StarCluster Documentation

Release 0.92

Justin Riley

October 17, 2011

CONTENTS

Contents:

CHAPTER

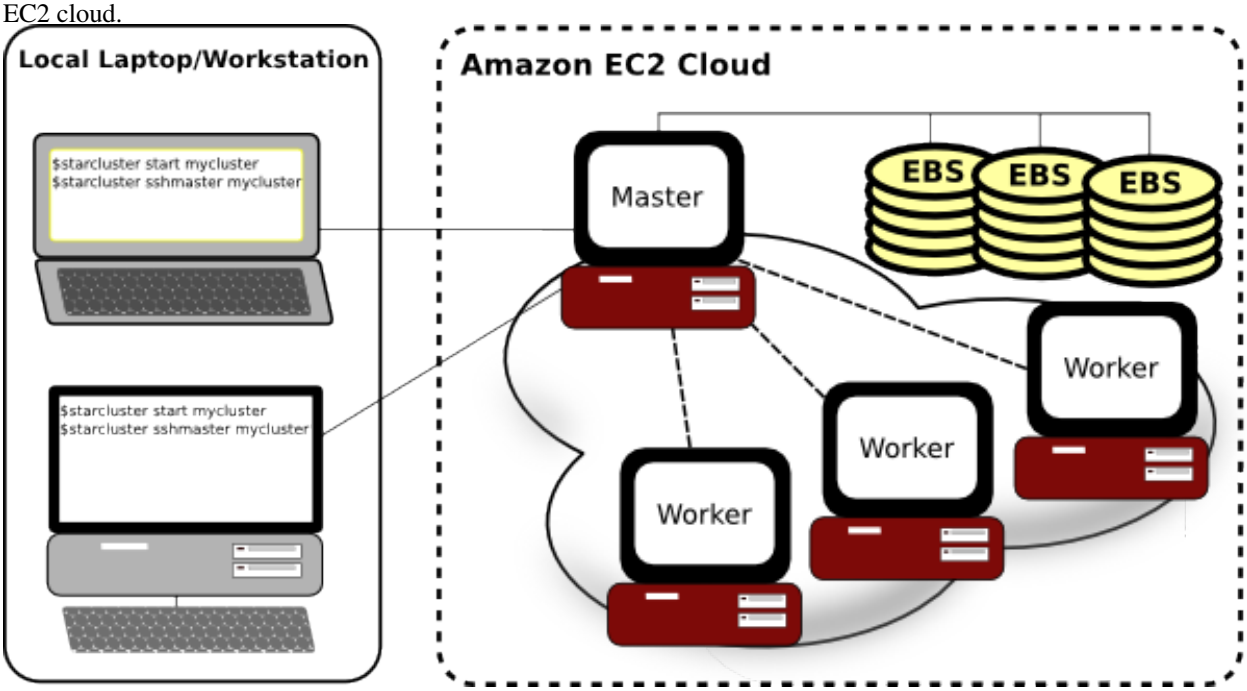
ONE

MASTER TABLE OF CONTENTS

Contents:

1.1 What is StarCluster?

StarCluster is an open source cluster-computing toolkit for Amazon's Elastic Compute Cloud (EC2). StarCluster has been designed to simplify the process of building, configuring, and managing clusters of virtual machines on Amazon's EC2 cloud.



The following sections provide a brief overview of StarCluster's feature set. For more details, please refer to the *StarCluster User Manual*.

1.1.1 Cluster Configuration

StarCluster takes care of the heavy lifting when it comes to creating and configuring a cluster on EC2. StarCluster configures the following items out-of-the-box:

Security Groups

StarCluster configures a new security group for each cluster created (e.g. *@sc-mycluster*). This allows you to control all network access to the cluster simply by applying various firewall rules to the cluster's security group. These rules can be applied using the Amazon Web Console or by StarCluster via the *permissions configuration options*

User-friendly Hostnames

StarCluster uses a simple naming convention for all of the nodes in the cluster. The head node's hostname is set to *master*. Each worker node is then labeled *node001*, *node002*, etc. StarCluster uses these user-friendly hostnames instead of the random *ec2-123-123-123.compute-aws.com* EC2 dns names making it easier for you to manage the nodes:

```
$ starcluster listclusters
StarCluster - (http://web.mit.edu/starcluster)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

-------------------------------------------
mycluster (security group: @sc-mycluster)
-------------------------------------------
Launch time: 2011-10-12 17:37:33
Uptime: 4 days, 08:01:15
Zone: us-east-1c
Keypair: myec2key
EBS volumes:
    vol-7777777 on master:/dev/sdz (status: attached)
Cluster nodes:
     master running i-99999999 ec2-123-123-123-123.compute-1.amazonaws.com
    node001 running i-88888888 ec2-123-123-123-132.compute-1.amazonaws.com
    node002 running i-77777777 ec2-123-123-123-148.compute-1.amazonaws.com
Total nodes: 3
```

User Accounts

By default StarCluster configures a non-root user account for you to use for non administrative tasks that don't require root privileges:

```
$ starcluster sshmaster -u sgeadmin mycluster
```

Password-less SSH

StarCluster configures the cluster so that ssh may be used from any node in the cluster to connect to any other node in the cluster *without using a password*. This means you can simply login to the master node of a cluster:

```
$ starcluster sshmaster mycluster
```

and connect to any of the nodes (e.g. node001, node002, etc) simply by running:

```
$ ssh node001
```

This is useful for both interactive use of the cluster as well as performing various administrative tasks across the cluster. Password-less SSH is also a strict requirement for MPI applications.

NFS Shares

By default, StarCluster configures */home* on the master node to be NFS-shared across the cluster. Any EBS volumes specified in a cluster's configuration will also be NFS-shared across the cluster.

EBS Volumes

StarCluster allows you to easily attach and NFS-share EBS volumes across the cluster for persistent storage. This can be done in a few lines in the config:

```
[volume mydata]
volume_id = vol-9999999
mount_path = /mydata

[cluster mycluster]
...
volumes = mydata
```

Scratch Space

Each node in the cluster is configured with a /scratch directory linked to its ephemeral storage (usually mounted on */mnt*). This scratch space provides a convenient location for temporary working files. As the name *ephemeral* implies, the contents of /scratch on each node will be lost when a cluster is terminated.

Queueing System

StarCluster configures the Oracle Grid Engine queueing system for distributing tasks, or *jobs*, across the cluster. Oracle Grid Engine takes care to schedule the tasks so that the entire cluster is fully utilized but not overloaded.

1.1.2 Customize Cluster Using Plugins

StarCluster also has support for plugins which allow users to further configure the cluster to their liking after StarCluster's defaults. Plugins are written in Python and use StarCluster's API to interact with the nodes. The API supports executing commands, copying files, and other OS-level operations on the nodes. Below is a simple example of a plugin that installs a Debian/Ubuntu package on all of the nodes:

```
from starcluster.clustersetup import ClusterSetup

class PackageInstaller(ClusterSetup):
    """
    Installs a Debian/Ubuntu package on all nodes in the cluster
    """
    def __init__(self, pkg_to_install):
        self.pkg_to_install = pkg_to_install

    def run(self, nodes, master, user, user_shell, volumes):
        for node in nodes:
            node.ssh.execute('apt-get -y install %s' % self.pkg_to_install)
```

For more details see *the plugin guide*.

1.1.3 StarCluster Machine Images (AMIs)

In addition to automatic cluster configuration, StarCluster also ships with it's own Amazon machine images (AMIs) that contain applications and libraries for scientific computing and software development. The AMIs currently consist of the following scientific libraries:

1. OpenMPI - Library for writing parallel applications
2. ATLAS optimized for the larger Amazon EC2 instance types
3. NumPy/SciPy compiled against the optimized ATLAS install
4. IPython - interactive parallel computing in Python

StarCluster AMIs also exist for the Cluster Compute and Cluster GPU instance types that come with the CUDA SDK as well as PyCUDA. To get a list of all of StarCluster's available AMIs use the *listpublic* command:

```
$ starcluster listpublic
```

1.1.4 Create and Manage Clusters

StarCluster allows easily creating one or more clusters of virtual machines in the cloud:

```
$ starcluster start -s 10 mycluster
```

Use the *listclusters* command to keep track of your clusters:

```
$ starcluster listclusters
```

Login to the master node of your cluster:

```
$ starcluster sshmaster mycluster
```

Add additional nodes to your cluster for more compute power:

```
$ starcluster addnode mycluster
```

Remove idle nodes from your cluster to minimize costs:

```
$ starcluster removenode mycluster node003
```

When you're done using the cluster and wish to stop paying for it:

```
$ starcluster terminate mycluster
```

1.1.5 Dynamically Resize Clusters

StarCluster also supports dynamically adding and removing nodes to and from the cluster using the Oracle Grid Engine load balancer:

```
$ starcluster loadbalance mycluster
```

The load balancer will continuously monitor the tasks in the Oracle Grid Engine queue. If the task queue becomes overloaded the load balancer will add more nodes to relieve the load. If the task queue becomes empty the load balancer will begin removing nodes from the cluster in favor of cutting costs.

1.1.6 Easily Create and Format EBS Volumes

Usually when creating a new EBS volume by hand you would need to create a new volume, launch an instance in the volume's zone, attach the volume to the instance, login to the instance, and format the volume. StarCluster does all that for you automatically in one convenient command:

```
$ starcluster createvolume 50 us-east-1c
```

1.1.7 Easily Build S3 and EBS AMIs

There are a lot of tedious steps involved when creating a new S3, or instance-store, AMI by hand. Similarly, converting an S3-based AMI to an EBS-based AMI can also be tedious and time consuming. Fortunately, StarCluster provides two commands, *s3image* and *ebsimage*, that greatly simplify the process of creating new S3 and EBS AMIs.

To create a new AMI simply launch an instance, customize it to your liking, and use either the *s3image* command:

```
$ starcluster s3image i-99999999 my-new-image my-s3-bucket
```

or *ebsimage* comand:

```
$ starcluster ebsimage i-99999999 my-new-image
```

to create a new AMI.

1.1.8 Copy Data To and From a Cluster

StarCluster allows you to easily copy data to and from a running cluster without having to look up hostnames or figure out OpenSSH's *scp* command, SSH keys, etc. To copy files from your local computer to a remote cluster:

```
$ starcluster put mycluster /local/file/or/dir /remote/path
```

To copy files from a remote cluster to your local computer:

```
$ starcluster get mycluster /remote/path /local/file/or/dir
```

The above commands will automatically handle recursion for you in the case that you're copying a directory.

1.1.9 Reduce Costs using Spot Instances

StarCluster also has support for launching and using spot instances. Using spot instances with StarCluster is as simple as specifying a spot bid to the *start* command:

```
$ starcluster start -b 0.50 mycluster
```

The above command will request spot instances with a bid of $0.50 each for each worker node in the cluster. The master node is *always* launched as a flat-rate instance for stability reasons.

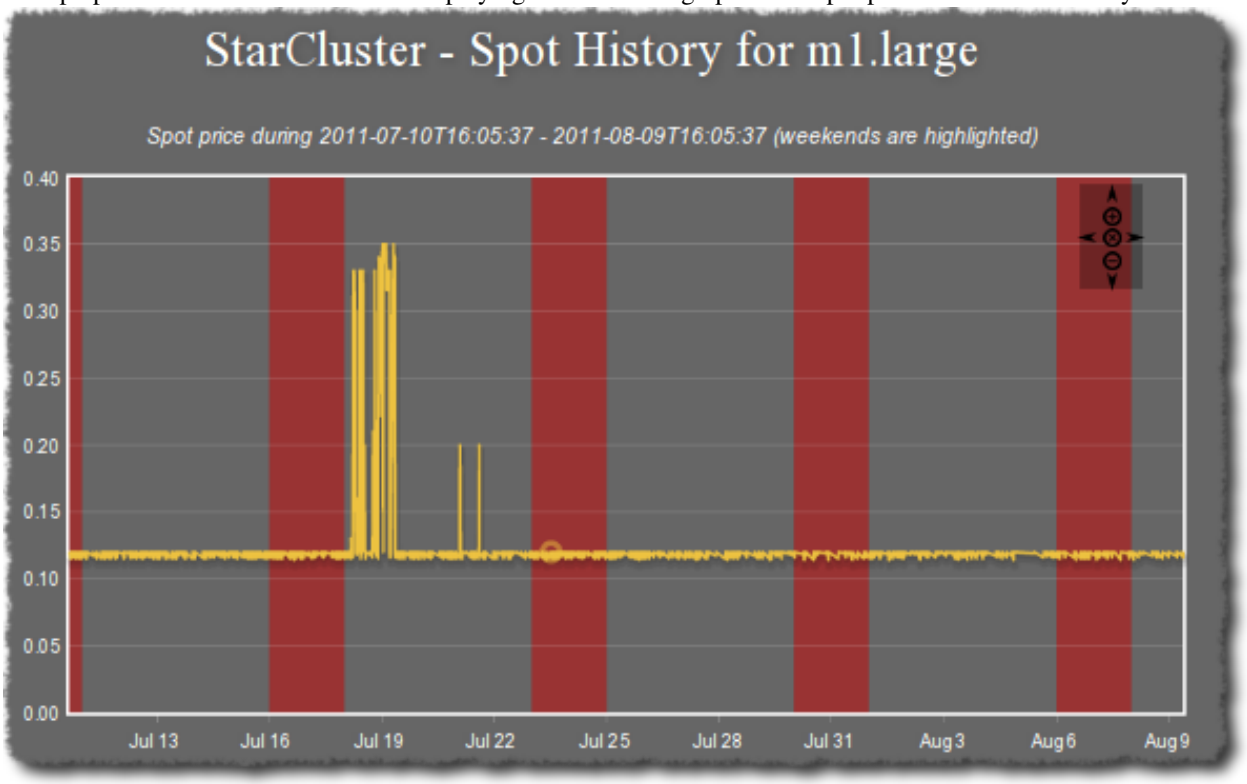
You can determine a decent spot bid to use by investigating the current, maximum, and average spot price using the *spothistory* command:

```
% starcluster spothistory -p m1.large
StarCluster - (http://web.mit.edu/starcluster) (v. 0.92)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

>>> Current price: $0.12
```

```
>>> Max price: $0.35
>>> Average price: $0.13
```

The above command shows the current spot price as well as the average and maximum spot price over the last 30 days. The -p option launches a web browser displaying an interactive graph of the spot price over the last 30 days:



1.1.10 And more...

StarCluster has a lot of features. For all the details, please see the full *StarCluster User Manual*.

