesh_cmd3/plugins/cm_shell_project.pyc:70:   INFO - fg82 pro
fg82 project is a default project now

cm project

Project Information
-------------------

   default: fg82
  projects: fg20, fg82
 completed: fg130
```

You can see your default project has been changed from fg20 to fg82. Note that `cm project` is same command with `cm project info`. `info` option can be suppressed.

Set an active project

You can add a project to your account as one of the active projects. You can have multiple projects in your active project list.

```
cm project active fg415

CM ... site/cloudmesh-1.0-py2.7.egg/cloudmesh_cmd3/plugins/cm_shell_project.pyc:75:   INFO - Sets the
CM ... site/cloudmesh-1.0-py2.7.egg/cloudmesh_cmd3/plugins/cm_shell_project.pyc:83:   INFO - fg415 pr
fg415 project is an active project(s) now

cm project info

Project Information
-------------------

   default: fg82
  projects: fg20, fg82, fg415
 completed: fg130
```

You can see the new project `fg415` added to the active projects.

Change the status of project to `completed`

If your project is completed and no longer needed, you may want to change the status from active to completed. `completed` option allows to change the status of the selected project.

```
cm project completed fg415

CM ... site/cloudmesh-1.0-py2.7.egg/cloudmesh_cmd3/plugins/cm_shell_project.pyc:115:   INFO - Sets a
CM ... site/cloudmesh-1.0-py2.7.egg/cloudmesh_cmd3/plugins/cm_shell_project.pyc:126:   INFO - fg415 p
fg415 project is in a completed project(s)

cm project

Project Information
-------------------
```

```
   default: fg82
  projects: fg20, fg82
 completed: fg130, fg415
```

Help message

```
cm "project -h"

Usage:
       project
       project info [--json]
       project default NAME
       project active NAME
       project delete NAME
       project completed NAME

Manages the project

Arguments:

  NAME           The project id

Options:

   -v       verbose mode

Usage:
       project
       project info [--json]
       project default NAME
       project active NAME
       project delete NAME
       project completed NAME

Manages the project

Arguments:

  NAME           The project id

Options:

   -v       verbose mode
```

5.3.3 Install IPython Server

Cloudmesh contains a convenient mechanism to set up an IPython web server. The server will run on port 8888 of your vm. It will install in your environment a notebook directory in ~/notebook. Here you can place your ipython notebooks. initially this directory will be empty.

You create the notebook server with:

```
cm notebook create
```

You start the notebook server with

```
cm notebook start
```

You can terminate the notebook server with:

```
cm notebook kill
```

When terminating, make sure you saved your notebooks. You can restart the server in case you need to run it again.

5.4 Cloudmesh API

5.4.1 Cloudmesh API Initialization

To access the many useful functions available in cloudmesh, we need to import the basic cloudmesh module.

```
import cloudmesh
```

Version

Cloudmesh has a version number that can be retrived with the version function

```
print cloudmesh.version()
```

```
1.0
```

5.4.2 Defining Names

Often we need to define some unique names to distinguish virtual machines or other objects that we use as part of our programming. Besisdes uuid that is provided byt pythin cloudmesh ahs additional functions that simplify name generation.

UUIDs

UUIDs are provided as part of the pythin standard libraries. There are a number of different initializations and which to uuse depends on your needs.

```
import uuid
```

Machine dependent uuid

One of them is to create UUIDs dependent on your machine name with uuid1.

```
uuid.uuid1()
```

```
UUID('f040aaf5-3242-11e4-bd35-600308a5f9d2')
```

Random Uuid

uuid4 creates random uuids.

```
uuid.uuid4()
```

```
UUID('ac332de1-faef-4054-aae8-6f32eb15f982')
```

As in some cases we want to generate names that do not include special characters such as - or . we avoid using the uuid function for now and use the function get_unique_name instaed.

```
uuid.UUID(bytes=uuid.uuid4().bytes)
```

```
UUID('e529ec9c-e40d-4e3b-b4db-4b46802bd4f6')
```

```
uuid.uuid4().int
```

```
64005057163216856052777834181321712834L
```

Cloudmesh get_unique_name

Sometimes it is beneficial to create uuids without the - in it. For this we have a convenience function in cloudmesh.

```
from cloudmesh_common.util import get_unique_name
```

```
print get_unique_name()
```

```
f04729ee324211e49889600308a5f9d2
```

As you can see it is just like the uuid function (currently uuid1 with the - removed.

In addition one can place a prefix into the uuid to make furher distinctions. However this is rarely needed.

```
get_unique_name("gregor")
```

```
'gregorf0487de3324211e4b177600308a5f9d2'
```

Generating VM names

To create a name for a virtual machine we often recommend to use a prefix with a number for an identifier of the virtual machine. This will come in handy when we need to start multile virtual machines and distinguish them with a different name. CLoudmesh uses internally by using your username from the FutureGrid portal as defined in the cloudmesh.yaml file.

```
import cloudmesh
```

```
print cloudmesh.vm_name("gregor", 1)
```

```
gregor-00001
```

5.4.3 Load Command

CLoudmesh comes with a number f easy to use configuration files. All of them are yaml files. These configuration files configure certain aspects. They are important when cloudmesh starts up or is run in user more. In server mode the information is typically retrieved from the cloudmesh database.

We focus here on some of the most important configuration files. After deployment they can be found in the `~/.cloudmesh` directory

API for cloudmesh.yaml

Information about the user and which clouds he has access to are stored in the `cloudmesh.yaml` file.

```
import cloudmesh

user = cloudmesh.load()

user.cloudnames()

['alamo', 'hp_east', 'sierra', 'aws', 'hp', 'india', 'azure']

user.firstname

'Gregor'

user.lastname

'von Laszewski'
```

To obtain the futuregrid username use the command `user.username()`

API for cloudmesh_server.yaml

To configure the server and the databases the server yaml file is used. via the load command it will be loaded into a dict. A special get function can get sub dictionaries.

```
config = cloudmesh.load("server")

print config.keys()

['kind', 'cloudmesh', 'meta']

config.get('meta').keys()

['yaml_version', 'kind', 'prefix', 'location', 'filename']

config.get('meta.filename')

'/Users/flat/.cloudmesh/cloudmesh_server.yaml'

print config.get('cloudmesh').keys()

['server']
```

API for cloudmesh_launcher.yaml

We are currently working on integrating PaaS launchers into cloudmesh that easily deploy software based on configuration parameters specified in the launchers. The specification of the launchers are stored under `cloudmesh.launcher.recipies`. We provide the information for such a launcher as an example.

```
config = cloudmesh.load("launcher")

print config.keys()
```

```
['kind', 'cloudmesh', 'meta']
```

```
config.get('cloudmesh').keys()
```

```
['launcher']
```

```
config.get('cloudmesh.launcher').keys()
```

```
['recipies']
```

```
config.get('cloudmesh.launcher.recipies').keys()
```

```
['mooc', 'nagios', 'slurm', 'ganglia', 'hadoop']
```

```
from pprint import pprint
pprint (config.get('cloudmesh.launcher.recipies.mooc'))
```

```
{`description': `Deploys a Slurm cluster. One of the Vms is the Master, while the others re
 `image': `/static/img/launcher/mooc.png',
 `label': `slurm',
 `name': `Mooc',
 `parameters': [{`nodes': {`type': `integer', `value': 1}},
                {`other': {`type': `float', `value': 5.0}},
                {`types': {`type': `string',
                           `value': `enter vm or bare metal'}},
                {`selector': {`type': `selector',
                              `value': ['vm', `baremetal', `xyz']}},
                {`cloud': {`type': `selector', `value': ['india', `sierra']}}],
 `script': {`type': `sh',
            `value': `echo ``hello'' {{ selector }} -np {{ nodes }}'}}
```

5.4.4 Batch Queues

Cloudmesh includes a convenient interface to PBS qstat. It is using an ssh command to connect to the login node on which to execute qstat. The result of the call is available as a dict and you can print it to inspect it closer.

In addition cloudmesh has a customization that alloes resources in FUturegrid to be seperated by machine. This is necessary as the login node on india is also in control of the queues on a number of machines instead of just one.

The username for FutureGrid is loaded from the configuration yaml file in the example

```
import cloudmesh
```

```
username = cloudmesh.load().username()
```

```
india = cloudmesh.PBS(username, "india.futuregrid.org")
```

QStat

```
qstat = india.qstat()
```

Next, let us list the nuber of clusters managed with the india PBS deployment

```
qstat.keys()
```

```
['echo.futuregrid.org',
 'india.futuregrid.org',
 'delta.futuregrid.org',
 'bravo.futuregrid.org']
```

```
qstat["india.futuregrid.org"].keys()
```

```
[u'1906163.i136',
 u'1906085.i136',
 u'1906155.i136',
 u'1906160.i136',
 u'1906164.i136',
 u'1906161.i136',
 u'1906086.i136',
 u'1906088.i136',
 u'1906084.i136',
 u'1906162.i136',
 u'1906087.i136',
 u'1906154.i136']
```

```
for jobname in qstat["india.futuregrid.org"]:
    job = qstat["india.futuregrid.org"][jobname]
    print jobname, job["job_state"]
```

```
1906163.i136 H
1906085.i136 R
1906155.i136 R
1906160.i136 H
1906164.i136 H
1906161.i136 H
1906086.i136 R
1906088.i136 R
1906084.i136 R
1906162.i136 H
1906087.i136 R
1906154.i136 Q
```

QInfo

```
qinfo = india.qinfo()
```

```
To list the queue names, simply print the keys.
```

```
print qinfo["india.futuregrid.org"].keys()
```

```
['reserved', 'systest', 'batch', 'b534', 'long', 'interactive']
```

More information is available when you inspect the dict.

5.4.5 Cloudmesh VM API

Cloudmesh supports the simple management of heterogeneous virtual machines from a variety of cloud farmeworks, including Openstack, Azure, AWS, Eucalyptus and but also EC2 complient clouds. This page shows how to use Cloudmesh functions in python by introducing a few examples on how to launch end terminate instances via the API.

Import Cloudmesh

A simple import allows you to enable all features of Cloudmesh in Python.

```
import cloudmesh
```

Choose a place to initialize

Cloudmesh provides *yaml* or *mongo* option where to load basic information. *yaml* relies on the yaml files in the $HOME/.cloudmesh directory, *mongo* retrieves information from the mongo database.

```
mesh = cloudmesh.mesh("mongo")
```

Get a username

In most Cloudmesh functions, you need to provide a username to tell the server who is going to use cloud services.

```
username = cloudmesh.load().username()
```

Activate the user account

With the activation, Cloudmesh is verified to connect IaaS cloud services.

```
mesh.activate(username)
```

Get Flavors

Available flavors can be listed with the following function.