1.2 Installing StarCluster

StarCluster is available via the Python Package Index (PyPI) and comes with two public Amazon EC2 AMIs (i386 and x86_64). Below are instructions for installing the latest stable release of StarCluster via PyPI (**recommended**). There are also instructions for installing the latest development version from github for advanced users.

1.2.1 Install Latest Stable Release from PyPI

To install the latest stable release of StarCluster from the PYthon Package Index (PYPI) on Linux/Mac operating systems, execute the following command in a terminal:

```
$ sudo easy_install StarCluster
(enter your root/admin password)
```

Assuming this command completes successfully you're now ready to create the configuration file.

Manually Install Latest Stable Release from PyPI

To manually install StarCluster from the PYthon Package Index (PYPI) on Linux/Mac operating systems, download StarCluster-XXX.tar.gz from http://pypi.python.org/pypi/StarCluster. Then change to the directory you downloaded StarCluster-XXX.tar.gz to and execute the following in a terminal:

```
$ tar xvzf StarCluster-XXX.tar.gz
$ cd StarCluster-XXX
$ sudo python distribute_setup.py
$ sudo python setup.py install
```

Assuming this command completes successfully you're now ready to create the configuration file.

1.2.2 Install development version from github

Warning: These methods describe installing the latest development snapshot. Although we try to keep the latest code functional, please be aware that things may break with these snapshots and that you use them at your own risk. This section is really only meant for those that are interested in contributing to or testing the latest development code.

There are two ways to install the latest development version of StarCluster from github:

Install development version using a downloaded snapshot

This method does not require any special tools other than a web browser and python and is recommended if you don't use git but would still like the latest development changes.

Download a zip or tar snapshot of the latest development code.

After downloading the code, perform the following in a terminal to install:

```
$ cd StarCluster
$ sudo python distribute_setup.py
$ sudo python setup.py install
```

Assuming this command completes successfully you're now ready to create the configuration file.

```
$ starcluster help
```

Install development version using git

This method requires that you have git installed on your machine. If you're unsure, either use the latest development snapshot as described above, or install the latest stable version from pypi.

```
$ git clone git://github.com/jtriley/StarCluster.git
$ cd StarCluster
$ sudo python distribute_setup.py
$ sudo python setup.py install
```

After this is complete, you will need to setup the configuration file.

```
$ starcluster help
```

1.3 Quick-Start

Note: These instructions are meant to get users up and running quickly without going through all of the steps in detail. For more information please refer to the full *user manual*.

Install StarCluster using easy_install:

```
$ sudo easy_install StarCluster
```

or to install StarCluster manually:

```
$ (Download StarCluster from http://web.mit.edu/starcluster)
$ tar xvzf starcluster-X.X.X.tar.gz  (where x.x.x is a version number)
$ cd starcluster-X.X.X
$ sudo python setup.py install
```

After the software has been installed, the next step is to setup the configuration file:

```
$ starcluster help
StarCluster - (http://web.mit.edu/starcluster) (v. 0.9999)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

cli.py:87 - ERROR - config file /home/user/.starcluster/config does not exist

Options:
--------
[1] Show the StarCluster config template
[2] Write config template to /home/user/.starcluster/config
[q] Quit

Please enter your selection:
```

Select the second option by typing *2* and pressing enter. This will give you a template to use to create a configuration file containing your AWS credentials, cluster settings, etc. The next step is to customize this file using your favorite text-editor:

```
$ vi ~/.starcluster/config
```

This file is commented with example "cluster templates". A cluster template defines a set of configuration settings used to start a new cluster. The example config provides a *smallcluster* template that is ready to go out-of-the-box. However, first, you must fill in your AWS credentials and keypair info:

```
[aws info]
aws_access_key_id = #your aws access key id here
aws_secret_access_key = #your secret aws access key here
aws_user_id = #your 12-digit aws user id here
```

The next step is to fill in your keypair information. If you don't already have a keypair you can create one from StarCluster using:

```
$ starcluster createkey mykey -o ~/.ssh/mykey.rsa
```

This will create a keypair called *mykey* on Amazon EC2 and save the private key to ~/.ssh/mykey.rsa. Once you have a key the next step is to fill-in your keypair info in the StarCluster config file:

```
[key key-name-here]
key_location = /path/to/your/keypair.rsa
```

For example, the section for the keypair created above using the **createkey** command would look like:

```
[key mykey]
key_location = ~/.ssh/mykey.rsa
```

After defining your keypair in the config, the next step is to update the default cluster template *smallcluster* with the name of your keypair on EC2:

```
[cluster smallcluster]
keyname = key-name-here
```

For example, the *smallcluster* template would be updated to look like:

```
[cluster smallcluster]
keyname = mykey
```

Now that the config file has been set up we're ready to start using StarCluster. Next we start a cluster named "mycluster" using the default cluster template *smallcluster* in the example config:

```
$ starcluster start mycluster
```

The *default_template* setting in the **[global]** section of the config specifies the default cluster template and is automatically set to *smallcluster* in the example config.

After the **start** command completes you should now have a working cluster. You can login to the master node as root by running:

```
$ starcluster sshmaster mycluster
```

You can also copy files to/from the cluster using the **put** and **get** commands. To copy a file or entire directory from your local computer to the cluster:

```
$ starcluster put /path/to/local/file/or/dir /remote/path/
```

To copy a file or an entire directory from the cluster to your local computer:

```
$ starcluster get /path/to/remote/file/or/dir /local/path/
```

Once you've finished using the cluster and wish to stop paying for it:

```
$ starcluster terminate mycluster
```

Have a look at the rest of StarCluster's available commands:

```
$ starcluster --help
```

1.3.1 Learn more...

Watch an ~8min screencast @ http://web.mit.edu/stardev/cluster

To learn more have a look at the rest of the documentation: http://web.mit.edu/stardev/cluster/docs

The docs explain the configuration file in detail, how to create/use EBS volumes with StarCluster, how to use the Sun Grid Engine queueing system to submit jobs on the cluster, using and creating plugins, and much more.

1.4 StarCluster User Manual

Contents:

1.4.1 StarCluster Configuration File

The StarCluster configuration file uses ini formatting. It is made up of various sections which are described here in detail. This document explains how to configure the three required sections **[aws info]**, **[keypair]**, and **[cluster]** as well as optional **[global]**, **[volume]**, and **[plugin]** sections.

Creating the configuration file

The default starcluster config lives in ~/.starcluster/config. You can either create this file manually or have starcluster create it for you with an example configuration template (recommended).

To have StarCluster generate an example configuration file (~/.starcluster/config), simply run "starcluster help" at the command-line. Provided that the configuration file does not exist, you will see the following:

```
user@localhost$ starcluster help
StarCluster - (http://web.mit.edu/starcluster)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

cli.py:475 - ERROR - Config file /home/user/.starcluster/config does not exist

Options:
--------
[1] Show the StarCluster config template
[2] Write config template to /home/user/.starcluster/config
[q] Quit

Plase enter your selection:
```

Selecting 1 will print the example configuration file template to standard output.

Selecting 2 will write the configuration file template to ~/.starcluster/config

The configuration template provided by StarCluster should be ready to go out-of-the-box after filling in your Amazon Web Services credentials and setting up a keypair. This example config provides a simple *cluster template* called 'smallcluster'. This 'smallcluster' template is also set as the default *cluster template*. The following sections explain each section of the config and their options.

Amazon Web Services Credentials

The first required section in the configuration file is **[aws info]**. This section specifies all of your AWS credentials. The following settings are required:

```
[aws info]
# replace these with your AWS keys (required)
aws_access_key_id = #your_aws_access_key_id
aws_secret_access_key = #your_secret_access_key
# these settings are optional and only used for creating new AMIs
aws_user_id= #your userid
ec2_cert = /path/to/your/ec2_cert.pem
ec2_private_key = /path/to/your/ec2_pk.pem
```

Amazon EC2 Regions

StarCluster uses the us-east-1 EC2 region by default. If you wish to use a different EC2 region you will need to specify the following additional settings in your **[aws info]** section:

```
[aws info]
....
aws_region_name = eu-west-1
aws_region_host = ec2.eu-west-1.amazonaws.com
```

Here *aws_region_name* is the name of the region you wish to use and *aws_region_host* is the region-specific EC2 endpoint host. Below is a table of EC2 region-specific endpoints:

aws_region_name	aws_region_host
us-east-1	ec2.us-east-1.amazonaws.com
us-west-1	ec2.us-west-1.amazonaws.com
eu-west-1	ec2.eu-west-1.amazonaws.com
ap-southeast-1	ec2.ap-southeast-1.amazonaws.com
ap-northeast-1	ec2.ap-northeast-1.amazonaws.com

The above table is only for convenience. In practice you should use the official list from Amazon instead.

Amazon S3 Region-Specific Endpoints

StarCluster uses s3.amazonaws.com as the S3 endpoint by default. If you'd like to switch S3 endpoints you can do so by specifying an additional *aws_s3_host* setting in your **[aws info]** section:

```
[aws info]
....
aws_region_name = us-west-1
aws_region_name = ec2.us-west-1.amazonaws.com
aws_s3_host = s3-us-west-1.amazonaws.com
```

Below is a table of S3 region-specific endpoints:

Region	aws_s3_host
us-east-1	s3.amazonaws.com
us-west-1	s3-us-west-1.amazonaws.com
eu-west-1	s3-eu-west-1.amazonaws.com
ap-southeast-1	s3-ap-southeast-1.amazonaws.com
ap-northeast-1	s3-ap-northeast-1.amazonaws.com

Note: Switching S3 endpoints is usually not necessary. From amazon: Switching to a region-specific S3 endpoint is completely optional. The main advantage of doing so is to reduce the temporary latency you might experience immediately after creating a bucket in a specific region. This temporary latency typically lasts less than one hour.

Amazon EC2 Keypairs

In addition to supplying your **[aws info]** you must also define at least one **[keypair]** section that represents one of your keypairs on Amazon EC2. Amazon EC2 keypairs are used by StarCluster to connect and configure your instances.

You should define a new **[keypair]** section for each Amazon EC2 keypair you wish to use with StarCluster. As an example, suppose we have two keypairs on Amazon EC2 that we wish to use with StarCluster named "mykeypair1" and "mykeypair2" on Amazon.

Note: If you do not know the name of your keypair(s), use StarCluster's *listkeypairs* command or the *ec2-describe-keypairs* command in the EC2 command line tools. The **[keypair]** section name *must* match the name of the keypair on Amazon EC2.

To configure StarCluster for these keypairs we define a **[keypair]** section for each of them in the configuration file:

```
[keypair mykeypair1]
# this is the path to your openssh private key for mykeypair4
key_location=/path/to/your/mykeypair1.rsa

[keypair mykeypair3]
# this is the path to your openssh private key for mykeypair2
key_location=/path/to/your/mykeypair2.rsa
```

These keypair sections can now be referenced in a *cluster templates*' **keyname** setting as we'll *show below* in an example *cluster template*.

Note: In order for StarCluster to interact with **any** instances you have on EC2, the keypair used to launch those instances **must** be defined in the config. You can check what keypairs were used to launch an instance using StarCluster's *listinstances* command or the *ec2-describe-instances* command from the ec2 command-line tools.

Defining Cluster Templates

In order to launch StarCluster(s) on Amazon EC2, you must first provide a *cluster template* that contains all of the configuration for the cluster. A *cluster template* is simply a **[cluster]** section in the config. Once a *cluster template* has been defined, you can launch multiple StarClusters from it. Below is an example *cluster template* called 'smallcluster' which defines a 2-node cluster using *m1.small* EC2 instances and the mykeypair1 keypair we defined above.

```
# Sections starting with "cluster" define your cluster templates
# The section name is the name you give to your cluster template e.g.:
[cluster smallcluster]
# change this to the name of one of the keypair sections defined above
# (see the EC2 getting started guide tutorial on using ec2-add-keypair to learn
# how to create new keypairs)
keyname = mykeypair1

# number of ec2 instances to launch
cluster_size = 2

# create the following user on the cluster
cluster_user = sgeadmin
# optionally specify shell (defaults to bash)
# options: bash, zsh, csh, ksh, tcsh
cluster_shell = bash

# AMI for master node. Defaults to NODE_IMAGE_ID if not specified
# The base i386 StarCluster AMI is ami-0330d16a
# The base x86_64 StarCluster AMI is ami-0f30d166
master_image_id = ami-0330d16a

# instance type for master node.
# defaults to NODE_INSTANCE_TYPE if not specified
master_instance_type = m1.small

# AMI for worker nodes.
```

```
# Also used for the master node if MASTER_IMAGE_ID is not specified
# The base i386 StarCluster AMI is ami-0330d16a
# The base x86_64 StarCluster AMI is ami-0f30d166
node_image_id = ami-0330d16a

# instance type for worker nodes. Also used for the master node if
# MASTER_INSTANCE_TYPE is not specified
node_instance_type = m1.small

# availability zone
availability_zone = us-east-1c
```

Defining Multiple Cluster Templates

You are not limited to defining just one *cluster template*. StarCluster allows you to define multiple independent cluster templates by simply creating a new **[cluster]** section with all of the same settings (different values of course).

However, you may find that defining new *cluster templates* is some what repetitive with respect to redefining the same settings over and over. To remedy this situation, StarCluster allows *cluster templates* to extend other *cluster templates*:

```
[cluster mediumcluster]
# Declares that this cluster uses smallcluster's settings as defaults
extends = smallcluster
# this rest of this section is identical to smallcluster except for the following settings:
keyname = mykeypair2
node_instance_type = c1.xlarge
cluster_size = 8
volumes = biodata2
```

In the example above, *mediumcluster* will use all of *smallcluster*'s settings as defaults. All other settings in the *mediumcluster* template override these defaults. For the *mediumcluster* template above, we can see that *mediumcluster* is the same as *smallcluster* except for its keyname, node_instance_type, cluster_size, and volumes settings.

Setting the Default Cluster Template

StarCluster allows you to specify a default *cluster template* to be used when using the "start" command. This is useful for users that mostly use a single *cluster template*. To define a default *cluster template*, define a **[global]** section and configure the **default_template** setting:

```
[global]
default_template = smallcluster
```

The above example sets the 'smallcluster' *cluster template* as the default.

Note: If you do not specify a default *cluster template* in the config you will have to specify one at the command line using the –cluster-template option.

Amazon EBS Volumes

Warning: Using EBS volumes with StarCluster is completely optional, however, if you do not use an EBS volume with StarCluster, any data that you wish to keep around after shutdown must be manually copied somewhere outside of the cluster (e.g. download the data locally or move it to S3 manually). This is because local instance storage on EC2 is ephemeral and does not persist after an instance has been terminated. The advantage of using EBS volumes with StarCluster is that when you shutdown a particular cluster, any data saved on an EBS volume attached to that cluster will be available the next time the volume is attached to another cluster or EC2 instance.

StarCluster has the ability to use Amazon EBS volumes to provide persistent data storage on a given cluster. If you wish to use EBS volumes with StarCluster you will need to define a **[volume]** section in the configuration file for each volume you wish to use with StarCluster and then add this **[volume]** section name to a *cluster template*'s **volumes** setting.

To configure an EBS volume for use with Starcluster, define a new **[volume]** section for each EBS volume. For example, suppose we have two volumes we'd like to use: vol-c9999999 and vol-c8888888. Below is an example configuration for these volumes:

```
[volume myvoldata1]
# this is the Amazon EBS volume id
volume_id=vol-c9999999
# the path to mount this EBS volume on
# (this path will also be nfs shared to all nodes in the cluster)
mount_path=/home

[volume myvoldata2]
volume_id=vol-c8888888
mount_path=/scratch

[volume myvoldata2-alternate]
# same volume as myvoldata2 but uses 2nd partition instead of 1st
volume_id=vol-c8888888
mount_path=/scratch2
partition=2
```

StarCluster by default attempts to mount either the entire drive or the first partition in the volume onto the master node. It is possible to use a different partition by configuring a **partition** setting in your **[volume]** section as in the *myvoldata2-alternate* example above.

After defining one or more **[volume]** sections, you then need to add them to a *cluster template* in order to use them. To do this, specify the **[volume]** section name(s) in the **volumes** setting in one or more of your *cluster templates*. For example, to use both myvoldata1 and myvoldata2 from the above example in a *cluster template* called *smallcluster*:

```
[cluster smallcluster]
#...
volumes = myvoldata1, myvoldata2
#...
```

Now any time a cluster is started using the *smallcluster* template, myvoldata1 will be mounted to /home on the master, myvoldata2 will be mounted to /scratch on the master, and both /home and /scratch will be NFS shared to the rest of the cluster nodes.

Amazon Security Group Permissions

When starting a cluster each node is added to a common security group. This security group is created by StarCluster and has a name of the form "@sc-<*cluster_tag*>" where <*cluster_tag*> is the name you provided to the "start"

command.

By default, StarCluster adds a permission to this security group that allows access to port 22 (ssh) from all IP addresses. This is needed so that StarCluster can connect to the instances and configure them properly. If you want to specify additional security group permissions to be set after starting your cluster you can do so in the config by creating one or more **[permission]** sections. These sections can then be specified in one or more cluster templates. Here's an example that opens port 80 (web server) to the world for the *smallcluster* template:

```
[permission www]
# open port 80 to the world
from_port = 80
to_port = 80

[permission ftp]
# open port 21 only to a single ip
from_port = 21
to_port = 21
cidr_ip = 66.249.90.104/32

[permission myrange]
# open all ports in the range 8000-9000 to the world
from_port = 8000
to_port = 9000

[cluster smallcluster]
#...
permissions = www, ftp, myrange
#...
```

A permission section specifies a port range to open to a given network range (cidr_ip). By default, the network range is set to 0.0.0.0/0 which represents any ip address (ie the "world"). In the above example, we created a permission section called *www* that opens port 80 to the "world" by setting the from_port and to_port both to be 80. You can restrict the ip addresses that the rule applies to by specifying the proper cidr_ip setting. In the above example, the *ftp* permission specifies that only 66.249.90.104 ip address can access port 21 on the cluster nodes.

Defining Plugins

StarCluster also has support for user contributed plugins (see *StarCluster Plugin System*). To configure a *cluster template* to use a particular plugin, we must first create a plugin section for each plugin we wish to use. For example, suppose we have two plugins myplug1 and myplug2:

```
[plugin myplug1]
setup_class = myplug1.SetupClass
myplug1_arg_one = 2

[plugin myplug2]
setup_class = myplug2.SetupClass
myplug2_arg_one = 3
```

In this example, myplug{1,2}_arg_one are arguments to the plugin's *setup_class*. The 'myplug{1,2}_arg_one' variable names were made up for this example. The names of these arguments depend on the plugin being used. Some plugins may not even have arguments.

After you've defined some **[plugin]** sections, you can reference them in a *cluster template* like so:

```
[cluster mediumcluster]
# Declares that this cluster uses smallcluster's settings as defaults
extends = smallcluster
```

```
# this rest of this section is identical to smallcluster except for the following settings:
keyname = mykeypair2
node_instance_type = c1.xlarge
cluster_size = 8
volumes = biodata2
plugins = myplug1, myplug2
```

Notice the added *plugins* setting for the 'mediumcluster' template. This setting instructs StarCluster to first run the 'myplug1' plugin and then the 'myplug2' plugin afterwards. Reversing myplug1/myplug2 in the plugins setting in the above example would reverse the order of execution.

See Also:

See also the documentation for the *StarCluster Plugin System*

1.4.2 Using EBS Volumes for Persistent Storage

StarCluster utilizes Amazon's Elastic Block Storage (EBS) volumes for persistent storage. These volumes can be anywhere from 1GB to 1TB in size. StarCluster will mount each volume specified in a cluster template to the **MOUNT_PATH** specified in the volume's configuration section on the master node. This **MOUNT_PATH** is then shared on all nodes using the network file system (NFS).

For example, suppose we have the following (abbreviated) configuration defined:

```
[vol myvol]
volume_id = vol-v99999
mount_path = /data

[cluster smallcluster]
cluster_size=3
keyname=mykey
node_instance_type=m1.small
node_image_id=ami-8cf913e5
volumes=myvol
```

In this case, whenever a cluster is launched using the *smallcluster* template StarCluster will attach the EBS volume *vol-v99999* to the *master* node on */data* and then NFS-share */data* to all the nodes in the cluster.

Create and Format a new EBS Volume

StarCluster's **createvolume** command completely automates the process of creating a new EBS volume. This includes launching a host instance in the target zone, attaching the new volume to the host, and formatting the entire volume.

> **Warning:** The **createvolume** command *only* supports formatting the *entire volume* using all of the space on the device. If you need multiple partitions you're probably better off creating multiple volumes instead. If this is not the case and you have a use-case for partitioning EBS volumes please send a note to the StarCluster mailing list (starcluster 'at' mit 'dot' edu).

To create and format a new volume simply specify a volume size (in GB) and the zone you want to create the volume in:

```
$ starcluster createvolume 20 us-east-1c
```

This command will launch a host instance in the us-east-1c availability zone, create a 20GB volume in us-east-1c, attach the new volume to the host instance, and format the entire volume.

You can also use the –bid option to request a spot instance when creating the volume host:

```
$ starcluster createvolume 20 us-east-1c --bid 0.50
```

Warning: In previous versions the **createvolume** command used to terminate the host instance after creating the volume. **The latest version does not do this by default** in order to allow multiple volumes to be created in the same zone with a *single* host instance. You can pass the *–shutdown-volume-host* option to the *createvolume* command to have StarCluster automatically shutdown the volume host after creating the new volume.

Let's look at an example:

```
$ starcluster createvolume 20 us-east-1c
StarCluster - (http://web.mit.edu/starcluster)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

>>> No keypair specified, picking one from config...
>>> Using keypair: jtriley
>>> Creating security group @sc-volumecreator...
>>> No instance in group @sc-volumecreator for zone us-east-1c,
>>> launching one now.
>>> Waiting for volume host to come up... (updating every 30s)
>>> Waiting for open spot requests to become active...
1/1 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Waiting for all nodes to be in a 'running' state...
1/1 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Waiting for SSH to come up on all nodes...
1/1 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Checking for required remote commands...
>>> Creating 1GB volume in zone us-east-1c
>>> New volume id: vol-2f3a5344
>>> Waiting for new volume to become 'available'...
>>> Attaching volume vol-2f3a5344 to instance i-fb9ceb95...
>>> Formatting volume...
mke2fs 1.41.11 (14-Mar-2010)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
65536 inodes, 262144 blocks
13107 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=268435456
8 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376

Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 30 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
>>> Leaving volume vol-2f3a5344 attached to instance i-fb9ceb95
>>> Not terminating host instance i-fb9ceb95
```

```
*** WARNING - There are still volume hosts running: i-fb9ceb95
*** WARNING - Run 'starcluster terminate volumecreator' to terminate
*** WARNING - *all* volume host instances once they're no longer needed
>>> Creating volume took 7.396 mins
>>> Your new 1GB volume vol-2f3a5344 has been created successfully
```

Notice the warning at the bottom of the above output. StarCluster will leave the host instance running with the new volume attached after creating and formatting the new volume. This allows multiple volumes to be created in a given availability zone without launching a new instance for each volume. To see the volume hosts simply run the *listclusters* command:

```
$ starcluster listclusters volumecreator
StarCluster - (http://web.mit.edu/starcluster)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

-----------------------------------------------
volumecreator (security group: @sc-volumecreator)
-----------------------------------------------
Launch time: 2011-06-13 13:51:25
Uptime: 00:02:09
Zone: us-east-1c
Keypair: mykey
EBS volumes: N/A
Cluster nodes:
    volhost-us-east-1c running i-fd9clb9z  (spot sir-2a8zb4lr)
Total nodes: 1
```

From the above example we see that we have a volume-host in us-east-1c called *volhost-us-east-1c*. Any volumes that were created will still be attached to the volume host until you terminate the *volumecreator* cluster. If you'd rather detach the volume after it's been successfully created use the *–detach-volume* (-d) option:

```
$ starcluster createvolume --detach-volume 20 us-east-1c
```

You can login to a volume host instance using:

```
$ starcluster sshnode volumecreator volhost-us-east-1c
```

After logging in you can inspect the volume, upload data, etc. When you're done using the volumecreator cluster don't forget to terminate it:

```
$ starcluster terminate volumecreator
```

If you'd rather avoid having to terminate the volumecreator each time you can pass the *–shutdown-volume-host (-s)* option to the *createvolume* command to have StarCluster automatically terminate the host-instance after successfully creating the new volume:

```
$ starcluster createvolume --shutdown-volume-host 20 us-east-1c
```

Managing EBS Volumes with StarCluster

In addition to creating and formatting new EBS volumes StarCluster also allows you to browse and remove your EBS volumes.