```
mesh.flavors(cm_user_id=username, clouds=["india"])
```

refresh function updates the data from the IaaS cloud. The cached data in the mongo database will be updated.

```
mesh.refresh(username,types=['flavors'],names=["india"])
```

Start VM

A simple function *start* provides a quick launch of vm instances in cloudmesh.

```
result = mesh.start("india", username)
```

Delete VM

If you know the id of the virtual machine that you want to destroy, *delete* funcion in cloudmesh simply terminate the instance. This example deletes the vm that we just launched above by getting the id from the result dict.

```
server = result['server']['id']
mesh.delete(cloud, server, username)
```

VM Images

The functions about vm images provide an available vm images on a selected cloud.

```
mesh.images(clouds=['india'],cm_user_id=username)
```

Select a image

If you know the name of the vm image, you can specify the vm image to user it later. In this example, we choose Ubuntu trusty 14.04 image.

```
image=mesh.image('india','futuregrid/ubuntu-14.04')
```

Select a flavor

Flavor is also selectable. The selected image or flavor can be used to set a default image or a default flavor.

```
flavor = mesh.flavor('india', 'm1.small')
```

Set a default flavor or image

Each cloud must have a default image or a default flavor to launch vm instances in a simple step. default function provides a way to set default values for an image and a flavor. In this example, we use *image*, *flavor* variables which created from the examples above.

Need a help for the function?

You can execute a cell with a function name and a single question mark (?) to see a short description of a function. A double question marks (??) provide a source code with a docstring of the function. Try *mesh.default?* or *mesh.default??*

```
defaults = mesh.default('india', 'image', image)
```

```
defaults = mesh.default('india', 'flavor', flavor)
```

More options to launch a VM instance

When you create a new VM instance, you can also choose multiple options such as a flavor, an image, or a key associated with the instance. *start()* function accepts keyword parameters as a user input of these options. To see a brief description of the function, try *mesh.start?* in the IPython Notebook cell.

Available options are:

- prefix: The VM instance name starts with
- index: The number of vm instances (auto increment)

```
result = mesh.start("india", "Taylor", index=5, prefix="hoho")
```

Vitual Machine Name

In Cloudmesh, the default VM name consists of your username and an auto incremented number, for example, *alex_1*. Couple of functions allow you to manage or modify the VM name as you wish.

```
mesh.vmname()
```

```
u'Alex_37'
```

vmname() without parameters returns the current VM name that you can use. If you want to use a different name, you can specify *prefix='*' and *idx='*' as parameters.

```
mesh.vmname("Brian", 10)
```

```
u'Brian_10'
```

vmname_next() returns a VM name with an increased index number.

```
mesh.vmname_next()
```

```
u'Brian_11'
```

There are some tricks to update the index number in *vmname()* function.

```
mesh.vmname("Alex", "+5")
```

```
u'Alex_21'
```

Assign a public IP address to the VM

assign_public_ip() function obtains a public IP address and assign it to the VM.

```
ip=mesh.assign_public_ip('india', server, username)
```

SSH to the VM

Once you obtained the public ip address, you can execute a test command via SSH to the VM. You can use wait() function with retry options. Otherwise, you can simply use ssh_execute() function.

```
mesh.wait(ipaddr=ip, command="uname -a", interval=5, retry=10)
```

```
mesh.ssh_execute(ipaddr=ip, command="uname -a")
```

More examples follows soon, for example listing vm instances.

5.5 Cloudmesh Shell

5.5.1 Cloudmesh Shell API

```
import cloudmesh
```

```
print cloudmesh.shell("help")
```

```
Documented commands (type help <topic>):
========================================
EOF       edit      help       label   plugins  register        user
banner    exec      image      list    project  script          var
clear     exp       info       man     py       security_group  verbose
cloud     flavor    init       metric  q        storm           version
defaults  graphviz  inventory  open    quit     timer           vm
dot2      group     keys       pause   rain     use             web

Gui Commands
============
web

Cloud Commands
==============
cloud     group  inventory  rain            storm  keys
defaults  image  list       register        user   project
flavor    init   metric     security_group  vm
```

5.5.2 Cloudmesh Shell

Initialization

```
import cloudmesh

print cloudmesh.shell("help")
```

```
Documented commands (type help <topic>):
========================================
EOF       edit      help       label   plugins  register        user
banner    exec      image      list    project  script          var
clear     exp       info       man     py       security_group  verbose
cloud     flavor    init       metric  q        storm           version
defaults  graphviz  inventory  open    quit     timer           vm
dot2      group     keys       pause   rain     use             web

Gui Commands
============
web

Cloud Commands
==============
cloud     group  inventory  rain            storm  keys
defaults  image  list       register        user   project
flavor    init   metric     security_group  vm
```

Activating Clouds

In order for cloudmesh to work with multiple coulds, we need to find out first which clouds are available. Users can add their own clouds later which we describe in the registration section.

Let us inspect what is already available by invoking the list command

```
print cloudmesh.shell("cloud list")
```

```
+----------+----------+
| cloud    | active   |
+==========+==========+
| alamo    |          |
+----------+----------+
| aws      |          |
+----------+----------+
| azure    |          |
+----------+----------+
| hp       |          |
+----------+----------+
| hp_east  |          |
+----------+----------+
| india    | True     |
+----------+----------+
| sierra   |          |
+----------+----------+
```

As you see we have a number of clouds, but none of them is already active. Thus we need to first activate a cloud. We assume that you have an account on FutureGrid. Let us activate the cloud india

```
print cloudmesh.shell("cloud on india")
```

```
* india
Refreshing gvonlasz servers india ->
Refresh time: 0.361881971359
Store time: 0.00755286216736
[92mcloud 'india' activated.[0m
```

We also have a conveniet interactive selector to select a cloud to work with, that however does not work with ipython

```
"cloud select"
```

or you may also input “cloud select india” to select a specific cloud india

To check if the cloud was activated, simply use the list command again

```
print cloudmesh.shell("cloud list")
```

```
+----------+----------+
| cloud    | active   |
+==========+==========+
| alamo    |          |
+----------+----------+
| aws      |          |
+----------+----------+
| azure    |          |
+----------+----------+
| hp       |          |
+----------+----------+
| hp_east  |          |
+----------+----------+
| india    | True     |
+----------+----------+
| sierra   |          |
+----------+----------+
```

Starting VMs

Now let us see how to start VMs on a cloud, here is how to start a VM on cloud india

```
print cloudmesh.shell("vm start --cloud=india --image=futuregrid/ubuntu-14.04 --flavor=m1.small")
```

```
* india
Refreshing gvonlasz servers india ->
Refresh time: 0.340050935745
Store time: 0.00301289558411
* india
Refreshing gvonlasz flavors india ->
Refresh time: 0.149047136307
Store time: 0.0033278465271
Refreshing gvonlasz images india ->
Refresh time: 0.72811293602
Store time: 0.0171279907227

# ######################################################################
# Starting vm->gvonlasz_1 on cloud->india using image->m1.small, flavor->futuregrid/ubuntu-14.04, ke
# ######################################################################
job status: PENDING
to check realtime vm status: list vm --refresh
```

You may don't know what images or flavors are available on the cloud, or you don't want to type a long command every time you start a VM, things can get a lot easier by performing some setting up...

set default image

You can invoke command "cloud set image india" to set a image interactively, however this does not work with ipython. Then we can use the following commands to get a list of images first and set default image by giving its image name or image id

```
print cloudmesh.shell("list image india --refresh")
```

```
* india
Refreshing gvonlasz servers india ->
Refresh time: 0.101712942123
Store time: 0.0334851741791
* india
Refreshing gvonlasz images india ->
Refresh time: 0.260214090347
Store time: 0.0410861968994
+--------------------------------------+
images of cloud 'india'
+-----------------------------------------------------+----------+-----------------------------------
| name                                                | status   | id
+=====================================================+==========+===================================
| futuregrid/fedora-19                                | ACTIVE   | 5bf9905b-3314-4b83-8609-d94a3b4c89a
+-----------------------------------------------------+----------+-----------------------------------
| salsahpc/cloud-mooc-m1-large-4GB                    | ACTIVE   | 384ca88c-f674-4d3d-999f-353fc691560
+-----------------------------------------------------+----------+-----------------------------------
| CentOS6                                             | ACTIVE   | ad35042c-d242-4514-bde0-138718b5aa
+-----------------------------------------------------+----------+-----------------------------------
| SL64-blank-sparse10gb-C vmdk                        | ACTIVE   | f480f1e7-38af-4410-bc13-6a1e43bf58
+-----------------------------------------------------+----------+-----------------------------------
| fg7/rynge/centos6-v1                                | ACTIVE   | ac464f65-7175-44df-9e0e-34b1a32f8a
```

```
+----------------------------------------------------+----------+------------------------------------
| futuregrid/centos-6                                | ACTIVE   | 81b27cb5-4f8b-4583-afd4-1901053f6a2
+----------------------------------------------------+----------+------------------------------------
| DaLiAna-vm2014e-geant4.10.vmdk Jan best            | ACTIVE   | f4f9821f-881b-433a-97ab-1ad87273c5
+----------------------------------------------------+----------+------------------------------------
| ubuntu-13.10                                       | ACTIVE   | cd844b12-f138-414a-8cd5-2c977f0a23
+----------------------------------------------------+----------+------------------------------------
| fg101/richieriee/my-ubuntu-01                      | ACTIVE   | 7915f335-d1a9-4380-a9f3-159eb67d72
+----------------------------------------------------+----------+------------------------------------
| sl64-gluex-vm2014e-40gb.vmdk Justin                | ACTIVE   | f51fc1e4-d495-4376-824b-19000c41ab
+----------------------------------------------------+----------+------------------------------------
| fg10/jcharcal/centos6.5_x86_64                     | ACTIVE   | 28f2ad75-8f42-4079-85e1-3d4fe98631
+----------------------------------------------------+----------+------------------------------------
| balewski/kernel-2.6.32-431.5.1-sl65                | ACTIVE   | 00e935c3-82a6-499c-9056-6db37d2743
+----------------------------------------------------+----------+------------------------------------
| SL64-blank-sparse40GB vmdk                         | ACTIVE   | 882871a5-3157-426a-b7bc-10cca91860
+----------------------------------------------------+----------+------------------------------------
| futuregrid/fedora-20                               | ACTIVE   | 928b52df-06cf-46b6-a6a6-af25a6a147
+----------------------------------------------------+----------+------------------------------------
| futuregrid/ubuntu-12.04                            | ACTIVE   | 9e1765c7-f554-4a10-a8e5-e2d3bf7590
+----------------------------------------------------+----------+------------------------------------
| Ubuntu-12.04-blank2.vmdk                           | ACTIVE   | b49324f8-d2c3-413a-8f8c-83b498e815
+----------------------------------------------------+----------+------------------------------------
| balewski/ramdisk-2.6.32-431.5.1-sl65               | ACTIVE   | 9589cf93-f0e6-45d8-930f-63dd2d9c2f
+----------------------------------------------------+----------+------------------------------------
| sl6_x64-qemu  french ?bad                          | ACTIVE   | c9ca9852-dfcb-4ceb-a164-a58c895945
+----------------------------------------------------+----------+------------------------------------
| balewski/sl6.5-blank-80gb-b works                  | ACTIVE   | 5cb90f55-c3a1-468b-b60a-0ecd589baf
+----------------------------------------------------+----------+------------------------------------
| grp17Cent                                          | ACTIVE   | 3fcf5075-f8e4-44db-b963-183de5a17c
+----------------------------------------------------+----------+------------------------------------
| futuregrid/ubuntu-14.04                            | ACTIVE   | 02cf1545-dd83-493a-986e-583d53ee37
+----------------------------------------------------+----------+------------------------------------
| ndssl-vt/ubuntu-12.04-small                        | ACTIVE   | b9455041-407b-488b-a3f1-5dce6c480b
+----------------------------------------------------+----------+------------------------------------
| cglmoocs/ipython                                   | ACTIVE   | 0eff8bfc-e455-429e-9dfb-e16488d410
+----------------------------------------------------+----------+------------------------------------
| balewski/daliana-vm2014e2-sl6.5-geant4.10-root5.34 | ACTIVE   | 64866d96-8a9e-4d7c-983f-90b0bb5943
+----------------------------------------------------+----------+------------------------------------
| DaLiAna-vm2014d-SL64-A.vmdk                        | ACTIVE   | ff5bb4de-d7e9-4bfb-b42b-cec9ed8d2d
+----------------------------------------------------+----------+------------------------------------
count: 25
+-------+
```