Getting Volume Status

To get a list of all your volumes as well as their current status use the **listvolumes** command:

```
$ starcluster listvolumes
```

If you'd like to see details for a single volume:

```
$ starcluster listvolumes -v vol-99999999
```

You can also filter the results by status:

```
$ starcluster listvolumes -S available
```

and also by attachment state:

```
$ starcluster listvolumes -a attached
```

Other filters are available, have a look at the help menu for more details:

```
$ starcluster listvolumes --help
```

Removing Volumes

Warning: This process cannot be reversed!

To **permanently** remove an EBS volume use the *removevolume* command:

```
$ starcluster removevolume vol-99999999
```

Resizing Volumes

After you've created and used an EBS volume over time you may find that you need to add additional disk space to the EBS volume. Normally you would need to snapshot the volume, create a new, larger, volume from the snapshot, attach the new volume to an instance, and expand the filesystem to fit the new volume. Fortunately, StarCluster's *resizevolume* command streamlines this process for you.

Note: The EBS volume must either be unpartitioned or contain only a single partition. Any other configuration will be aborted.

For example, to resize a 10GB volume, say vol-99999999, to 20GB:

```
$ starcluster resizevolume vol-99999999 20
```

The above command will create a *new*, larger, 20GB volume containing the data from the original volume vol-99999999. The new volume's filesystem will also be expanded to fit the new volume size.

Just like the *createvolume* command, the *resizevolume* command will also launch a host instance in order to attach the new volume and expand the volume's filesystem. Similarly, if you wish to shutdown the host instance automatically after the new resized volume has been created, use the *–shutdown-volume-host* option:

```
$ starcluster resizevolume --shutdown-volume-host vol-99999999 20
```

Otherwise, you will need to terminate the volume host manually after the *resizevolume* command completes.

Moving Volumes Across Availability Zones

In some cases you may need to replicate a given volume to another availability zone so that the data can be used with instances in a different data center. The *resizevolume* command supports creating a newly expanded volume within an alternate availability zone via the *-z*, or *–zone*, flag:

```
$ starcluster resizevolume -z us-east-1d vol-9999999 20
```

The above command will create a new 20GB volume in us-east-1d containing the data in vol-99999999. If you only want to move the volume data without resizing simply specify the same size as the original volume.

1.4.3 Launching a StarCluster on Amazon EC2

Use the **start** command in StarCluster to launch a new cluster on Amazon EC2. The start command takes two arguments: the cluster template and a tagname for cluster identification.

Below is an example of starting a StarCluster from the *default* cluster template defined in the config and tagged as *physicscluster*. This example will be used throughout this section.

```
$ starcluster start physicscluster # this line starts the cluster
StarCluster - (http://web.mit.edu/starcluster)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

>>> Validating cluster template settings...
>>> Cluster template settings are valid
>>> Starting cluster...
>>> Launching a 2-node cluster...
>>> Launching master node (ami: ami-8cf913e5, type: m1.small)...
>>> Creating security group @sc-physicscluster...
>>> Waiting for cluster to come up... (updating every 30s)
>>> Waiting for all nodes to be in a 'running' state...
1/1 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Waiting for SSH to come up on all nodes...
1/1 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> The master node is ec2-123-12-12-123.compute-1.amazonaws.com
>>> Setting up the cluster...
>>> Attaching volume vol-99999999 to master node on /dev/sdz ...
>>> Configuring hostnames...
1/1 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Mounting EBS volume vol-99999999 on /home...
>>> Creating cluster user: myuser (uid: 1001, gid: 1001)
1/1 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring scratch space for user: myuser
1/1 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring /etc/hosts on each node
1/1 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring NFS...
>>> Setting up NFS took 0.209 mins
>>> Configuring passwordless ssh for root
>>> Configuring passwordless ssh for myuser
>>> Installing Sun Grid Engine...
>>> Creating SGE parallel environment 'orte'
1/1 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Adding parallel environment 'orte' to queue 'all.q'
>>> Shutting down threads...
20/20 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
```

```
>>> Starting cluster took 6.922 mins

The cluster is now ready to use. To login to the master node
as root, run:

    $ starcluster sshmaster physicscluster

When you are finished using the cluster and wish to
terminate it and stop paying for service:

    $ starcluster terminate physicscluster

NOTE: Terminating an EBS cluster will destroy all EBS
volumes backing the nodes.

Alternatively, if the cluster uses EBS instances, you can
use the 'stop' command to put all nodes into a 'stopped'
state:

    $ starcluster stop physicscluster

NOTE: Any data stored in ephemeral storage (usually /mnt)
will be lost!

This will shutdown all nodes in the cluster and put them in
a 'stopped' state that preserves the EBS volumes backing the
nodes. A 'stopped' cluster may then be restarted at a later
time, without losing data on the local disks, by passing the
-x option to the 'start' command:

    $ starcluster start -x physicscluster

This will start all 'stopped' EBS instances and reconfigure
the cluster.
```

The output of the **start** command should look similar to the above if everything went successfully.

If you wish to use a different template besides the default, *largecluster* for example, the command becomes:

```
$ starcluster start -c largecluster physicscluster
```

This command will do the same thing as above only using the *largecluster* cluster template.

Managing Multiple Clusters

To list all of your StarClusters on Amazon EC2 run the following command:

```
$ starcluster listclusters
```

The output should look something like:

```
$ starcluster listclusters
StarCluster - (http://web.mit.edu/starcluster)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

-----------------------------------------------------
physicscluster (security group: @sc-physicscluster)
```

```
-----------------------------------------------
Launch time: 2010-02-19T20:55:20.000Z
Uptime: 00:29:42
Zone: us-east-1c
Keypair: gsg-keypair
EBS volumes:
    vol-c8888888 on master:/dev/sdj (status: attached)
Cluster nodes:
     master running i-99999999 ec2-123-123-123-121.compute-1.amazonaws.com
    node001 running i-88888888 ec2-123-123-123-122.compute-1.amazonaws.com
Total nodes: 2
```

This will list each StarCluster you've started by tag name.

Logging into the master node

To login to the master node as root:

```
$ starcluster sshmaster physicscluster
```

or as user sgeadmin:

```
$ starcluster sshmaster -u sgeadmin physicscluster
```

Logging into the worker nodes

To login to a worker node as root:

```
$ starcluster sshnode physicscluster node001
```

or as user sgeadmin:

```
$ starcluster sshnode -u sgeadmin physicscluster node001
```

The above commands will ssh to node001 of the *physicscluster*.

Rebooting a Cluster

Some times you might encounter an error while starting and setting up a new cluster or using an existing cluster. Rather than terminating the cluster and starting a new one to get around the errors, you can instead completely reconfigure the cluster without terminating instances and wasting instance-hours using the *restart* command:

```
$ starcluster restart physicscluster
```

This will reboot all of the instances, wait for them to come back up, and then completely reconfigure the cluster from scratch as if you had terminated and re-created the cluster.

Shutting Down a Cluster

Once you've finished using the cluster and wish to stop paying for it, simply run the **terminate** command providing the cluster tag name you gave when starting:

```
$ starcluster terminate physicscluster
```

This command will prompt for confirmation before destroying the cluster:

```
$ starcluster terminate physicscluster
StarCluster - (http://web.mit.edu/starcluster)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

Terminate cluster physicscluster (y/n)? y
>>> Shutting down i-99999999
>>> Shutting down i-88888888
>>> Removing cluster security group @sc-physicscluster
```

This will terminate all instances in the cluster tagged "physicscluster" and removes the @sc-physicscluster security group.

1.4.4 Copying Data to and from a Cluster

StarCluster now supports conveniently copying data to and from a running cluster via the new **put** and **get** commands. These commands provide the same functionality as the *scp* command from OpenSSH only without the need to specify SSH keypairs or EC2 dns names.

Enable Experimental Features

For now the **put** and **get** commands are experimental which means you will need to set *enable_experimental* to *True* in the **[global]** section of the config in order to use them:

```
[global]
enable_experimental=True
```

Below are a few examples of how to use the **put** and **get** command

---

Note: All of the examples below automatically handle recursion without requiring any extra command line options

---

Copying Data to a Cluster (put)

To copy data from your local computer to a cluster on Amazon use the **put** command. Recursion will be handled automatically if necessary. By default the **put** command will operate on the master node as the *root* user:

```
$ starcluster put mycluster /path/to/file/or/dir /path/on/remote/server
```

Copy a file or directory to the master as a normal user

By default the **put** command copies files as the root user. To copy files as a different cluster user, use the *–user* option:

```
$ starcluster put mycluster --user myuser /local/path /remote/path
```

Copy a file or directory to a cluster node

By default the **put** command copies files to the master node. To copy files to a different cluster node, use the *–node* option:

```
$ starcluster put mycluster --node node001 /local/path /remote/path
```

Copying Data from a Cluster (get)

To copy data from a cluster on Amazon to your local computer use the **get** command. Recursion will be handled automatically if necessary. By default the **get** command will operate on the master node as the *root* user:

```
$ starcluster get mycluster /path/on/remote/server /path/to/file/or/dir
```

Copy a file or directory from the master as a normal user

By default the **get** command copies files as the root user. To copy files as a different cluster user, use the *–user* option:

```
$ starcluster get mycluster --user myuser /remote/path /local/path
```

Copy a file or directory from a cluster node

By default the **get** command copies files from the master node. To copy files from a different cluster node, use the *–node* option:

```
$ starcluster get mycluster --node node001 /remote/path /local/path
```

1.4.5 Using the Cluster

After you've created a StarCluster on Amazon, it's time to login and do some real work. The sections below explain how to access the cluster, verify that everything's configured properly, and how to use OpenMPI and Sun Grid Engine on StarCluster.

For these sections we used a small two node StarCluster for demonstration. In some cases, you may need to adjust the instructions/commands for your size cluster. For all sections, we assume **cluster_user** is set to *sgeadmin*. If you've specified a different **cluster_user**, please replace sgeadmin with your **cluster_user** in the following sections.

Logging into the master node

To login to the master node as root:

```
$ starcluster sshmaster mycluster
StarCluster - (http://web.mit.edu/starcluster)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

The authenticity of host 'ec2-123-123-123-231.compute-1.amazonaws.com (123.123.123.231)' can't be est
RSA key fingerprint is 85:23:b0:7e:23:c8:d1:02:4f:ba:22:53:42:d5:e5:23.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-123-123-123-231.compute-1.amazonaws.com,123.123.123.231' (RSA) to the
Last login: Wed May 12 00:13:51 2010 from 192.168.1.1

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/\*/copyright.
```

```
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/

Created From:
Amazon EC2 Ubuntu 9.10 jaunty AMI built by Eric Hammond
http://alestic.com http://ec2ubuntu-group.notlong.com

StarCluster EC2 AMI created by Justin Riley (MIT)
url: http://web.mit.edu/stardev/cluster
email: star 'at' mit 'dot' edu
root@master:~#
```

This command is used frequently in the sections below to ensure that you're logged into the master node of a StarCluster on Amazon's EC2 as root.

Logging into a worker node

You also have the option of logging into any particular worker node as root by using the **sshnode** command. First, run "starcluster listclusters" to list the nodes:

```
$ starcluster listclusters
StarCluster - (http://web.mit.edu/starcluster)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

-----------------------------------------------
mycluster (security group: @sc-mycluster)
-----------------------------------------------
Launch time: 2010-02-19T20:55:20.000Z
Zone: us-east-1c
Keypair: gsg-keypair
EBS volumes:
    vol-c8888888 on master:/dev/sdj (status: attached)
Cluster nodes:
     master i-99999999 running ec2-123-123-123-121.compute-1.amazonaws.com
    node001 i-88888888 running ec2-123-123-123-122.compute-1.amazonaws.com
    node002 i-88888888 running ec2-123-23-23-24.compute-1.amazonaws.com
    node003 i-77777777 running ec2-123-23-23-25.compute-1.amazonaws.com
    ....
```

Then use "starcluster sshnode mycluster" to login to a node:

```
$ starcluster sshnode mycluster node001
StarCluster - (http://web.mit.edu/starcluster)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

The authenticity of host 'ec2-123-123-123-232.compute-1.amazonaws.com (123.123.123.232)' can't be est
RSA key fingerprint is 86:23:b0:7e:23:c8:d1:02:4f:ba:22:53:42:d5:e5:23.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-123-123-123-232.compute-1.amazonaws.com,123.123.123.232' (RSA) to the
Last login: Wed May 12 00:13:51 2010 from 192.168.1.1

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
```

```
individual files in /usr/share/doc/\*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/

Created From:
Amazon EC2 Ubuntu 9.04 jaunty AMI built by Eric Hammond
http://alestic.com http://ec2ubuntu-group.notlong.com

StarCluster EC2 AMI created by Justin Riley (MIT)
url: http://web.mit.edu/stardev/cluster
email: star 'at' mit 'dot' edu

0 packages can be updated.
0 updates are security updates.

root@node001:~#
```

Verify /etc/hosts

Once StarCluster is up, the /etc/hosts file should look like:

```
$ starcluster sshmaster mycluster
root@master:~# cat /etc/hosts
# Do not remove the following line or programs that require network functionality will fail
127.0.0.1 localhost.localdomain localhost
10.252.167.143 master
10.252.165.173 node001
```

As you can see, the head node is assigned an alias of 'master' and each node after that is labeled node001, node002, etc.