```
print cloudmesh.shell("cloud")
```

```
+--------------------------+--------+
| cloud                    | active |
+--------------------------+--------+
| alamo                    | True   |
| aws                      | True   |
| azure                    | True   |
| dreamhost                |        |
| hp                       |        |
| hp_east                  |        |
| india_eucalyptus         |        |
| india_openstack_havana   | True   |
| sierra_eucalyptus        |        |
| sierra_openstack_grizzly | True   |
+--------------------------+--------+
count: 10
+--------+

print cloudmesh.shell("cloud set image india --image=futuregrid/ubuntu-14.04")

* india
Refreshing gvonlasz servers india ->
Refresh time: 0.328827857971
Store time: 0.00358605384827
* india
Refreshing gvonlasz images india ->
Refresh time: 0.438119888306
Store time: 0.0100111961365
[92m'futuregrid/ubuntu-14.04' is selected[0m
```

set default flavor

Similar as setting default image, to set up default flavor interactively, "cloud set flavor india", otherwise you may get a list of flavors then set by giving flavor name or flavor id

```
print cloudmesh.shell("list flavor india --refresh")

* india
Refreshing gvonlasz servers india ->
Refresh time: 0.349723100662
Store time: 0.00250005722046
* india
Refreshing gvonlasz flavors india ->
Refresh time: 0.157089948654
Store time: 0.00244212150574
+----------------------+
flavors of cloud 'india'
+------+-----------+---------+-------+--------+----------------------+
|   id | name      |   vcpus |   ram |   disk | refresh time         |
+======+===========+=========+=======+========+======================+
|    1 | m1.tiny   |       1 |   512 |      0 | 2014-08-30T16-08-55Z |
+------+-----------+---------+-------+--------+----------------------+
|    3 | m1.medium |       2 |  4096 |     40 | 2014-08-30T16-08-55Z |
+------+-----------+---------+-------+--------+----------------------+
|    2 | m1.small  |       1 |  2048 |     20 | 2014-08-30T16-08-55Z |
+------+-----------+---------+-------+--------+----------------------+
|    5 | m1.xlarge |       8 | 16384 |    160 | 2014-08-30T16-08-55Z |
+------+-----------+---------+-------+--------+----------------------+
|    4 | m1.large  |       4 |  8192 |     80 | 2014-08-30T16-08-55Z |
+------+-----------+---------+-------+--------+----------------------+
count: 5
+------+

print cloudmesh.shell("cloud set flavor india --flavorid=2")

* india
Refreshing gvonlasz servers india ->
Refresh time: 0.300434112549
Store time: 0.00324892997742
* india
```

```
Refreshing gvonlasz flavors india ->
Refresh time: 0.161976099014
Store time: 0.00268888473511
[92m'm1.small' is selected[0m
```

set default cloud

If you want to make things even more convenient, you can set a default cloud or select a cloud to work with so that you don't have to type in a cloud everytime you need to specify a cloud, to set india as default cloud

```
print cloudmesh.shell("cloud set default india")
```

to select a cloud

```
print cloudmesh.shell("cloud select india")
```

You can see a selected cloud as a temporarily default cloud to work with.

For more details of using command cloud to set up a cloud

```
print cloudmesh.shell("cloud -h")
```

simple way to start a VM

After all setting up above, now you can start a VM simply by typing in

```
print cloudmesh.shell("vm start")
```

set default VM name

If the user doesn't provide a name while starting VMs, cloudmesh will generate labels for them. The default form to name VMs is prefix_index, where prefix is a string and index is an non-negative integer. If a index is used, the index value will be automatically added by one waiting to be used for next VM. To check your current prefix and index

```
print cloudmesh.shell("label")
```

To change the prefix and/or reset index(e.g. to abc and 3)

```
print cloudmesh.shell("label --prefix=abc --id=3")
```

Refreshing VM status

After you have started or deleted VMs, you may want to check clouds' VMs status. To refresh cloud india's VMs information

```
print cloudmesh.shell("list vm india --refresh")
```

Starting multiple VMs

Sometimes we want to start more than one VM at the same time, we can choose the option –count=int where int is the number of VMs you want to start. For example, to start 5 VMs on india

```
print cloudmesh.shell("vm start --cloud=india --count=5")
```

Deleting VMs

To delete one VM is easy, what if we want to delete 1000 VMs, we need a more convenient way to do it. Cloudmesh shell provides several methods to find the VMs and delete them, you may think there are two phases of VM deletion, searching and deleting. Here are some examples:

to delete all VMs of cloud india

```
print cloudmesh.shell("vm delete --cloud=india --force")
```

Note here we use the option "–force", without it the shell will give you a list of VMs to delete and ask for your confirmation.

to delete a VM by giving its name (you may always provide a cloud unless you have specified a default cloud or have selected a cloud)

```
print cloudmesh.shell("vm delete --cloud=india abc_2 --force")
```

to delete a VM by group

```
print cloudmesh.shell("vm delete --cloud=india --goup=testgroup --force")
```

We can also narrow the search result by giving more search conditions. For example, to delete VMs of cloud india that they are also in the group 'testgroup' and they have the prefix name 'abc' and their indices' range is no greater than 100

```
print cloudmesh.shell("vm delete --cloud=india --goup=testgroup --prefix=abc --range=,100 --force")
```

For more details for command vm

```
print cloudmesh.shell("vm -h")
```

CHAPTER

SIX

IAAS

6.1 OpenStack Clouds

OpenStack is an open-sourced, IaaS cloud computing platform founded by Rackspace Hosting and NASA, and used widely in industry. Some of the modules that it provides to users includess compute, storage and network components allowing users to get easily stated in cloud computing.

Many clouds exist that have OpenStack as a foundation as part of the cloudservices offered to users. Some of them are free, some are commercial services.

6.1.1 OpenStack on FutureGrid

Login

Currently FutureGrid OpenStack Havana installed on India and Openstack Grizzly on sierra. There are minor differences with these versions, but as Havana is the more recent one we will be focussing on that deployment.

To use it you need to first log into sierra and prepare your Openstack credentials (Make sure to replace the 'username' with your actual FG username):

```
$ ssh username@india.futuregrid.org
```

Setup OpenStack Environment

In case you like to use the shell commandline tools you can load them with

```
$ module load novaclient
```

Creating the novarc file

An initial novarc file is currently created for you automatically and can be activated wih

```
$ source ~/.futuregrid/openstack_havana/novarc
```

In future this file will be created with the help of cloudmesh simplifying access to multiple heterogeneous clouds on FutureGrid.

List flavors

To list the flavors, please execute the following command

```
$ nova flavor-list
```

Everything is fine, if you see an output similar to

```
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+----------
| ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor | Is_Public | extra_spec
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+----------
| 1  | m1.tiny   | 512       | 0    | 0         |      | 1     | 1.0         | True      | {}
| 2  | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0         | True      | {}
| 3  | m1.medium | 4096      | 40   | 0         |      | 2     | 1.0         | True      | {}
| 4  | m1.large  | 8192      | 80   | 0         |      | 4     | 1.0         | True      | {}
| 5  | m1.xlarge | 16384     | 160  | 0         |      | 8     | 1.0         | True      | {}
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+----------
```

If not your environment may not be set up correctly. Make sure that you follow the steps in this section and the account management section carefully.

List images

After you got the flavor list, you can list the current set of uploaded images with the nova image-list command:

```
$ nova image-list
```

You will see an output similar to:

```
+--------------------------------------+-------------------------+--------+--------+
| ID                                   | Name                    | Status | Server |
+--------------------------------------+-------------------------+--------+--------+
| 18c437e5-d65e-418f-a739-9604cef8ab33 | futuregrid/fedora-18    | ACTIVE |        |
| 1a5fd55e-79b9-4dd5-ae9b-ea10ef3156e9 | futuregrid/ubuntu-12.04 | ACTIVE |        |
+--------------------------------------+-------------------------+--------+--------+
```

Hint

$USER is your username on sierra machine.

Key management

To start a virtual machine you must first upload a key to the cloud. This can be easily done in the following way:

```
$ nova keypair-add $USER-key > ~/.ssh/$USER-key
$ chmod 600 ~/.ssh/$USER-key
$ nova keypair-list
+---------------+-------------------------------------------------+
| Name          | Fingerprint                                     |
+---------------+-------------------------------------------------+
| <USER>-key    | ab:a6:63:82:dd:08:d3:bc:c0:21:56:4c:e2:bb:22:ac |
+---------------+-------------------------------------------------+
```

Where USER is your login name on sierra.

Make sure you are not already having the key with that name in order to avoid overwriting it in the cloud. Thus be extra careful to execute this step twice. Often it is the case that you already have a key in your ~/.ssh directory that you may want to use. For example if you use rsa, your key will be located at ~/.ssh/id_rsa.pub.

Managing security groups

In the next step we need to make sure that the security groups allow us to log into the VMs. To do so we create the following policies as part of our default security policies. Not that when you are in a group project this may already have been done for you by another group member. We will add ICMP and port 22 on default group:

```
$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
$ nova secgroup-list-rules default
```

You will see the following output if everything went correctly:

```
+-------------+-----------+---------+-----------+--------------+
| IP Protocol | From Port | To Port | IP Range  | Source Group |
+-------------+-----------+---------+-----------+--------------+
| icmp        | -1        | -1      | 0.0.0.0/0 |              |
| tcp         | 22        | 22      | 0.0.0.0/0 |              |
+-------------+-----------+---------+-----------+--------------+
```

Booting an image

To boot an instance you simply can now use the command:

```
$ nova boot --flavor m1.small \
            --image "futuregrid/ubuntu-12.04" \
            --key_name $USER-key $USER-001
```

If everything went correctly, you will see an output similar to:

```
+-----------------------------+--------------------------------------+
| Property                    | Value                                |
+-----------------------------+--------------------------------------+
| status                      | BUILD                                |
| updated                     | 2013-05-15T20:32:03Z                 |
| OS-EXT-STS:task_state       | scheduling                           |
| key_name                    | <USER>-key                           |
| image                       | futuregrid/ubuntu-12.04              |
| hostId                      |                                      |
| OS-EXT-STS:vm_state         | building                             |
| flavor                      | m1.small                             |
| id                          | e15ad5b6-c3f0-4c07-996c-3bbe709a63b7 |
| security_groups             | [{u'name': u'default'}]              |
| user_id                     | 3bd2d773911c4502982e5c2cd81437f7     |
| name                        | vm001                                |
| adminPass                   | KgiKjek99dgk                         |
| tenant_id                   | b7ea98db7b3449b184b58d28e80c7541     |
| created                     | 2013-05-15T20:32:03Z                 |
| OS-DCF:diskConfig           | MANUAL                               |
| metadata                    | {}                                   |
| accessIPv4                  |                                      |
| accessIPv6                  |                                      |
| progress                    | 0                                    |
| OS-EXT-STS:power_state      | 0                                    |
| OS-EXT-AZ:availability_zone | None                                  |
| config_drive                |                                       |
+-----------------------------+---------------------------------------+
```

Where USER is your login name on sierra.

List running images

To check if your instance is active you can repeatedly issue the list command and monitor the Status field in the table:

```
$ nova list
```

```
+--------------------------------------+----------------+--------+----------------------+
| ID                                   | Name           | Status | Networks             |
+--------------------------------------+----------------+--------+----------------------+
| e15ad5b6-c3f0-4c07-996c-3bbe709a63b7 | <USER>-001     | ACTIVE | private=10.35.23.18 |
+--------------------------------------+----------------+--------+----------------------+
```

Once it has changed from for example BUILD to ACTIVE, you can log in. Pleas use the IP address provided under networks. Note that the first address is private and can not be reached from outside sierra:

```
$ ssh -l ubuntu -i ~/.ssh/$USER-key 10.35.23.18
```

If you see a warning similar to:

```
Add correct host key in /home/<USER>/.ssh/known_hosts to get rid of this message.
Offending key in /home/<$USER>/.ssh/known_hosts:3
```

you need to delete the offending host key from .ssh/known_hosts.

Use block storage

You can create a block storage with the volume-create command. A volume is useful as you can store data in it and associate that particular volumen to a VM. Hence, if you delete the VM, your volume and the data on t is still there to be reused. To create one 5G volume you can do

```
$ nova volume-create 5
```

To list the volumes you can use:

```
$ nova volume-list
+--------------------------------------+-----------+--------------+------+-------------+------------
| ID                                   | Status    | Display Name | Size | Volume Type | Attached to
+--------------------------------------+-----------+--------------+------+-------------+------------
| 6d0d8285-xxxx-xxxx-xxxx-xxxxxxxxxxxx | available | None         |  5   | None        |
+--------------------------------------+-----------+--------------+------+-------------+------------
```

To attach the volume to your instance you can use the volume-attach subcommand. Let us assume we like to attache it as "/dev/vdb", than you can use the command::

```
$ nova volume-attach $USER-001 6d0d8285-xxxx-xxxx-xxxx-xxxxxxxxxxxx "/dev/vdb"
```

Hint

Note thate $USER-001 refers to the name of the VM that we have created earlier with the boot command.

Next, let us login to your instance, make filesystem and mount it. Here's an example, mounting on /mnt:

```
$ ssh -l ubuntu -i ~/.ssh/$USER-key 10.35.23.18
ubuntu@<USER>-001:~$ sudo su -
root@<USER>-001:~# mkfs.ext4 /dev/vdb
root@<USER>-001:~# mount /dev/vdb /mnt
root@<USER>-001:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        20G  2.1G   17G  11% /
none            4.0K     0  4.0K   0% /sys/fs/cgroup
udev            998M  8.0K  998M   1% /dev
tmpfs           201M  236K  201M   1% /run
none            5.0M     0  5.0M   0% /run/lock
none           1002M     0 1002M   0% /run/shm
none            100M     0  100M   0% /run/user
/dev/vdb        4.8G   23M  4.2G   1% /mnt
```

When you want to detach it, unmount /mnt first, go back to sierra's login node and execute volume-detach:

```
root@<USER>-001:~# umount /mnt
root@<USER>-001:~# exit
ubuntu@<USER>-001:~$ exit

$ nova volume-detach $USER-001 6d0d8285-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Set up external access to your instance

So far we only used the internal IP address, but you can also assign an external address, so that you can log in from other machines than sierra. Firts, Create an external ip address with:

```
$ nova floating-ip-create

+-----------------+-------------+----------+------+
| Ip              | Instance Id | Fixed Ip | Pool |
+-----------------+-------------+----------+------+
| 198.202.120.193 | None        | None     | nova |
+-----------------+-------------+----------+------+
```

Next, put it on your instance with:

```
$ nova add-floating-ip $USER-001 198.202.120.193
$ nova floating-ip-list

+-----------------+--------------------------------------+-------------+------+
| Ip              | Instance Id                          | Fixed Ip    | Pool |
+-----------------+--------------------------------------+-------------+------+
| 198.202.120.193 | c0bd849a-221a-4e53-bf7b-7097541a9bcc | 10.35.23.20 | nova |
+-----------------+--------------------------------------+-------------+------+
```

Now you should be able to ping and ssh from external and can use the given ip address.

Make a snapshot of an instance

To allow snapshots, you must use the following convention:

- use your project number fg### in the prefix of your snapshot name followed by a /
- If needed you can also add your username as a prefix in addition to the project number.

Let us assume your project is fg101 and you want to save the image with by reminding you it was a my-ubuntu-01 image you want to key. Than you can issue on sierra the following command:

```
$ nova image-create $USER-001 fg101/$USER/my-ubuntu-01
$ nova image-list
+--------------------------------------+----------------------------+--------+-----------------------
| ID                                   | Name                       | Status | Server
+--------------------------------------+----------------------------+--------+-----------------------
| 18c437e5-d65e-418f-a739-9604cef8ab33 | futuregrid/fedora-18       | ACTIVE |
| 1a5fd55e-79b9-4dd5-ae9b-ea10ef3156e9 | futuregrid/ubuntu-12.04    | ACTIVE |
| f43375b4-44d3-4350-a9a8-a73f35589344 | fg101/<USER>/my-ubuntu-01  | ACTIVE | c0bd849a-221a-4e53-bf
+--------------------------------------+----------------------------+--------+-----------------------
```

If you want to download your customized image, you can do it with this:

```
$ glance image-download --file "my-ubuntu-01.img" "fg101/$USER/custom-ubuntu-01"
```

Hint

Please note that images not following this convention will be deleted.

Automate some initial configuration

You may want to install some packages into the image, enable root, and add ssh authorized_keys. With the OpenStack cloud-init such steps can be simplified.

Create a file(mycloudinit.txt) containing these lines:

```
#cloud-config

# Enable root login.
disable_root: false

# Install packages.
packages:
- apt-show-versions
- wget
- build-essential

# Add some more ssh public keys.
ssh_authorized_keys:
- ssh-rsa AAAfkdfeiekf....fES7060rb myuser@s1
- ssh-rsa AAAAAAkgeig78...skdfjeigi myuser@myhost
```

Now boot your instance with –user-data mycloudinit.txt like this:

```
$ nova boot --flavor m1.small \
            --image "futuregrid/ubuntu-12.04" \
            --key_name $USER-key \
            --user-data mycloudinit.txt $USER-002
```

You should be able to login to <USER>-002 as root, and the added packages are installed.

Get the latest version of Ubuntu Cloud Image and upload it to the OpenStack

Todo

In future we will just host these images so we do not duplicate them on the server

---

Several versions of Ubuntu cloud images are available at http://cloud-images.ubuntu.com/. Choose the version you want and download the file name with ******-cloudimg-amd64-disk1.img. For example, downloading Ubuntu-13.04(Raring Ringtail)is like this:

```
$ wget http://cloud-images.ubuntu.com/raring/current/raring-server-cloudimg-amd...
```

You can upload the image with the glance client like this:

```
$ glance image-create \
       --name fg101/$USER/myimages/ubuntu-13.04 \
       --disk-format qcow2 \
       --container-format bare \
       --file raring-server-cloudimg-amd64-disk1.img
```

Now your new image is listed on `nova image-list`and will be available when the status become “ACTIVE”.

Delete your instance

1. You can delete your instance with:

   ```
   $ nova delete $USER-002
   ```

   Please do not forget to also delete your 001 vm if you no longer need it

How to change your password

1. Sometimes, users accidentally send password to a collaborator/support for debugging, and then regret. When you put yourself in the situation by mistake, don’t worry. Just use keystone client and reset your password with:

   ```
   $ keystone password-update
   ```

   * Remember, you will also need to change it in your novarc. This can be achieved by either editing your novarc file directly, or by editing the file ~/.futuregrid/cloudmesh.yaml and recreating your novarc file.

Things to do when you need Euca2ools or EC2 interfaces

Even though the nova client and protocols will provide you with more advanced features, some users still want to access OpenStack with EC2 compatible tools. One such tool are the euca2tools. We explain briefly how you can access them.

1. Create a directory for putting eucarc, and create pk.pem, cert.pem and cacert.pem:

   ```
   cd ~/.futuregrid/openstack_havana
   nova x509-create-cert
   nova x509-get-root-cert
   ls -la
   ```

2. Create EC2_ACCESS_KEY and EC2_SECRET_KEY:

   ```
   keystone ec2-credentials-create
   ```

3. Create the file calle *~/.futuregrid/openstack_havana/eucarc* and put your EC2_ACCESS_KEY and EC2_SECRET_KEY that you obtained from the previous command into this file:

```
export NOVA_KEY_DIR=$(cd $(dirname ${BASH_SOURCE[0]}) && pwd)
export EC2_ACCESS_KEY="Your EC2_ACCESS_KEY"
export EC2_SECRET_KEY="Your EC2_SECRET_KEY"
export EC2_URL="http://i57r.idp.iu.futuregrid.org:8773/services/Cloud"
export S3_URL="http://i57r.idp.iu.futuregrid.org:3333"
export EC2_USER_ID=11
export EC2_PRIVATE_KEY=${NOVA_KEY_DIR}/pk.pem
export EC2_CERT=${NOVA_KEY_DIR}/cert.pem
export NOVA_CERT=${NOVA_KEY_DIR}/cacert.pem
export EUCALYPTUS_CERT=${NOVA_CERT}
alias ec2-bundle-image="ec2-bundle-image --cert ${EC2_CERT} --privatekey ${EC2_PRIVATE_KEY} --us
alias ec2-upload-bundle="ec2-upload-bundle -a ${EC2_ACCESS_KEY} -s ${EC2_SECRET_KEY} --url ${S3_
```

4. Confirm if euca2ools works:

```
module load euca2ools/3.1.0
source ~/.futuregrid/openstack_havana/eucarc
euca-describe-images
euca-describe-instances
```

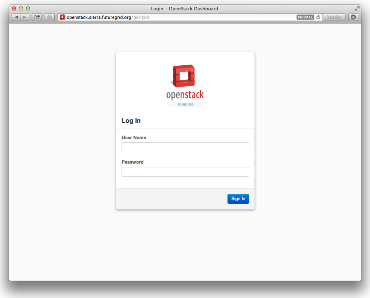
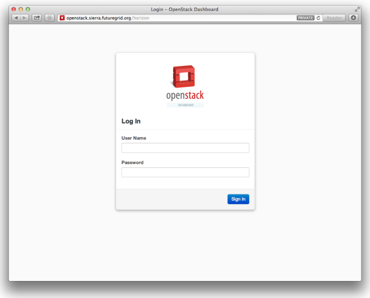
Note: Here's our known issues on using euca2ools or ec2 interface.

- euca-upload-bundle with Boto 2.25.0 fails with "S3ResponseError: 404 Not Found".
- tagging function such as euca-create-tags, euca-describe-tags fail with "InvalidRequest: The request is invalid."

Horizon GUI

Horizon is a graphical user interface/dashbooard for OpenStack. For starting up VMs and stoping them by hand horizon may be a good mechanism to manage your Virtual machines. We have currently the following horizon deployments available. However, please note that on Alamo an older version of Openstack is run.

Table 6.1: Horizon endpoints

Image	Version	Machine	Protocol	Description
 Login - OpenStack Dashboard openstack Log In User Name Password Sign In	Havana	India	Native OpenStack	India offers a Graphical user interface to access OpenStack. For those interested in only managing a few images this may be a good way to start. The link to the GUI is https://openstack-h.india.futuregrid.org/horizon The password can be found by following the method dicussed above.
 Login - OpenStack Dashboard openstack Log In User Name Password Sign In	Grizzly	Sierra	Native OpenStack	Sierra offers a Graphical user interface to access OpenStack. For those interested in only managing a few images this may be a good way to start. The link to the GUI is http://openstack-sierra.futuregrid.org/horizon The password can be found by following the method dicussed above.

CHAPTER

SEVEN

PAAS

7.1 Using Hadoop in FutureGrid

Screencast

A screenncast of a subset of the information presented her is available at PC8h1CtVzH4.

We have various platforms that support Hadoop on FutureGrid. MyHadoop is probably the easiest solution offered for you. It provides the advantage that it is integrated into the queuing system and allows hadoop jobs to be run as batch job. This is of especial interest for classes that may run quickly out of resources if every students wants to run their hadoop application at the same time.

7.1.1 Running Hadoop as a Batch Job using MyHadoop

MapReduce is a programming model developed by Google. Their definition of MapReduce is as follows: "MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key." For more information about MapReduce, please see the Google paper here.

The Apache Hadoop Project provides an open source implementation of MapReduce and HDFS (Hadoop Distributed File System).

This tutorial illustrates how to run Apache Hadoop thru the batch systems on FutureGrid using the MyHadoop tool.

myHadoop on FutureGrid

MyHadoop is a set of scripts that configure and instantiate Hadoop as a batch job.

myHadoop 0.20.2 is currently installed on Alamo, Hotel, India, and Sierra FutureGrid systems.

Login into a machine tha has myHadoop installed

To run the example we need to firts log into a FutureGrid system that has myHadoop available. In this tutorial, we are executing from the sierral machine:

```
$ ssh portalname@sierra.futuregrid.org
```

Note that this also works on other FutureGrid machines such as india.

This machine accepts SSH public key and One Time Password (OTP) logins only. If you do not have a public key set up, you will be prompted for a password. This is *not* your FutureGrid password, but the One Time Password generated from your OTP token. Do not type your FutureGrid password, it will not work. If you do not have a token or public key, you will not be able to login. The portalname is your account name that allows you to log into the FutureGrid portal.

Load the needed modules

Next, we need to load the myHadoop module. On some FutureGrid systems, you may also need to load the "torque" module as well if qstat is not already in your environment:

```
$ module load myhadoop
SUN Java JDK version 1.6.0 (x86_64 architecture) loaded
Apache Hadoop Common version 0.20.203.0 loaded
myHadoop version 0.2a loaded
```

Before we can submit it we still need to load the module java as Haddop relies on java:

```
module add java
```

Run the Example

To run the example we need to create a script to tell the queing system how to run it. We are providing the following script that you can store in a file pbs-example.sh.

You can paste and copy it from here, or just copy it via:

```
cp $MY_HADOOP_HOME/pbs-example.sh .
```

This script includes information about which queue hadoop should be run in. To find out more about the queuing system, please see the HPC services section. The pbs-example.sh script we use in this example looks as follows:

```
#!/bin/bash

#PBS -q batch
#PBS -N hadoop_job
#PBS -l nodes=4:ppn=8
#PBS -o hadoop_run.out
#PBS -j oe
#PBS -V

module add java

### Run the myHadoop environment script to set the appropriate variables
#
# Note: ensure that the variables are set correctly in bin/setenv.sh
. /N/soft/myHadoop/bin/setenv.sh

#### Set this to the directory where Hadoop configs should be generated
# Don't change the name of this variable (HADOOP_CONF_DIR) as it is
# required by Hadoop - all config files will be picked up from here
#
# Make sure that this is accessible to all nodes
export HADOOP_CONF_DIR="${HOME}/myHadoop-config"
```

```
#### Set up the configuration
# Make sure number of nodes is the same as what you have requested from PBS
# usage: $MY_HADOOP_HOME/bin/pbs-configure.sh -h
echo "Set up the configurations for myHadoop"
# this is the non-persistent mode
$MY_HADOOP_HOME/bin/pbs-configure.sh -n 4 -c $HADOOP_CONF_DIR
# this is the persistent mode
# $MY_HADOOP_HOME/bin/pbs-configure.sh -n 4 -c $HADOOP_CONF_DIR -p -d /oasis/cloudstor-group/HDFS
echo

#### Format HDFS, if this is the first time or not a persistent instance
echo "Format HDFS"
$HADOOP_HOME/bin/hadoop --config $HADOOP_CONF_DIR namenode -format
echo

#### Start the Hadoop cluster
echo "Start all Hadoop daemons"
$HADOOP_HOME/bin/start-all.sh
#$HADOOP_HOME/bin/hadoop dfsadmin -safemode leave
echo

#### Run your jobs here
echo "Run some test Hadoop jobs"
$HADOOP_HOME/bin/hadoop --config $HADOOP_CONF_DIR dfs -mkdir Data
$HADOOP_HOME/bin/hadoop --config $HADOOP_CONF_DIR dfs -copyFromLocal $MY_HADOOP_HOME/gutenberg Data
$HADOOP_HOME/bin/hadoop --config $HADOOP_CONF_DIR dfs -ls Data/gutenberg
$HADOOP_HOME/bin/hadoop --config $HADOOP_CONF_DIR jar $HADOOP_HOME/hadoop-0.20.2-examples.jar wordcou
$HADOOP_HOME/bin/hadoop --config $HADOOP_CONF_DIR dfs -ls Outputs
$HADOOP_HOME/bin/hadoop --config $HADOOP_CONF_DIR dfs -copyToLocal Outputs ${HOME}/Hadoop-Outputs
echo

#### Stop the Hadoop cluster
echo "Stop all Hadoop daemons"
$HADOOP_HOME/bin/stop-all.sh
echo

#### Clean up the working directories after job completion
echo "Clean up"
$MY_HADOOP_HOME/bin/pbs-cleanup.sh -n 4 -c $HADOOP_CONF_DIR
echo
```

Details of the Script

Let us examine this script in more detail. In the example script, a temporary directory to store Hadoop configuration files is specified as ${HOME}/myHadoop-config:

```
#### Set this to the directory where Hadoop configs should be generated
# Don't change the name of this variable (HADOOP_CONF_DIR) as it is
# required by Hadoop - all config files will be picked up from here
#
# Make sure that this is accessible to all nodes
export HADOOP_CONF_DIR="${HOME}/myHadoop-config"
```

The pbs-example.sh script runs the "wordcount" program from the hadoop-0.20.2-examples.jar. There is sample text data from the Project Gutenberg website located a $MY_HADOOP_HOME/gutenberg:

```
$ ls $MY_HADOOP_HOME/gutenberg
1342.txt.utf8
```

The following lines in the script create a data directory in HDFS. This directory is specified in $MY_HADOOP_HOME/bin/setenv.sh. To activate the environment, pleas execute:

```
source $MY_HADOOP_HOME/bin/setenv.sh
```

The next lines in the script will copy over the gutenberg data, executes the Hadoop job, and then copies the output back your ${HOME}/Hadoop-Outputs directory.

```
#### Run your jobs here
echo "Run some test Hadoop jobs"
$HADOOP_HOME/bin/hadoop --config $HADOOP_CONF_DIR dfs -mkdir Data
$HADOOP_HOME/bin/hadoop --config $HADOOP_CONF_DIR dfs -copyFromLocal $MY_HADOOP_HOME/gutenberg Data
$HADOOP_HOME/bin/hadoop --config $HADOOP_CONF_DIR dfs -ls Data/gutenberg
$HADOOP_HOME/bin/hadoop --config $HADOOP_CONF_DIR jar $HADOOP_HOME/hadoop-0.20.2-examples.jar wordcou
$HADOOP_HOME/bin/hadoop --config $HADOOP_CONF_DIR dfs -ls Outputs
$HADOOP_HOME/bin/hadoop --config $HADOOP_CONF_DIR dfs -copyToLocal Outputs ${HOME}/Hadoop-Outputs
```

Submission of the Hadoop job

Now submit the pbs-example.sh script to Hotel:

```
$ qsub $MY_HADOOP_HOME/pbs-example.sh
40256.svc.uc.futuregrid.org
```

The job will take about 5 minutes to complete. To monitor its status, type 'qstat'. The "R" means the job is running:

```
$ qstat
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
40256.svc                  hadoop_job       albert                 0 R batch
```

When it is done, the status of the job will be "C" meaning the job has completed (or it will no longer be displayed in qstat output). You should see a new hadoop_run.out file and an "Hadoop-Outputs" directory

```
$ qstat
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
40256.svc                  hadoop_job       albert         00:00:05 C batch
$ ls
Hadoop-Outputs hadoop_run.out
```

View results of the word count operation:

```
$ head Hadoop-Outputs/part-r-00000
"'After    1
"'My   1
"'Tis  2
"A 12
"About 2
"Ah!   2
"Ah!" 1
"Ah,   1
"All   2
"All!  1
```

Now to run you own custom Hadoop job, make a copy of the $MY_HADOOP_HOME/pbs-example.sh script and modify the lines described in Step 7.