In this example we have two nodes so only master and node001 are in /etc/hosts.

Verify Passwordless SSH

StarCluster should have automatically setup passwordless ssh for both root and the CLUSTER_USER you specified.

To test this out, let's login to the master node and attempt to run the hostname command via SSH on node001 without a password for both root and sgeadmin (ie CLUSTER_USER):

```
$ starcluster sshmaster mycluster
root@master:~# ssh node001 hostname
node001
root@master:~# su - sgeadmin
sgeadmin@master:~# ssh node001 hostname
node001
sgeadmin@master:~# exit
root@master:~#
```

Verify /home is NFS Shared

The /home folder on all clusters launched by StarCluster should be NFS shared to each node. To check this, login to the master as root and run the mount command on each node to verify that /home is mounted from the master:

```
$ starcluster sshmaster mycluster
root@master:~# ssh node001 mount
/dev/sda1 on / type ext3 (rw)
none on /proc type proc (rw)
none on /sys type sysfs (rw)
/dev/sda2 on /mnt type ext3 (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
master:/home on /home type nfs (rw,user=root,nosuid,nodev,user,addr=10.215.42.81)
```

The last line in the output above indicates that /home is mounted from the master node over NFS. Running this for the rest of the nodes (e.g. node002, node003, etc) should produce the same output.

Ensure EBS Volumes are Mounted and NFS shared (OPTIONAL)

If you chose to use EBS for persistent storage (recommended) you should check that it is mounted and shared across the cluster via NFS at the location you specified in the config. To do this we login to the master and run a few commands to ensure everything is working properly. For this example we assume that a single 20GB volume has been attached to the cluster and that the volume has *MOUNT_PATH=/home* in the config. If you've attached multiple EBS volumes to the cluster, you should repeat these checks for each volume you specified in the config.

The first thing we want to do is to make sure the device was actually attached to the master node as a device. To check that the device is attached on the master node, we login to the master and use "fdisk -l" to look for our volume:

```
$ starcluster sshmaster mycluster

root@master:~# fdisk -l

...

Disk /dev/sdz: 21.4 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x2a2a3cscg

Device Boot Start End Blocks Id System
/dev/sdz1 1 2610 20964793+ 83 Linux
```

From the output of fdisk above we see that there is indeed a 20GB device /dev/sdz with partition /dev/sdz1 attached on the master node.

Next check the output of mount on the master node to ensure that the volume's *PARTITION* setting (which defaults to 1 if not specified) has been mounted to the volume's *MOUNT_PATH* setting specified in the config (/home for this example):

```
root@master:~# mount
...
/dev/sdz1 on /home type ext3 (rw)
...
```

From the output of mount we see that the partition /dev/sdz1 has been mounted to /home on the master node as we specified in the config.

Finally we check that the *MOUNT_PATH* specified in the config for this volume has been NFS shared to each cluster node by running mount on each node and examining the output:

```
$ starcluster sshmaster mycluster
root@master:~# ssh node001 mount
/dev/sda1 on / type ext3 (rw)
none on /proc type proc (rw)
none on /sys type sysfs (rw)
/dev/sda2 on /mnt type ext3 (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
master:/home on /home type nfs (rw,user=root,nosuid,nodev,user,addr=10.215.42.81)
root@master:~# ssh node002 mount
...
master:/home on /home type nfs (rw,user=root,nosuid,nodev,user,addr=10.215.42.81)
...
```

The last line in the output above indicates that *MOUNT_PATH* (/home for this example) is mounted on each worker node from the master node via NFS. Running this for the rest of the nodes (e.g. node002, node003, etc) should produce the same output.

Verify scratch space

Each node should be set up with approximately 140GB or more of local scratch space for writing temporary files instead of storing temporary files on NFS. The location of the scratch space is /scratch/CLUSTER_USER. So, for this example the local scratch for CLUSTER_USER=sgeadmin is /scratch/sgeadmin.

To verify this, login to the master and run "ls -l /scratch":

```
$ starcluster sshmaster mycluster
root@master:/# ls -l /scratch/
total 0
lrwxrwxrwx 1 root root 13 2009-09-09 14:34 sgeadmin -> /mnt/sgeadmin
```

From the output above we see that /scratch/sgeadmin has been symbolically linked to /mnt/sgeadmin

Next we run the df command to verify that at least ~140GB is available on /mnt (and thus /mnt/sgeadmin):

```
root@master:~# df -h
Filesystem Size Used Avail Use% Mounted on
...
/dev/sda2 147G 188M 140G 1% /mnt
...
root@master:~#
```

Compile and run a "Hello World" OpenMPI program

Below is a simple Hello World program in MPI

```
#include <stdio.h> /* printf and BUFSIZ defined there */
#include <stdlib.h> /* exit defined there */
#include <mpi.h> /* all MPI-2 functions defined there */

int main(argc, argv)
        int argc;
        char *argv[];
        {
        int rank, size, length;
        char name[BUFSIZ];

        MPI_Init(&argc, &argv);
```

```
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        MPI_Get_processor_name(name, &length);

        printf("%s: hello world from process %d of %d\n", name, rank, size);

        MPI_Finalize();

        exit(0);
}
```

Save this code to a file called helloworldmpi.c in /home/sgeadmin. You can then compile and run the code across the cluster like so:

```
$ starcluster sshmaster mycluster
root@master:~# su - sgeadmin
sgeadmin@master:~$ mpicc helloworldmpi.c -o helloworldmpi
sgeadmin@master:~$ mpirun -n 2 -host master,node001 ./helloworldmpi
master: hello world from process 0 of 2
node001: hello world from process 1 of 2
sgeadmin@master:~$
```

Obviously if you have more nodes, the -host mater,node001 list specified will need to be extended. You can also create a hostfile instead of listing each node for OpenMPI to use that looks like:

```
sgeadmin@:~$ cat /home/sgeadmin/hostfile
master
node001
```

After creating this hostfile, you can now call mpirun with less options:

```
sgeadmin@master:~$ mpirun -n 2 -hostfile /home/sgeadmin/hostfile ./helloworldmpi
master: hello world from process 0 of 2
node001: hello world from process 1 of 2
sgeadmin@master:~$
```

1.4.6 Customizing the StarCluster AMI

The StarCluster base AMIs are meant to be fairly minimal in terms of the software installed. If you'd like to customize the software installed on the AMI you can create a new AMI based on the StarCluster AMIs using the *s3image* and *ebsimage* commands.

Note: In previous versions it was strongly advised not to create new AMIs from an instance/node started by StarCluster (e.g. master, node001, etc). **This is no longer a limitation and should work fine**. If it doesn't, please report an issue on github.

Launching and Customizing an Image Host

In order to create a new AMI you must first launch an instance of an existing AMI that you wish to extend. This instance is referred to as an *image host* and is used to build a new AMI. The *image host* allows you to easily customize the software environment included in your new AMI simply by logging into the *image host* and making the necessary changes.

Launching a New Image Host

When launching a new *image host* it is recommended that you start a new cluster called *imagehost* using the following command:

```
$ starcluster start -o -s 1 -i <INSTANCE-TYPE> -n <BASE-AMI-ID> imagehost
```

Note: Replace **<INSTANCE-TYPE>** with an instance type that is compatible with your **<BASE-AMI-ID>**. If you're not creating a new Cluster/GPU Compute AMI, you can use m1.small (32bit) or m1.large (64bit) to minimize costs when creating the image. If you *are* creating a new Cluster/GPU Compute AMI the you'll need to launch the *image host* with a Cluster/GPU Compute instance type.

This command will create a single node (-s 1) cluster called *imagehost* using the AMI you wish to customize (*-n <BASE-AMI-ID>*) and a compatible instance type (*-i <INSTANCE-TYPE>*). The *-o* option tells StarCluster to only create the instance(s) and not to setup and configure the instance(s) as a cluster. This way you start with a *clean* version of the AMI you're extending.

You can also use a spot instance (via the **-b** or **–bid** flag) for the image host:

```
$ starcluster start -o -s 1 -b 0.50 -i <INSTANCE-TYPE> -n <BASE-AMI-ID>
```

If you used the *-o* option you'll need to periodically run the *listclusters* command to check whether or not the *image host* is up:

```
$ starcluster listclusters imagehost
```

Once the *image host* is up, login and customize the instance's software environment to your liking:

```
$ starcluster sshmaster imagehost
```

The above command will log you in as *root* at which point you can install new software using either *apt-get* or by manually installing the software from source. Once you've customized the *image host* to your liking you're ready to create a new AMI.

Using an Existing Instance as an Image Host

Of course you don't *have* to use the above method for creating a new AMI. You can use *any* instance on EC2 as an *image host*. The *image host* also doesn't have to be started with StarCluster. The only requirement is that you must have the keypair that was used to launch the instance defined in your StarCluster configuration file.

In previous versions it was strongly recommended *not* to use nodes launched by StarCluster as *image hosts*. **This is no longer the case and you can now safely use any node/instance started by StarCluster as an image host.**

Creating a New AMI from an Image Host

After you've finished customizing the software environment on the *image host* you're ready to create a new AMI. Before creating the image you must decide whether to create an *instance-store* (aka *S3-backed*) AMI or an EBS-backed AMI. Below are some of the advantages and disadvantages of using S3 vs EBS:

Factor	EBS backed AMI's	S3 backed AMI's
Root Storage Size	1TB	10GB
Instances can be Stopped	Yes	No
Boot Time	Faster (~1min)	Slower (~5mins)
Data Persistence	EBS volume attached as root disk (persist)	Local root disk (doesn't persist)
Charges	Volume Storage + Volume Usage + AMI Storage + Instance usage	AMI Storage + Instance usage
Customized AMI Storage Charges	Lower (charged only for the changes)	Higher (full storage charge)
Instance Usage Charge	No charge for stopped instances. Charged full instance hour for *every* transition from stopped to running state	Not Applicable

(see Amazon's summary of EBS vs S3 backed AMIs for more details)

If you're in doubt about which type of AMI to create, choose an EBS-backed AMI. This will allow you to create clusters that you can start and stop repeatedly without losing data on the root disks in between launches. Using EBS-backed AMIs also allows you to snapshot the root volume of an instance for back-up purposes and to easily expand the root disk size of the AMI without paying full storage charges. In addition, EBS-backed AMIs usually have much faster start up time given that there's no transferring of image files from S3 as is the case with S3-backed AMIs.

Creating an EBS-Backed AMI

This section assumes you want to create an *EBS-backed* AMI. See the next section if you'd prefer to create an S3-backed AMI instead. To create a new EBS-backed AMI, use the **ebsimage** command:

```
$ starcluster ebsimage i-9999999 my-new-image
```

In the above example, *i-99999999* is the instance id of the instance you wish to create a new image from. If the instance is a part of a cluster, such as *imagehost* in the sections above, you can get the instance id from the output of the *listclusters* command. Otherwise, you can get the instance id of *any* node, launched by StarCluster or not, via the *listinstances* command. The argument after the instance id is the name you wish to give to your new AMI.

After the **ebsimage** command completes successfully it will print out the new AMI id that you then can use in the *node_image_id/master_image_id* settings in your *cluster templates*.

Creating an S3-backed (instance-store) AMI

This section assumes you want to create an *S3-backed* AMI. See the previous section if you'd prefer to create an EBS AMI instead. To create a new S3-backed AMI, use the **s3image** command:

```
$ starcluster s3image i-9999999 my-new-image mybucket
```

In the above example, *i-99999999* is the instance id of the instance you wish to create a new image from. If the instance is a part of a cluster, such as *imagehost* in the sections above, you can get the instance id from the output of the *listclusters* command. The arguments after the instance id are the name you wish to give the AMI and the name of a bucket in S3 to store the new AMI's files in. The bucket will be created if it doesn't exist.

After the **s3image** command completes successfully it will print out the new AMI id that you can then use in the *node_image_id/master_image_id* settings in your *cluster templates*.

1.4.7 Listing All Public StarCluster AMIs

From time to time StarCluster may provide updated AMIs. These AMIs might simply contain new OS versions or could fix a bug experienced by StarCluster users. Either way, it's useful to be able to look at the *latest* available StarCluster AMIs.