Persistent Mode

The above example copies input to local HDFS scratch space you specified in $MY_HADOOP_HOME/bin/setenv.sh, runs MapReduce, and copies output from HDFS back to your home directory. This is called non-persistent mode and is good for small amounts of data. Alternatively, you can run in persistent mode which is good if you have access to a parallel file system or have a large amount of data that will not fit in scratch space. To enable persistent mode, follow the directions in pbs-example.sh.

Customizing Hadoop Settings

To modify any of the Hadoop settings like maximum_number_of_map_task, maximum_number_of_reduce_task, etc., make you own copy of myHadoop and customize the settings accordingly. For example:

1. Copy the $MY_HADOOP_HOME directory to your home directory:

   ```
   $ cp -r $MY_HADOOP_HOME $HOME/myHadoop
   ```

2. Then edit $HOME/myHadoop/pbs-example.sh and on line 16, replace it with:

   ```
   . ${HOME}/myHadoop/bin/setenv.sh
   ```

3. Similarly edit $HOME/myHadoop/bin/setenv.sh and on line 4, replace it with:

   ```
   export MY_HADOOP_HOME=$HOME/myHadoop
   ```

4. Customize the settings in the Hadoop files as needed in $HOME/myHadoop/etc
5. Submit your copy of pbs-example.sh:

   ```
   $ qsub $HOME/myHadoop/pbs-example.sh
   ```

Using a Different Installation of Hadoop

If you would like to use a different version of my Hadoop or have customized the Hadoop code in some way, you can specify a different installation of Hadoop by redefining the HADOOP_HOME variable after $MY_HADOOP_HOME/bin/setenv.sh is called within your own copy of pbs-example.sh:

```
### Run the myHadoop environment script to set the appropriate variables
#
# Note: ensure that the variables are set correctly in bin/setenv.sh
. /opt/myHadoop/bin/setenv.sh
export HADOOP_HOME=${HOME}/my-custom-hadoop
```

References

- Much of this information is copied from The MyHadoop Installation Instructions
- A screenncast of a subset of the information presented her is available at PC8h1CtVzH4.

CHAPTER

EIGHT

HPC

8.1 Login Nodes

Several of the clusters have High Performance Computing (HPC) services installed. Access to them is provided via a Linux Login node for each of the clusters on which these services are installed.

To access the login nodes you need a FG resource account and an SSH public key you have uploaded to FutureGrid (this process is described in the section about *s-accounts*. After you are part of a valid project and have a FutureGrid account, you can log into the FutureGrid resources with ssh. The resources include the following login nodes:

- bravo.futuregrid.org
- foxtrot.futuregrid.org
- india.futuregrid.org
- sierra.futuregrid.org
- xray.futuregrid.org

For example, assume your portalname is "portalname", than you can login to sierra as follows:

```
$ ssh portalname@sierra.futuregrid.org
Welcome to sierra.futuregrid.org
Last login: Thu Aug 12 19:19:22 2010 from ....
```

8.1.1 SSH Add

Sometimes you may wish to log in repeatedly in other machines while using a cached password. To do that you can use ssh agent and ssh add. First start the agent:

```
eval `ssh-agent`
```

Then add your key with:

```
ssh-add
```

Follow the instructions on the screen. Thus before you ssh in, you may want to use ssh agent. This way you do not have to repeatedly type in your key password.

8.1.2 SSH Config

Also you may want to setup your ~/.ssh/config file to create shortcut for the username and hosts on which you want to log in. Let us assume your username is albert, then add the following lines in the .ssh/config file:

```
Host india
      Hostname india.futuregrid.org
      User albert
```

This will allow you to log into the machine just while typing in:

```
ssh india
```

8.1.3 Modules

The login nodes have the modules package installed. It provides a convenient tool to adapt your environment and enables you to activate different packages and services dependent on your specific needs. The Modules are utilized to let you dynamically control your environment. Modules allows you to load and unload packages and ensure a coherent working environment. This ensures that your $PATH, $LD_LIBRARY_PATH, $LD_PRELOAD, and other environment variables are properly set, and that you can access the programs and libraries you need. For additional information about the Modules package you can consult the manual page.

To display the list of available modules:

```
$ module avail
```

To display the list of currently loaded modules:

```
$ module list
```

To add and remove packages from your environment you can use the *module load* and *module unload* commands:

```
$ module load <package name>/<optional package version>
$ module unload <package name>/<optional package version>
```

The available command are listed in the next table:

Table 8.1: Module commands

Command	Description
module avail	List all software packages available on the system.
module avail package	List all versions of package available on the system
module list	List all packages currently loaded in your environment.
module load package/version	Add the specified version of the package to your environment
module unload package	Remove the specified package from your environment.
module swap package_A package_B	Swap the loaded package (package_A) with another package (package_B).
module show package	Shows what changes will be made to your environment (e.g. paths to libraries and executables) by loading the specified package.

Example - List the currently loaded modules on sierra after login:

```
$ module list

Currently Loaded Modulefiles:
  1) torque/2.4.8   2) moab/5.4.0
```

Example - list the avialable modules on sierra:

```
$ module avail

----------------- /opt/Modules/3.2.8/modulefiles/applications ------------------
```

```
R/2.11.1(default)       hpcc/1.3.1(default)    velvet/1.0.15
git/1.7.10              ncbi/2.2.23(default)   wgs/6.1
gromacs/4.0.7(default) soapdenovo/1.04

------------------- /opt/Modules/3.2.8/modulefiles/compilers -------------------
cmake/2.8.1(default)        java/1.6.0-i586
intel/10.1                  java/1.6.0-x86_64(default)
intel/11.1(default)

------------------- /opt/Modules/3.2.8/modulefiles/debuggers -------------------
null                        totalview/8.8.0-2(default)

------------------- /opt/Modules/3.2.8/modulefiles/libraries -------------------
intelmpi/4.0.0.028(default)  openmpi/1.4.3-intel
mkl/10.2.5.035(default)      otf/1.7.0(default)
openmpi/1.4.2(default)       unimci/1.0.1(default)
openmpi/1.4.3-gnu            vampirtrace/intel-11.1/5.8.2

--------------------- /opt/Modules/3.2.8/modulefiles/tools ---------------------
cinderclient/1.0.4(default)   moab/5.4.0(default)
cloudmesh/0.8(default)        myhadoop/0.2a
euca2ools/1.2                 novaclient/2.13.0(default)
euca2ools/1.3.1               precip/0.1(default)
euca2ools/2.0.2(default)      python/2.7(default)
genesisII/2.7.0               python/2.7.2
glanceclient/0.9.0(default)   torque/2.4.8(default)
keystoneclient/0.2.3(default) vim/7.2
marmot/2.4.0(default)
```

Example - load the default version of a module (in this case git):

```
$ module load git
```

Please note that for loading the default you do not have to specify the version number.

8.1.4 Filesystem Layout

***Home* directories:** Home directories are accessible through the $HOME shell variable which are located at */N/u/<username>*. This is where users are encouraged to keep source files, configuration files and executables. Users should not run code from their $HOME directories. Please note that this is an NFS file system, and may result in slower access for some applications. We also advise the users to provide external backup storage at their home institution or a code repository. For example, we recommend that you use git or svn to make sure you backup your changes to the code. Also make sure you backup your data. As a testbed, we do not guarantee data loss.

***Scratch* directories:** Scratch directories are located at different locations on the systems. To find out more about the file layout, please see the section *s-storage*

***System software* directories:** System software directories are located at */N/soft*. System and community software are typically installed here. Table *t-storage-mountpoint* provides a summary of the various mount points.

8.2 Message Passing Interface (MPI)

The *Message Passing Interface Standard (MPI)* is the *de facto* standard communication library for almost many HPC systems, and is available in a variety of implementations. It has been created through consensus of the MPI Forum, which has dozens of participating organizations, including vendors, researchers, software library developers, and users. The goal of the Message Passing Interface is to provide a portable, efficient, and flexible standard for programs using message passing. For more information about MPI, please visit:

- http://www.mpi-forum.org/
- http://www.mcs.anl.gov/research/projects/mpi/tutorial/
- http://www.open-mpi.org/

8.2.1 MPI Libraries

Several FutureGrid systems support MPI as part of their HPC services. An up to date status about it can be retrieved via our Inca status pages.

Table 8.2: MPI versions installed on FutureGrid HPC services

System	MPI version	Compiler	Infiniband Support	Module	
Bravo	OpenMPI 1.4.2	Intel 11.1	no	openmpi	
	OpenMPI 1.4.3	gcc 4.4.6	no	openmpi/1.4.3-gnu	
	OpenMPI 1.4.3	Intel 11.1	no	openmpi/1.4.3-intel	
	OpenMPI 1.5.4	gcc 4.4.6	no	openmpi/1.5.4-[gnu	intel]
India	OpenMPI 1.4.2	Intel 11.1	yes	openmpi	
Sierra	OpenMPI 1.4.2	Intel 11.1	no	openmpi	
Xray			N/A		

Loading the OpenMPI module adds the MPI compilers to your $PATH environment variable and the OpenMPI shared library directory to your $LD_LIBRARY_PATH. This is an important step to ensure that MPI applications will compile and run successfully. In cases where the OpenMPI is compiled with the Intel compilers loading the OpenMPI module will automatically load the Intel compilers as a dependency. To load the default OpenMPI module and associated compilers, just use:

```
$ module load openmpi
```

8.2.2 Compiling MPI Applications

To compile MPI applications, users can simply use the available mpi compile commands:

mpicc: To compile C programs with the CC/icc/gcc compilers

mpicxx: To compile c++ programs with CXX/icpc/g++ compilers

mpif90: To compile programs with F90/F77/FC/ifort/gfortran

To see in detail what these commands do you can add a *-show* as an option. Thus the following commands:

```
$ mpicc -show
$ mpicxx -show
$ mpif90 -show
```

will show you the detail of each of them. The resulting output can be used as a template to adapt compile flags in case the default settings are not suitable for you.

Assuming you have loaded the OpenMPI module into your environment, you can compile a simple MPI application easily by executing:

```
$ mpicc -o ring ring.c
```

Users MUST NOT run jobs on the login or headnodes. These nodes are reserved for editing and compiling programs. Furthermore running your commands on such nodes will not provide any useful information as you actually do not use the standard cluster node.

8.2.3 Batch Jobs

Once your MPI application is compiled, you run it on the compute nodes of a cluster via a batch processing. With the help of a batch processing services a job is run on the cluster without the users intervention via a job queue. The user does not have to worry much about the internal details of the job queue, but must provide the scheduler with some guidance about the job so it can be efficiently scheduled on the system.

To run jobs on resources with the HPC services, users must first activate their environment to use the job scheduler:

```
$ module load torque
```

A complete manual for the torque scheduler can be found in the Torque manual .

Next we need to create a script so that we can run the program on the cluster. We will be using our simple ring example to illustrate some of the parameters you need to adjust. Please save the following content to a file called ring.pbs.:

```
#! /bin/bash

# OPTIONS FOR THE SCRIPT
#PBS -M username@example.com
#PBS -N ring_test
#PBS -o ring_$PBS_JOBID.out
#PBS -e ring_$PBS_JOBID.err
#PBS -q batch
#PBS -l nodes=4:ppn=8
#PBS -l walltime=00:20:00


# make sure MPI is in the environment
module load openmpi

# launch the parallel application with the correct number of process
# Typical usage: mpirun -np <number of processes> <executable> <arguments>
mpirun -np 32 ring -t 1000

echo "Nodes allocated to this job: "
cat $PBS_NODEFILE
```

This file can be used to submit a job to the queueing system by calling the command:

```
qsub ring.pbs
```

In the job script, lines that begin with **#PBS** are directives to the job scheduler. You can disable any of these lines by adding an extra **#** character at the beginning of the line, as *##* is interpreted to be a comment. Common options include:

- -M: specify a mail address that is notified upon completion
- -N: To specify a job name
- -o: The name of the file to write stdout to
- -e: The name of the file to write stderr to
- -q: The queue to submit the job to
- -l: Resources specifications to execute the job

The first parameters are rather obvious, so let us focus on the *-q* option. Each batch service is configured with a number of queues that are targeting different classes of jobs to schedule them more efficiently. These queues can be switch on or off, be modified or new queues can be added to the system. It is useful to get a list of available queues on the system of where you would like to submit your jobs. You can also inspect which would be the most suitable queue to use for your purpose with the qstat command on the appropriate login node:

```
$ qstat -q
```

Currently we have the following queues:

HPC Job Queue Information:

Resource	Queue name	Default Wallclock Limit	Max Wallclock Limit	NOTES
india	batch	4 hours	24 hours	
	long	8 hours	168 hours	
	scalemp	8 hours	168 hours	restricted acce
sierra	batch	4 hours	24 hours	
	long	8 hours	168 hours	

Next we focus on the -l option that specifies the resources. The term:

```
nodes=4
```

means that we specify 4 servers on which we execute the job. The term:

```
ppn=8
```

means that we use 8 virtual processors per node, where a virtual processor is typically executed on a core of the server. Thus it is advisable not to exceed the number of cores per server. For some programs choosing the best performing number of servers and cores may be dependent on factors such as memory needs, IO access and other resource bounded properties. You may have to experiment with the parameters. To identify the number of servers and cores available please see Tables *Table: Compute Resources* and *Table: Compute Resource Details*. For example, India, and Sierra have 8 cores per node, thus 4 servers would provide you access to 32 processing units.

Often you may just want to have the stdout and stderr in one file, then you simply can replace the line with -e in it with:

```
#PBS -j oe
```

which simply means that you *join* stdout and stderr. Here j stands for join, o for stdout and e for stderr. In case you would like to have an e-mail sent to you based on the status of the job, you can add:

```
#PBS -m ae
```

to your script. It will send you a mail when the job aborts (indicated by a), or when the job ends (indicated by e).

8.3 Job Management

A list of all available scheduler commands is available from the Torque manual page. We describe next the use of some typical interactions to manage your jobs in the batch queue.

8.3.1 Job Submission

Once you have created a submission script, you can then use the qsub command to submit this job to be executed on the compute nodes:

```
$ qsub ring.pbs
20311.i136
```

The qsub command outputs either a job identifier or an error message describing why the scheduler would not accept your job. Alternatively, you can also use the msub command, which is very similar to the qsub command. For differences we ask you to consult the manual pages.

8.3.2 Job Deletion

Sometimes you may want to delete a job from the queue, which can be easily done with the qdel command, followed by the id:

```
$ qdel 20311
```

8.3.3 Job Monitoring

If your job is submitted successfully, you can track its execution using the qstat or showq commands. Both commands will show you the state of the jobs submitted to the scheduler. The difference is mostly in their output format.

showq: Divides the output into three sections: active jobs, eligible jobs, and blocked jobs:

```
$ showq
active jobs
------------------------
JOBID    USERNAME       STATE PROCS    REMAINING            STARTTIME
20311   yourusername       Running     16        3:59:59 Tue Aug 17 09:02:40
1 active job 16 of 264 processors in use by local jobs (6.06%)
                  2 of 33 nodes active (6.06%) eligible jobs
----------------------
JOBID    USERNAME       STATE PROCS    REMAINING            STARTTIME
0 eligible jobs blocked jobs
-----------------------
JOBID    USERNAME       STATE PROCS    REMAINING            STARTTIME
0 blocked jobs
Total job: 1
```

Legend:

Active jobs: are jobs that are currently running on resources.

Eligible jobs: are jobs that are waiting for nodes to become available before they can run. As a general rule, jobs are listed in the order that they will be scheduled, but scheduling algorithms may change the order over time.

Blocked jobs: are jobs that the scheduler cannot run for some reason. Usually a job becomes blocked because it is requesting something that is impossible, such as more nodes than those which currently exist, or more processors per node than are installed.

qstat: provides a single table view, where the status of each job is added via a status column called S:

```
$ qstat
Job id                             Name                User          Time Use S Queue
------------------------- --------------------- -------------------- -------- - -----
1981.i136                       sub19327.sub      inca                00:00:00 C batch
20311.i136                      testjob           yourusername               0 R batch
```

Legend:

Job id: is the identifier assigned to your job.

Name: is the name that you assigned to your job.

User: is the username of the person who submitted the job.

Time: is the amount of time the job has been running.

S: shows the job state. Common job states are R for a running job, Q for a job that is queued and waiting to run, C for a job that has completed, and H for a job that is being held.

Queue: is the name of the job queue where your job will run.

If you are interested in only your job use grep:

```
$ qstat | grep 20311
```

8.3.4 Job Output

If you gave your job a name with the **#PBS -N <jobname>** directive in your job script or by specifying the job name on the command line, your job output will be available in a file named **jobname.o######**, where the **######** is the job number assigned by the job manager. You can type **ls jobname.o*** to see all output files from the same job name.

If you explicitly name an output file with the **#PBS -o <outfile>** directive in your job script or by specifying the output file on the command line, your output will be in the file you specified. If you run the job again, the output file will be overwritten.

If you don't specify any output file, your job output will have the same name as your job script, and will be numbered in the same manner as if you had specified a job name (**jobname.o######**).

8.4 Xray HPC Services

To log into the login node of xray please use the command:

```
ssh portalname@xray.futuregrid.org
```

Extensive documentation about the user environment of the Cray can be found at

- Cray XTTM Programming Environment User's Guide

For MPI jobs, use cc (pgcc). For best performance, add the xtpe-barcelona module:

```
% module add xtpe-module
```

Currently there is only one queue (batch) available to users on the Cray, and all jobs are automatically routed to that queue. You can use the same commands as introduced in the previous sections. Thus, to list the queues please use:

```
qstat -Q
```

To obtain details of running jobs and available processors, use the showq command:

```
/opt/moab/default/bin/showq
```

8.4.1 Submitting a Job on xray

To execute an MPI program on xray we use a special program called aprun in the submit script. Additionally we have some special resource specifications that we can pass along, such as mppwidth and mppnppn. An example is the following program that will use 16 processors on 2 nodes:

```
$ cat job.pbs
```

```
#! /bin/sh

#PBS -l mppwidth=16
#PBS -l mppnppn=8
#PBS -N hpcc-16
#PBS -j oe
#PBS -l walltime=7:00:00

#cd to directory where job was submitted from
cd $PBS_O_WORKDIR
export MPICH_FAST_MEMCPY=1
export MPICH_PTL_MATCH_OFF=1
aprun -n 16 -N 8 -ss -cc cpu hpcc
```

```
$ qsub job.pbs
```

The XT5m is a 2D mesh of nodes. Each node has two sockets, and each socket has four cores. The batch scheduler interfaces with a Cray resource scheduler called APLS. When you submit a job, the batch scheduler talks to ALPS to find out what resources are available, and ALPS then makes the reservation.

Currently ALPS is a "gang scheduler" and only allows one "job" per node. If a user submits a job in the format aprun -n 1 a.out , ALPS will put that job on one core of one node and leave the other seven cores empty. When the next job comes in, either from the same user or a different one, it will schedule that job to the next node.

If the user submits a job with aprun -n 10 a.out , then the scheduler will put the first eight tasks on the first node and the next two tasks on the second node, again leaving six empty cores on the second node. The user can modify the placement with -N , -S , and -cc .

A user might also run a single job with multiple treads, as with OpenMPI. If a user runs this job aprun -n 1 -d 8 a.out , the job will be scheduled to one node and have eight threads running, one on each core.

You can run multiple, different binaries at the same time on the same node, but only from one submission. Submitting a script like this will not work:

```
OMP_NUM_THREADS=1 aprun -n 1 -d 1 -cc 0 ./my-binary
OMP_NUM_THREADS=1 aprun -n 1 -d 1 -cc 1 ./my-binary
OMP_NUM_THREADS=1 aprun -n 1 -d 1 -cc 2 ./my-binary
OMP_NUM_THREADS=1 aprun -n 1 -d 1 -cc 3 ./my-binary
OMP_NUM_THREADS=1 aprun -n 1 -d 1 -cc 4 ./my-binary
OMP_NUM_THREADS=1 aprun -n 1 -d 1 -cc 5 ./my-binary
OMP_NUM_THREADS=1 aprun -n 1 -d 1 -cc 6 ./my-binary
OMP_NUM_THREADS=1 aprun -n 1 -d 1 -cc 7 ./my-binary
```

This will run a job on each core, but not at the same time. To run all jobs at the same time, you need to first add all the binaries within one aprun command:

```
$ cat run-all.pbs
./my-binary1
./my-binary2
./my-binary3
./my-binary4
./my-binary5
./my-binary6
./my-binary7
./my-binary8
$ aprun -n 1 run.pbs
```

Alternatively, use the command aprun -n 1 -d 8 run.pbs. To run multiple serial jobs, you must build a batch script to divide the number of jobs into groups of eight, and the

8.5 Interactive Queues

The current queing system contains the ability to run interactive queues. This is quite usefule, if you need to debug programs interactively that you will run than in a bacth queue. To use this feature we provide here a simple exaple on how to use a node on bravo.