To look at a list of all currently available public StarCluster AMIs, including 32bit and 64bit, run the **listpublic** command:

```
$ starcluster listpublic
```

At the time of writing this doc the output looks like:

```
$ starcluster listpublic
StarCluster - (http://web.mit.edu/starcluster) (v. 0.9999)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

>>> Listing all public StarCluster images...

32bit Images:
-------------
[0] ami-8cf913e5 us-east-1 starcluster-base-ubuntu-10.04-x86-rc3
[1] ami-d1c42db8 us-east-1 starcluster-base-ubuntu-9.10-x86-rc8
[2] ami-17b15e7e us-east-1 starcluster-base-ubuntu-9.10-x86-rc7
[3] ami-8f9e71e6 us-east-1 starcluster-base-ubuntu-9.04-x86

64bit Images:
--------------
[0] ami-0af31963 us-east-1 starcluster-base-ubuntu-10.04-x86_64-rc1
[1] ami-8852a0e1 us-east-1 starcluster-base-ubuntu-10.04-x86_64-hadoop
[2] ami-a5c42dcc us-east-1 starcluster-base-ubuntu-9.10-x86_64-rc4
[3] ami-2941ad40 us-east-1 starcluster-base-ubuntu-9.10-x86_64-rc3
[4] ami-a19e71c8 us-east-1 starcluster-base-ubuntu-9.04-x86_64
[5] ami-06a75a6f us-east-1 starcluster-base-centos-5.4-x86_64-ebs-hvm-gpu-hadoop-rc2 (HVM-EBS)
[6] ami-12b6477b us-east-1 starcluster-base-centos-5.4-x86_64-ebs-hvm-gpu-rc2 (HVM-EBS)

total images: 11
```

1.4.8 StarCluster Plugin System

StarCluster has support for user contributed plugins. Plugins allow developers to further configure the cluster in addition to the default cluster configuration provided by StarCluster. Plugins are used to provide custom cluster configurations for a variety of computing needs.

Creating a New Plugin

A StarCluster plugin is simply a Python class that extends **starcluster.clustersetup.ClusterSetup** and implements a *run* method. This class must live in a module that is on the PYTHONPATH. By default, StarCluster will add the ~/.starcluster/plugins directory to the PYTHONPATH automatically. The ~/.starcluster/plugins directory is not created automatically so you will need to create it if it does not exist.

Below is a very simple example of a StarCluster plugin that installs a package on each node using apt-get after the cluster has been configured:

```
from starcluster.clustersetup import ClusterSetup

class PackageInstaller(ClusterSetup):
     def __init__(self, pkg_to_install):
          self.pkg_to_install = pkg_to_install
     def run(self, nodes, master, user, user_shell, volumes):
          for node in nodes:
               node.ssh.execute('apt-get -y install %s' % self.pkg_to_install)
```

For this example we assume that this class lives in a module file called **ubuntu.py** and that this file lives in the ~/.starcluster/plugins directory.

This is a very simple example that simply demonstrates how to execute a command on each node in the cluster. For a more sophisticated example, have a look at StarCluster's default setup class **starcluster.clustersetup.DefaultClusterSetup**. This is the class used to perform StarCluster's default setup routines. The DefaultClusterSetup class should provide you with a more complete example of the plugin API and the types of things you can do with the arguments passed to a plugin's *run* method.

Using the Logging System

When writing plugins it's better to use StarCluster's logging system rather than print statements in your code. This is because the logging system handles formatting messages and writing them to the StarCluster debug file. Here's a modified version of the *PackageInstaller* plugin above that uses the logging system:

```
from starcluster.clustersetup import ClusterSetup
from starcluster.logger import log

class PackageInstaller(ClusterSetup):
     def __init__(self, pkg_to_install):
          self.pkg_to_install = pkg_to_install
          log.debug('pkg_to_install = %s' % pkg_to_install)
     def run(self, nodes, master, user, user_shell, volumes):
          for node in nodes:
               log.info("Installing %s on %s" % (self.pkg_to_install, node.alias))
               node.ssh.execute('apt-get -y install %s' % self.pkg_to_install)
```

The first thing you'll notice is that we've added an additional import statement to the code. This line imports the log object that you'll use to log messages. In the plugin's constructor we've added a log.debug() call that shows the current value of the pkg_to_install variable. All messages logged with the log.debug() method will always be printed to the debug file, however, these messages will only be printed to the screen if the user passes the –debug flag to the starcluster command.

In the plugin's *run* method, we've added a log.info() call to notify the user that the package they specified in the config is being installed on a particular node. All messages logged with the log.info() method will always be printed to the screen and also go into the debug file. In addition to log.info() and log.debug() there are also log.warn(), log.critical(), log.fatal(), and log.error() methods for logging messages of varying severity.

Adding Your Plugin to the Config

To use a plugin we must first add it to the config and then add the plugin's config to a *cluster template*. Below is an example config for the example plugin above:

```
[plugin pkginstaller]
setup_class = ubuntu.PackageInstaller
pkg_to_install = htop
```

In this example, pkg_to_install is an argument to the plugin's constructor (ie __init__). A plugin can, of course, define multiple construtor arguments and you can configure these arguments in the config similar to *pkg_to_install* in the above example.

After you've defined a **[plugin]** section, you can now use this plugin in a *cluster template* by configuring its **plugins** setting:

```
[cluster smallcluster]
....
plugins = pkginstaller
```

This setting instructs StarCluster to run the *pkginstaller* plugin after StarCluster's default setup routines. If you want to use more than one plugin in a template you can do so by providing a list of plugins:

```
[cluster smallcluster]
....
plugins = pkginstaller, myplugin
```

In the example above, starcluster would first run the *pkginstaller* plugin and then the *myplugin* plugin afterwards. In short, order matters when defining plugins to use in a *cluster template*.

Using the Development Shell

To launch StarCluster's development shell, use the *shell* command:

```
$ starcluster shell
StarCluster - (http://web.mit.edu/starcluster) (v. 0.9999)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

>>> Importing module config
>>> Importing module plugins
>>> Importing module cli
>>> Importing module awsutils
>>> Importing module ssh
>>> Importing module utils
>>> Importing module static
>>> Importing module exception
>>> Importing module cluster
>>> Importing module node
>>> Importing module clustersetup
>>> Importing module image
>>> Importing module volume
>>> Importing module tests
>>> Importing module templates
>>> Importing module optcomplete
>>> Importing module boto
>>> Importing module paramiko

[~]|1>
```

This launches you into an IPython shell with all of the StarCluster modules automatically loaded. You'll also notice that you have the following variables available to you automagically:

1. **cm** - manager object for clusters (*starcluster.cluster.ClusterManager*)
2. **cfg** - object for retrieving values from the config file (*starcluster.config.StarClusterConfig*)
3. **ec2** - object for interacting with EC2 (*starcluster.awsutils.EasyEC2*)

4. **s3** - object for interacting with S3 (*starcluster.awsutils.EasyS3*)

Plugin Development Workflow

The process of developing and testing a plugin generally goes something like this:

1. Start a small test cluster (2-3 nodes):

```
$ starcluster start testcluster -s 2
```

2. Install and configure the additional software/settings by hand and note the steps involved:

```
$ starcluster sshmaster testcluster
root@master $ apt-get install myapp
...
```

3. Write a first draft of your plugin that attempts to do these steps programmatically
4. Add your plugin to the StarCluster configuration file
5. Launch the development shell and test your plugin on your small test cluster:

```
$ starcluster shell
[~]|1> cm.run_plugin('myplugin', 'testcluster')
```

6. Fix any coding errors in order to get the plugin to run from start to finish using the run_plugin() method
7. Login to the master node and verify that the plugin was successful:

```
$ starcluster sshmaster testcluster
```

1.4.9 Adding and Removing Nodes from StarCluster

StarCluster has support for manually shrinking and expanding the size of your cluster based on your resource needs. For example, you might start out with 10-nodes and realize that you only really need 5 or the reverse case where you start 5 nodes and find out you need 10. In these cases you can use StarCluster's *addnode* and *removenode* commands to scale the size of your cluster to your needs.

Note: The examples below assume we have a 1-node cluster running called *mycluster*.

Adding Nodes

Adding nodes is done using the *addnode* command. This command takes a *cluster tag* as an argument and will automatically add a new node to the cluster:

```
$ starcluster addnode mycluster
StarCluster - (http://web.mit.edu/starcluster) (v. 0.9999)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

>>> Launching node(s): node001
>>> Waiting for node(s) to come up... (updating every 30s)
>>> Waiting for open spot requests to become active...
2/2 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Waiting for all nodes to be in a 'running' state...
2/2 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
```

```
>>> Waiting for SSH to come up on all nodes...
2/2 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring hostnames...
1/1 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring /etc/hosts on each node
2/2 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring NFS...
>>> Mounting shares for node node001
1/1 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
1/1 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring scratch space for user: myuser
1/1 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring passwordless ssh for root
>>> Using existing key: /root/.ssh/id_rsa
>>> Configuring passwordless ssh for myuser
>>> Using existing key: /home/myuser/.ssh/id_rsa
>>> Adding node001 to SGE
```

If the –alias option is not specified StarCluster will apply the next available *nodeXXX* alias based on the current size of the cluster. In the above example only a *master* node existed so StarCluster automatically added the second node as *node001*. Similarly we had a *master* and *node001* then running addnode would name the new node *node002*.

You can also manually specify an alias for the new node using the –alias (-a) option:

```
$ starcluster addnode -a mynewnode mycluster
```

Running listclusters will then show your alias for the node in its output:

```
$ starcluster listclusters mycluster
```

And you can login directly to the new node by alias:

```
$ starcluster sshnode mycluster mynewnode
```

Removing Nodes

StarCluster also has the ability to remove an existing cluster node's via the *removenode* command. This command takes at least two arguments: the *cluster tag* representing the cluster you want to remove nodes from and a node *alias*:

```
% starcluster removenode mycluster node001
StarCluster - (http://web.mit.edu/starcluster) (v. 0.9999)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

>>> Removing node node001 (i-8bec7ce5)...
>>> Removing node001 from SGE
>>> Removing node001 from known_hosts files
>>> Removing node001 from /etc/hosts
>>> Removing node001 from NFS
>>> Cancelling spot request sir-3567ba14
>>> Terminating node: node001 (i-8bec7ce5)
```

The above command takes care to properly remove the node from the cluster by removing the node from the Sun Grid Engine queuing system, each node's ssh known_hosts file, /etc/hosts file, and from all NFS shares. Afterwards the node is terminated. If the node is a spot instance, as it is in the above example, the spot instance request will also be cancelled.

1.4.10 StarCluster Elastic Load Balancer

StarCluster's load balancer grows and shrinks an Oracle Grid Engine cluster according to the length of the cluster's job queue. When the cluster is heavily loaded and processing a long job queue, the load balancer can gradually add more nodes, up to the specified max_nodes, to distribute the work and improve throughput. When the queue becomes empty, the load balancer can remove idle nodes in order to save money. The cluster will shrink down to a single node, the master, terminating all of the other nodes as they become idle.

Goals

- To increase the size of the cluster to some user-defined maximum number of nodes when there is a large queue of waiting jobs
- To decrease the size of the cluster to a single node or some minimum number of nodes when there are no jobs waiting to optimize for cost
- To elastically balance the cluster's load deterministically, predictably, and slowly.

Usage

You must already have a cluster running in order to use StarCluster's load balancer. Once you have a cluster running you can load balance the cluster by passing the name of the cluster to the *loadbalance* command:

```
$ starcluster loadbalance mycluster
```

This will start the load balancer in an infinite loop running on your local machine. The load balancer will continuously monitor the cluster's Oracle Grid Engine queue and determine whether it should add or remove nodes to the cluster or not. A running load balancer session can be safely stopped at any time by pressing CTRL-C. A new load balancing session can be resumed later simply by re-executing the above command.

Cluster Size Limits

If a cluster that's being load balanced experiences a heavy load for a sustained amount of time, the load balancer will begin adding nodes continuously in order to reduce the load. Once the cluster size has reached an upper-bound limit the load balancer will stop adding new nodes even if it thinks another node should be added. Conversely, if the cluster is idle for a prolonged amount of time the load balancer will begin removing nodes to reduce costs. Once the cluster size has reached a lower-bound limit the load balancer will stop removing nodes.

By default, the upper-bound, is set to the original **CLUSTER_SIZE** used when creating the cluster and the lower-bound is set to one. This means that the number of nodes in the cluster will fluctuate, by default, between 1 and **CLUSTER_SIZE** as necessary to optimize for cost and performance.

You can change the upper-bound limit, or maximum number of nodes, using the *-m* or *–max_nodes* option:

```
$ starcluster loadbalance -m 20 mycluster
```

The above command tells the load balancer to increase the size of the cluster as necessary up until there are twenty nodes in the cluster.

You can also change the lower-bound limit, or minimum number of nodes, using the *-n* or *–min_nodes* option:

```
$ starcluster loadbalance -n 3 mycluster
```

The above command tells the load balancer to decrease the size of the cluster as necessary until there are three nodes in the cluster.

Increasing Cluster Growth Rate

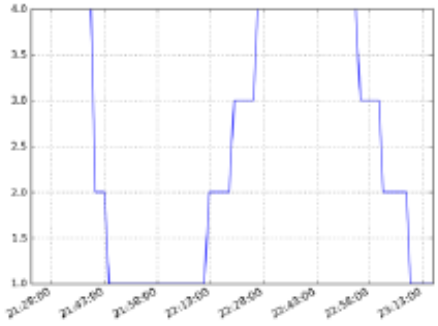
By default the load balancer will add a single node at a time when adding new nodes to the cluster. This allows the load balancer to gradually make a change and observe the impact on the queue without adding excessive resources. However, if you'd like to increase the number of nodes added when the load balancer determines more nodes are necessary use the *-a*, or *–add_nodes_per_iter*, option:

```
$ starcluster loadbalance -m 20 -a 2 mycluster
```

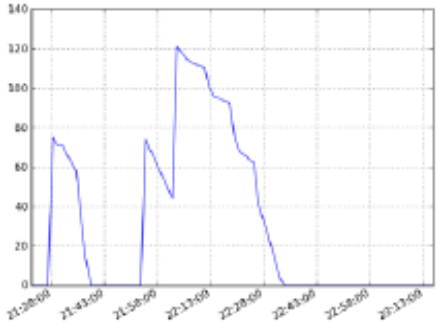
The above command will load balance *mycluster* up to a maximum of twenty nodes by adding two nodes at a time as necessary.

Load Balancer Statistics

The *loadbalance* command supports outputting various load balancing stats over time such as the number of nodes, number of running jobs, number of queued jobs, etc while it's running:



Number of Hosts



Number of Queued Jobs

To plot these stats over time in *png* format:

```
$ starcluster loadbalance -p mycluster
```

By default, this will generate the plots as *png* images in $HOME/.starcluster/sge/<cluster_tag>/. You can change where the load balancer outputs the images using the *-P* option:

```
$ starcluster loadbalance -p -P /path/to/stats/imgs/dir mycluster
```

You can also dump the raw stats used to build the above plots into a single csv file:

```
$ starcluster loadbalance -d mycluster
```

The above command will run the load balancer and output stats to a csv file. By default the stats are written to $HOME/.starcluster/sge/<cluster_tag>/sge-stats.csv, however, this can be changed using the *-D* option:

```
$ starcluster loadbalance -d -D /path/to/statsfile.csv mycluster
```

You can of course combine all of these options to generate both the plots and the raw statistics:

```
$ starcluster loadbalance -d -p mycluster
```

Advanced Configuration

The following parameters are also available for fine-tuning, however, the majority of users shouldn't need them:

1. **Polling interval** (-i INTERVAL or –interval=INTERVAL) - How often, in seconds, to collect statistics and make decisions
2. **Wait Time** (-w WAIT_TIME, –job_wait_time=WAIT_TIME) - Maximum wait time, in seconds, for a job before adding nodes
3. **Kill after** (-k KILL_AFTER, –kill_after=KILL_AFTER) - Minutes after which a node can be killed
4. **Stabilization time** (-s STAB, –stabilization_time=STAB) - How long, in seconds, to wait before cluster "stabilizes"
5. **Lookback window** (-l LOOKBACK_WIN, –lookback_window=LOOKBACK_WIN) - How long, in minutes, to look back for past job history

Experimental Features

The load balancer, by default, will not kill the master node in order to keep the cluster alive and functional. However, there are times when you might want to destroy the master if the cluster is completely idle and there are no more nodes left to remove. For example, you may wish to launch 10000 jobs and have the cluster shutdown when the last job has completed. In this case you can use the experimental *-K*, or *–kill-master*, option:

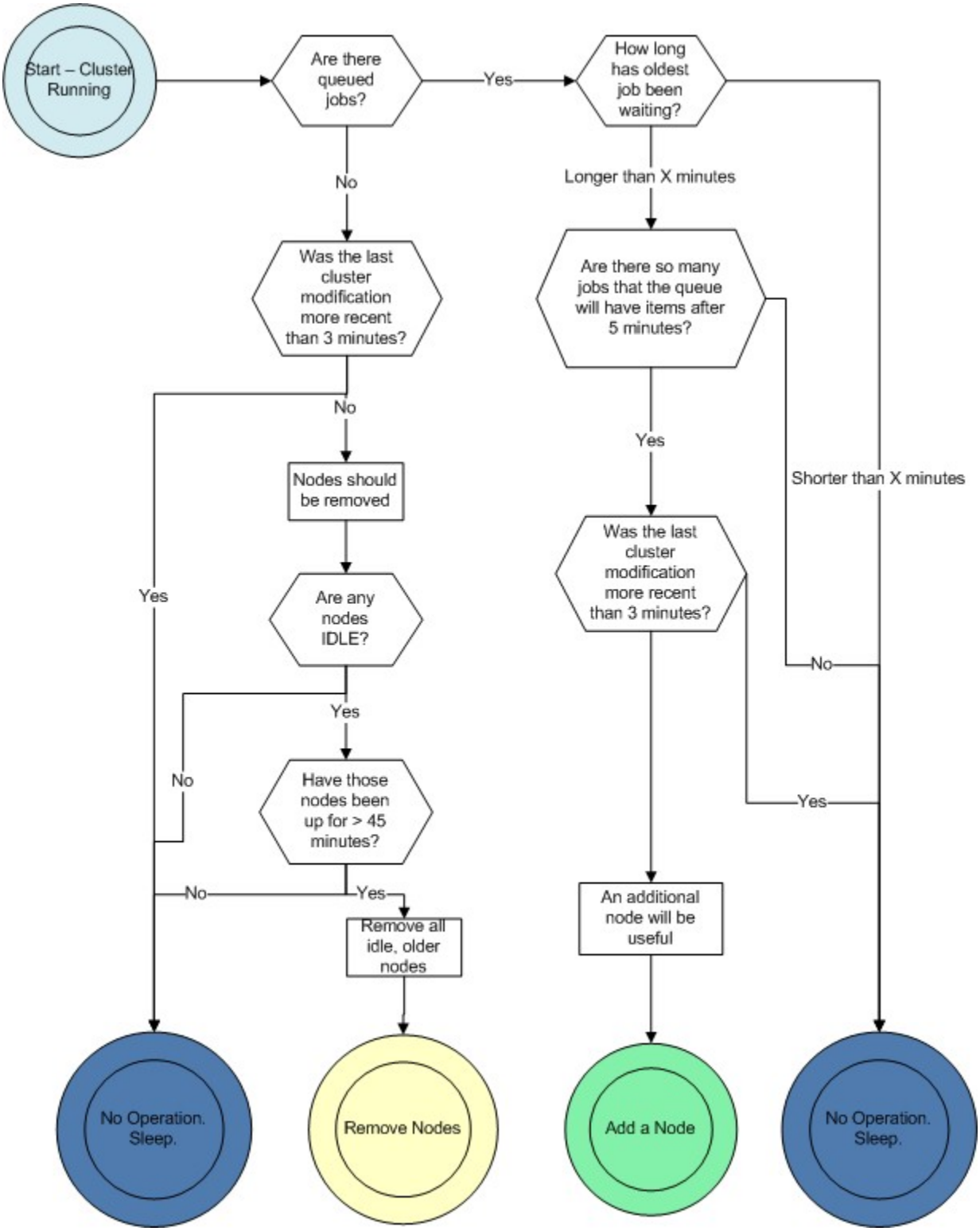
```
$ starcluster loadbalance --kill-master mycluster
```

The above command will load balance *mycluster* as usual, however, once all jobs have completed and all worker nodes have been shutdown by the load balancer the master node will also be terminated.

How it Works

There is a polling loop that runs every 60 seconds by default. The polling loop interval can be tuned using the *-i* configuration option discussed in the previous section. Every polling interval the load balancer will connect to the cluster, obtain statistics from Oracle Grid Engine, and decide whether or not to add or remove nodes based on the current job queue. The load balancer only deals only with the queue length and active machines. Currently the load balancer only supports monitoring the *default* queue, "all.q". Future releases will support balancing arbitrary once pull request 20 has been merged.

The diagram below illustrates the decisions that the load balancer will make in each loop:



Criteria for Adding a Node

A node will be added when *all* of the following criteria have been met:

1. There are jobs in the queued waiting (SGE's moniker is 'qw') state
2. The longest queued job has been waiting for more than 15 minutes
3. The number of nodes does not meet or exceed the maximum number of nodes set in the configuration file.

A user can set the number of nodes to be added per iteration. For instance, if the user wanted to add 1 node per iteration, which is standard and a recommended practice, they would set the *–add_nodes_per_iteration* parameter to one. If the user wanted two nodes to be added per iteration, that parameter should be set to two, and the cluster would grow at a faster rate, consequently incurring higher charges from Amazon.com.

Criteria for Removing a Node

A node will be removed when *all* of the following criteria have been met:

1. No jobs are in the queued waiting ('qw' state) state
2. The node in question is idle, meaning it is not running an SGE job
3. The node in question is not the master node
4. The node in question has been up for more than 45 minutes past the hour.

Each node in the cluster will be analyzed in turn, and any and all nodes meeting the above criteria will be terminated in that polling loop. The entire cluster need not be idle for a node to be terminated: If Node001 is working on a job, but Node002 is idle and there are no queued waiting jobs, Node002 is a candidate for termination.

The 45 Minutes Past the Hour Rule

Since Amazon charges by the hour, we are assuming that you have already paid for a full hour of server time. It would be wasteful to turn it off the moment it becomes idle. By keeping that node up for 45 minutes, we allow for it to complete the maximum workload from the queue, and use 75% of the hour you have already paid for.

Leaving a node up for this amount of time also increases the stability of the cluster. It is detrimental to the cluster and wasteful to be continuosly adding and removing nodes.

The Process of Adding a Node

Adding a new node is a multi-stage process:

1. Use the cluster class to start up a new node of the same instance and AMI as the other slave nodes in the cluster.
2. Wait for that node to come up. Name it with the highest Node # available: If Node001, Node003, and Node005, are started, the next node will be Node006.
3. Set up an /etc/hosts file on each node in the cluster, mapping the new node name to its ip address.
4. Create a cluster user account and cluster group on the new node.
5. Set up the /etc/exports file, creating the NFS shares for /home and sge on the master, and then exportfs so the shares are open to the slave nodes.
6. Mount the NFS shares on the new node.
7. Configure SGE: inform the master of the new host's address, and inform the new host of the master, and excute the sge commands to establish communications.

The Process of Removing a Node

Removing a node is also a multi-stage process:

1. Remove the node from SGE, so that no jobs can be sent to the node while it is in a transition period.
2. Remove the node from the /etc/hosts file on other cluster machines.
3. Remove the master's nfs export to this soon-to-be-killed node. Call exportfs to cut it off.
4. Terminate the node

Given that the node is immediately removed from SGE, and it seems like SGE takes about 15 seconds between a qsub command and a node beginning execution of a job, makes it very unlikely that a job will be started on a host as it is going down. There is a very small window of time within which this could happen.

Learning More

To learn more about the design and development of the load balancer please see Rajat Banerjee's master's thesis.

1.4.11 Enabling Tab Completion for Bash/Zsh

StarCluster has support for tab completion in both Bash and Zsh. If you're not familiar with tab completion, try typing **ls /** at a command prompt and then pressing the **Tab** key:

```
user@localhost % ls /
files
afs/         etc/         lib64/       mnt/         sbin/        usr/
bin/         home/        lost+found/  opt/         sys/         var/
boot/        lib@         media/       proc/        tera/
dev/         lib32/       mit/         root/        tmp/
```

Notice how after you pressed the **Tab** key the shell displayed a list of options for you to choose from. Typing a few more characters and pressing **Tab** again will reduce the number of options displayed:

```
user@localhost % ls /s
files
sbin/  sys/
```

Typing a **b** and pressing **Tab** would then automatically complete to **ls /sbin**.

Enabling StarCluster Tab Completion in BASH

To enable StarCluster bash-completion support for every shell you open, add the following line to your ~/.bashrc file:

```
source /path/to/starcluster/completion/starcluster-completion.sh
```

Enabling StarCluster Tab Completion in ZSH

To enable StarCluster zsh-completion support for every shell you open, add the following to the top of your ~/.zshrc file:

```
autoload -U compinit && compinit
autoload -U bashcompinit && bashcompinit
source /path/to/starcluster/completion/starcluster-completion.sh
```

Using StarCluster Tab Completion

After you've enabled StarCluster tab completion support in bash or zsh, you should now be able to tab complete all options to StarCluster actions.