Start an interactive shell with X11 forwarding on bravo you have to first login into india as the bravo queues are currently controlled on india:

```
ssh -X india
```

Than you need to start an interactive node:

```
qsub -I -q bravo -X
```

As xterm is currently not installed on bravo, you can test the X11 forwarding with:

```
firefox
```

The best way is to find your own resources and let us know which we shoudl add.

CHAPTER

NINE

HARDWARE

9.1 Hardware at Indiana University

In this section we describe the hardware that is available at Indiana University and allows you as part of course work or joint projects to gain easily access to them.

9.1.1 Compute Resources

The tables *Table: Compute Resources* and *Table: Compute Resource Details* show an overview of some imporatnt information about these clusters.

Table 9.1: Table: Compute Resources

Name	System Type	# Nodes	# CPUS	# Cores	TFLOPS	RAM (GB)	Storage (TB)	Site
india	IBM iDataplex	128	256	1024	11	3072	335	IU
xray	Cray XT5m	1	166	664	6	1328	5.4	IU
bravo	HP Proliant	16	32	128	1.7	3072	128	IU
delta	SuperMicro GPU Cluster	16	32	192		1333	144	IU
echo	SuperMicro ScaleMP Cluster	16	32	192	2	6144	192	IU

Table 9.2: Table: Compute Resource Details

Name	India	Echo	Bravo	Delta	Xray
Organization	Indiana University		Indiana University	Indiana University	Indiana University
Machine Type	Cluster	Cluster SclaeMP	Cluster	Cluster	Cluster
System Type	IBM iDataPlex dx 360 M2	Super-Micro	HP Proliant		Cray XT5m
CPU type	Intel Xeon X5550	Intel Xeon E5-2640	Intel Xeon E5620	Intel Xeon 5660	AMD Opteron 2378
Host Name	india	echo	bravo	delta	xray
CPU Speed	2.66GHz	2.50GHz	2.40GHz	2.80 GHz	2.4GHz
Number of CPUs	256		32	32	168
Number of nodes	128	12	16	16	1
RAM	24 GB DDR3 1333Mhz		192 GB DDR3 1333Mhz	192 GB DDR3 1333 Mhz	8 GB DDR2-800
Total RAM (GB)	3072		3072	3072	1344
Number of cores	1024	144	128		672
Operating System	Linux		Linux	Linux	Linux
Tflops	11		1.7		6
Disk Size (TB)	335	2.8		15	335
Hard Drives	3000 GB Internal 7200 RPM SATA Drive		6x2TB Internal 7200 RPM SATA Drive	Seagate Constellation 7.2 K RPM 64 MB Cache SATA 92GB	6 TB Internal Lustre Storage
Primary storage shared by all nodes	NFS		NFS	NFS	NFS
Storage details				RAID 9260-4i 1pt SAS2 512 MB SGL	
Connection configuration	Mellanox 4x DDR InfiniBand adapters		Mellanox 4x DDR InfiniBand adapters		Cray SeaStar Interconnect
Primary storage shared by all nodes					
CPUs (cores) per node				2	
Cores per CPU				6	
Total number of GPU cores				192	
GPU type				nVIDIA Tesla C2070	
Cores per GPU				448	
GPUs per node				2	
Batch system				Torque	

9.1.2 Networks

Resource Name	Network Devices		
IU Cray	Cray 2D Torus SeaStar		
IU iDataPlex	DDR IB	QLogic switch with Mellanox ConnectX adapters	Blade Network Technologies & Force10 Ethernet switches

Below is further information about networking:

Resource	Network Switch	Link
FutureGrid Core	Juniper EX8200	
India	Force10 C-150	Juniper/Dell EX series Force 10
Bravo	Force10 S60	force10-s60
Delta	Force10 S60	
Echo	Force10 S60	
Xray	Force10, C-150	Force10-c150
Node NICs	built-in (IBM iDataPlex DX360 M2) dual Intel 82575EB Gigabit Network Connection 10Gbps, Myricom Myri-10G Dual-Protocol NIC (available on login node)	

Todo

network swithes inside india not corerct old switch is IBM rack switches (formerly BNT) but that switch was replaced

CHAPTER

TEN

DEVOPS

10.1 Historical and Functionality Perspective

- Make (1977)
- GNU autotools
- rpm/yum
- DevOps

    *cfengine * puppet * chef

10.2 Makefile

To manage large software programs it is often necessary to recompile them and to just focus on the peaces of code that are new. Thus software developers have early on focussed on building software components and libraries that can be separately compiled and integrated in the overall program executable through for example libraries.

Unfortunately, this comes also at a price that the management of such "assembled" software can be quite complex and involves the compilation of code ina particular order or the creation of artifacts during compile time.

To coordinate such execution *Makefiles* have been popular as they provide the ability to integrate a simple structure in the compile process, while detecting changes to source code that itself invoke actions as part of the make process.

The coordination of the process is specified in a *makefile* that contains targets that get invoked based on conditions such as that a source file has changed. The target has a body attached with it that will be executed when the precondition to the target is fulfilled.

Through a series of targets relatively sophisticated compile workflows can be specified and often the developer has to just call the command:

```
make
```

In addition make has also the ability to execute the programs in parallel while using the option *-j*. For large programs this can provide a significant speedup during the program assembly.

A sample Makefile looks as follows:

```
all: ring

ring: main.o message.o ring.o
        g++ main.o message.o ring.o -o ring

main.o: main.cpp
```

```
        g++ -c main.cpp

message.o: message.cpp
        g++ -c message.cpp

ring.o: ring.cpp
        g++ -c ring.cpp

clean:
        rm -rf *o ring
```

This example makefile creates a program with the name ring while integrating the *ring.c* and the *message.c* code into a single executable called *ring*.

On Unix systems one can find out more about make when saying:

```
man make
```

Much more detailed information is provided at

- http://www.gnu.org/software/make/manual/make.html

10.2.1 Practical Other Applications of Make

If you look at the process on how we create the documentation of this Web page, you will also see a number of Makefiles. Although we do not create c, we do create a web pages based on Sphinx translating rst files to html pages. This indicates the versatility of make.

10.2.2 Exercises

1. Write a c++ program that prints "Hello Cloud". Use a library cloud.o and create the program hello with a Makefile

10.3 Shell Scripts

Often shells are used to interact with computers n the command line. they provide convenient access to a number of commands and functionality that makes interactive experiments through a series of command possible. Shells are interpreters that provide a minimal environment to allow easy scripting of commands as part of schell scripts. Features that are of the used are aliasses, command definition, function, and clearly batch operations by listing commands sequentially in a shell script. Programming language features such as loops and conditions are also provided.

As shell scripts are executed as part of the OS no further install is necessary to run them.

However in contrast to modern programming languages and interpreters some functionality is missing. Also large amount of shell scripts become quite difficult to maintain due to the lack of more modern programming language features.

Therefor it is today common to use perl and more important;y python for the development of large scale scripts.

However, a large number of "tricks" and existing scripts makes shell scripts still a viable option for many developers. In addition it has the advantage that it will be immediately available after the install of the OS, thus it is of great help in case of managing distributed machines.

Variants of shells exist that can execute the same command in parallel on multiple distributed machines which comes in handy for cluster management.

10.3.1 Exercises:

1. Write a shell script that prints you username and lists the files in your home directory.
2. Write a shell script that converts all jpg files to png. (If you copy the example form http://en.wikipedia.org/wiki/Shell_script please make sure to walk through the example and understand in detail the syntax and the meaning of the program).
3. Search in our page for the term "pdsh"

10.4 Configure

Due to the many different computer systems it is important o note that libraries compiled on one system may not exist on another system or that different files need to be involved as part of the make process on the various systems. For example a command may not exist with the same name on the different computers.

To assit in this task developers use configure scripts that adapt to the underlaying enviorinment by abstarcting system related dependencies and fulfilling them with concrete implementations. As part of this process the Makefile will typically created and also a target install will be defined in the Makefile tu guide the insatlation process. Hence the process usually looks like:

```
./confugure
make
make install
```

A good image showcaseing all component involved in the configure process is available in Wikipedia. It depicts how a compatible Makefile can be generated. Developers will work on creating a Makefile.in with the available tools, that than will be used as the input to configure to create the actual makefile.

10.5 Excersises

1. What is autoconf and auomake?
2. Showcase the development of a Makefile.in for our Hello cloud example.
3. Use configure to deploy it

10.6 Package Managers

RPM

- http://en.wikipedia.org/wiki/RPM_Package_Manager

YUM

- http://en.wikipedia.org/wiki/Yellowdog_Updater,_Modified

apt-get

- https://help.ubuntu.com/community/AptGet/Howto

CHAPTER

ELEVEN

RESTRUCTUREDTEXT

Cheatcheat

- http://github.com/ralsina/rst-cheatsheet/raw/master/rst-cheatsheet.pdf
- http://docutils.sourceforge.net/docs/ref/rst/directives.html

Important extensions:

- http://sphinx-doc.org/ext/todo.html

Todo

add missing details about rst here.

11.1 Sections

RST allows to specify a number of sections. You can do this with the various underlines:

```
*********************
Chapter
*********************
Section
=====================
Subsection
---------------------
Subsubsection
^^^^^^^^^^^^^^^^^^^^^
Paragraph
~~~~~~~~~~~~~~~~~~~~~
```

11.2 Exceltable

we have integrated Excel table from http://pythonhosted.org//sphinxcontrib-exceltable/ intou our sphinx allowing the definition of more elaborate tables specified in excel. Howere the most convenient way may be to use list-tables. The documentation to list tables can be found at http://docutils.sourceforge.net/docs/ref/rst/directives.html#list-table

11.3 Autorun

Autorun is an extension for **Sphinx_** that can execute the code from a runblock directive and attach the output of the execution to the document.

For example:

```
.. runblock:: pycon

    >>> for i in range(3):
    ...    print i
```

Produces

```
>>> for i in range(3):
...    print i
...
0
1
2
```

Another example:

```
.. runblock:: console

    $ date
```

Produces

```
$ date
Sun Sep 14 21:43:19 EDT 2014
```

However, when it comes to excersises we do preferthe use of ipython notebooks as this allows us to present them also to users as self contained excersises.

11.4 Todo

```
.. todo:: an example
```

Todo

an example

CHAPTER

TWELVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*
- *notebooks*