For example, typing "starcluster –" and pressing the **Tab** key will show you a list of StarCluster's global options:

```
user@localhost % starcluster --
--config   --debug    --help     --version
```

This also works for each *action* in starcluster:

```
user@localhost % starcluster start --
--availability-zone     --help                  --master-instance-type
--cluster-shell         --key-location          --no-create
--cluster-size          --keyname               --node-image-id
--cluster-user          --login-master          --node-instance-type
--description           --master-image-id       --validate-only
```

Pressing **Tab** on just the start action will also list the possible arguments. Since start takes a *cluster template* as an argument, you'll notice that the cluster templates you defined in the config file show up in the list of suggestions:

```
user@localhost % starcluster start
-a                      -k                      -n
--availability-zone     -K                      --no-create
--cluster-shell         --key-location          --node-image-id
--cluster-size          --keyname               --node-instance-type
--cluster-user          -l                      -s
-d                      largecluster            -S
--description           --login-master          smallcluster
eucatest                -m                      -u
-h                      --master-image-id       -v
--help                  --master-instance-type  --validate-only
-i                      mediumcluster           -x
-I                      molsim
```

In the example above, *smallcluster*, *mediumcluster*, *largecluster*, etc are all cluster templates defined in ~/.starcluster/config. Typing an **s** character after the *start* action will autocomplete the first argument to *smallcluster*

The *start* action is not the only action supporting tab completion. Pressing **Tab** on the *sshmaster*, *sshnode*, and *sshinstance* actions will also complete based on active cluster names, instance ids, and dns names:

```
user@localhost % starcluster sshmaster
-h           --help       mycluster  -u           --user
```

In the above example, *mycluster* is a currently running StarCluster. Typing a **m** character and pressing **Tab** would autocomplete the command to *starcluster sshmaster mycluster*:

```
user@localhost % starcluster sshnode
% starcluster sshnode
0            3            6            9            mycluster
1            4            7            -h           -u
2            5            8            --help       --user
```

In the above example, *mycluster* is a currently running StarCluster. The shell also suggests numbers 0-9 because there are 10 machines running in *mycluster*:

```
user@localhost % starcluster sshinstance
ec2-123-123-123-137.compute-1.amazonaws.com
ec2-123-123-123-231.compute-1.amazonaws.com
```

```
ec2-123-123-123-16.compute-1.amazonaws.com
ec2-123-123-123-190.compute-1.amazonaws.com
ec2-123-123-123-41.compute-1.amazonaws.com
ec2-123-123-123-228.compute-1.amazonaws.com
ec2-123-123-123-180.compute-1.amazonaws.com
ec2-123-123-123-191.compute-1.amazonaws.com
ec2-123-123-123-228.compute-1.amazonaws.com
ec2-123-123-123-199.compute-1.amazonaws.com
-h
--help
i-91zz1bea
i-91zz1be8
i-91zz1bee
i-91zz1be6
i-91zz1be4
i-91zz1bf8
i-91zz1bfe
i-91zz1bfc
i-91zz2eca
i-91zz1bde
-u
--user
```

In the above example, pressing **Tab** after the *sshinstance* action will present a list of dns names and instance ids to ssh to. Typing a few more characters, such as *ec2-* will reduce the suggestions to only dns names:

```
user@localhost % starcluster sshinstance ec2-
ec2-123-123-123-137.compute-1.amazonaws.com
ec2-123-123-123-231.compute-1.amazonaws.com
ec2-123-123-123-16.compute-1.amazonaws.com
ec2-123-123-123-190.compute-1.amazonaws.com
ec2-123-123-123-41.compute-1.amazonaws.com
ec2-123-123-123-228.compute-1.amazonaws.com
ec2-123-123-123-180.compute-1.amazonaws.com
ec2-123-123-123-191.compute-1.amazonaws.com
ec2-123-123-123-228.compute-1.amazonaws.com
ec2-123-123-123-199.compute-1.amazonaws.com
```

Similarly for instance ids:

```
user@localhost % starcluster sshinstance i-
i-91zz1bea  i-91zz1be8  i-91zz1bee  i-91zz1be6  i-91zz1be4
i-91zz1bf8  i-91zz1bfe  i-91zz1bfc  i-91zz2eca  i-91zz1bde
```

These examples show a small subset of the actions that can be tab completed. Try tab-completing the other actions in starcluster to see their available options and suggestions for their arguments.

1.5 StarCluster Guides

Contents:

1.5.1 Sun Grid Engine (SGE) QuickStart

The Sun Grid Engine queuing system is useful when you have a lot of tasks to execute and want to distribute the tasks over a cluster of machines. For example, you might need to run hundreds of simulations/experiments with varying

parameters or need to convert 300 videos from one format to another. Using a queuing system in these situations has the following advantages:

- **Scheduling** - allows you to schedule a virtually unlimited amount of work to be performed when resources become available. This means you can simply submit as many tasks (or *jobs*) as you like and let the queuing system handle executing them all.
- **Load Balancing** - automatically distributes tasks across the cluster such that any one node doesn't get overloaded compared to the rest.
- **Monitoring/Accounting** - ability to monitor all submitted jobs and query which cluster nodes they're running on, whether they're finished, encountered an error, etc. Also allows querying job history to see which tasks were executed on a given date, by a given user, etc.

Note: Of course, just because a queuing system is installed doesn't mean you *have* to use it at all. You can run your tasks across the cluster in any way you see fit and the queuing system should not interfere. However, you will most likely end up needing to implement the above features in some fashion in order to optimally utilize the cluster.

Submitting Jobs

A job in SGE represents a task to be performed on a node in the cluster and contains the command line used to start the task. A job may have specific resource requirements but in general should be agnostic to *which* node in the cluster it runs on as long as its resource requirements are met.

Note: All jobs require *at least* one available slot on a node in the cluster to run.

Submitting jobs is done using the *qsub* command. Let's try submitting a simple job that runs the *hostname* command on a given cluster node:

```
sgeadmin@master:~$ qsub -V -b y -cwd hostname
Your job 1 ("hostname") has been submitted
```

- The **-V** option to *qsub* states that the job should have the same environment variables as the shell executing *qsub* (*recommended*)
- The **-b** option to *qsub* states that the command being executed could be a single binary executable or a bash script. In this case the command *hostname* is a single binary. This option takes a *y* or *n* argument indicating either *yes* the command is a binary or *no* it is not a binary.
- The **-cwd** option to *qsub* tells Sun Grid Engine that the job should be executed in the same directory that *qsub* was called.
- The last argument to *qsub* is the command to be executed (*hostname* in this case)

Notice that the *qsub* command, when successful, will print the job number to stdout. You can use the job number to monitor the job's status and progress within the queue as we'll see in the next section.

Monitoring Jobs in the Queue

Now that our job has been submitted, let's take a look at the job's status in the queue using the command *qstat*:

```
sgeadmin@master:~$ qstat
job-ID prior name user state submit/start at queue slots ja-task-ID
-----------------------------------------------------------------
1 0.00000 hostname sgeadmin qw 09/09/2009 14:58:00 1
sgeadmin@master:~$
```

From this output, we can see that the job is in the **qw** state which stands for *queued and waiting*. After a few seconds, the job will transition into a **r**, or *running*, state at which point the job will begin executing:

```
sgeadmin@master:~$ qstat
job-ID  prior   name       user         state submit/start at     queue  slots ja-task-ID
-----------------------------------------------------------------------------------------
1 0.00000 hostname   sgeadmin     r     09/09/2009 14:58:14                1
sgeadmin@master:~$
```

Once the job has finished, the job will be removed from the queue and will no longer appear in the output of *qstat*:

```
sgeadmin@master:~$ qstat
sgeadmin@master:~$
```

Now that the job has finished let's move on to the next section to see how we view a job's output.

Viewing a Job's Output

Sun Grid Engine creates stdout and stderr files in the job's working directory for each job executed. If any additional files are created during a job's execution, they will also be located in the job's working directory unless explicitly saved elsewhere.

The job's stdout and stderr files are named after the job with the extension ending in the job's number.

For the simple job submitted above we have:

```
sgeadmin@master:~$ ls hostname.*
hostname.e1 hostname.o1
sgeadmin@master:~$ cat hostname.o1
node001
sgeadmin@master:~$ cat hostname.e1
sgeadmin@master:~$
```

Notice that Sun Grid Engine automatically named the job *hostname* and created two output files: hostname.e1 and hostname.o1. The **e** stands for stderr and the **o** for stdout. The **1** at the end of the files' extension is the job number. So if the job had been named *my_new_job* and was job #23 submitted, the output files would look like:

```
my_new_job.e23 my_new_job.o23
```

Monitoring Cluster Usage

After a while you may be curious to view the load on Sun Grid Engine. To do this, we use the *qhost* command:

```
sgeadmin@master:~$ qhost
HOSTNAME ARCH NCPU LOAD MEMTOT MEMUSE SWAPTO SWAPUS
-------------------------------------------------------------------------------
global - - - - - - -
master lx24-x86 1 0.00 1.7G 62.7M 896.0M 0.0
node001 lx24-x86 1 0.00 1.7G 47.8M 896.0M 0.0
```

The output shows the architecture (**ARCH**), number of cpus (**NCPU**), the current load (**LOAD**), total memory (**MEMTOT**), and currently used memory (**MEMUSE**) and swap space (**SWAPTO**) for each node.

You can also view the average load (load_avg) per node using the '-f' option to *qstat*:

```
sgeadmin@master:~$ qstat -f
queuename qtype resv/used/tot. load_avg arch states
---------------------------------------------------------------------------
```

```
all.q@master.c BIP 0/0/1 0.00 lx24-x86
---------------------------------------------------------------------------------
all.q@node001.c BIP 0/0/1 0.00 lx24-x86
```

Creating a Job Script

In the 'Submitting a Job' section we submitted a single command *hostname*. This is useful for simple jobs but for more complex jobs where we need to incorporate some logic we can use a so-called *job script*. A *job script* is essentially a bash script that contains some logic and executes any number of external programs/scripts:

```
#!/bin/bash
echo "hello from job script!"
echo "the date is" `date`
echo "here's /etc/hosts contents:"
cat /etc/hosts
echo "finishing job :D"
```

As you can see, this script simply executes a few commands (such as echo, date, cat, etc) and exits. Anything printed to the screen will be put in the job's stdout file by Sun Grid Engine.

Since this is just a bash script, you can put any form of logic necessary in the job script (i.e. if statements, while loops, for loops, etc) and you may call any number of external programs needed to complete the job.

Let's see how you run this new job script. Save the script above to /home/sgeadmin/jobscript.sh on your StarCluster and execute the following as the sgeadmin user:

```
sgeadmin@master:~$ qsub -V jobscript.sh
Your job 6 ("jobscript.sh") has been submitted
```

Now that the job has been submitted, let's call *qstat* periodically until the job has finished since this job should only take a second to run once it's executed:

```
sgeadmin@master:~$ qstat
job-ID prior name user state submit/start at queue slots ja-task-ID
-------------------------------------------------------------------
6 0.00000 jobscript. sgeadmin qw 09/09/2009 16:18:43 1

sgeadmin@master:~$ qstat
job-ID prior name user state submit/start at queue slots ja-task-ID
-------------------------------------------------------------------
6 0.00000 jobscript. sgeadmin qw 09/09/2009 16:18:43 1

sgeadmin@master:~$ qstat
job-ID prior name user state submit/start at queue slots ja-task-ID
-------------------------------------------------------------------
6 0.00000 jobscript. sgeadmin qw 09/09/2009 16:18:43 1

sgeadmin@master:~$ qstat
job-ID prior name user state submit/start at queue slots ja-task-ID
-------------------------------------------------------------------
6 0.00000 jobscript. sgeadmin qw 09/09/2009 16:18:43 1

sgeadmin@master:~$ qstat
job-ID prior name user state submit/start at queue slots ja-task-ID
-------------------------------------------------------------------
6 0.55500 jobscript. sgeadmin r 09/09/2009 16:18:57 all.q@node001.c 1
```

```
sgeadmin@master:~$ qstat
sgeadmin@master:~$
```

Now that the job is finished, let's take a look at the output files:

```
sgeadmin@master:~$ ls jobscript.sh*
jobscript.sh jobscript.sh.e6 jobscript.sh.o6
sgeadmin@master:~$ cat jobscript.sh.o6
hello from job script!
the date is Wed Sep 9 16:18:57 UTC 2009
here's /etc/hosts contents:
# Do not remove the following line or programs that require network functionality will fail
127.0.0.1 localhost.localdomain localhost
10.252.167.143 master
10.252.165.173 node001
finishing job :D
sgeadmin@master:~$ cat jobscript.sh.e6
sgeadmin@master:~$
```

We see from looking at the output that the stdout file contains the output of the echo, date, and cat statements in the job script and that the stderr file is blank meaning there were no errors during the job's execution. Had something failed, such as a command not found error for example, these errors would have appeared in the stderr file.

Deleting a Job from the Queue

What if a job is stuck in the queue, is taking too long to run, or was simply started with incorrect parameters? You can delete a job from the queue using the *qdel* command in Sun Grid Engine. Below we launch a simple 'sleep' job that sleeps for 10 seconds so that we can kill it using *qdel*:

```
sgeadmin@master:~$ qsub -b y -cwd sleep 10
Your job 3 ("sleep") has been submitted
sgeadmin@master:~$ qdel 3
sgeadmin has registered the job 3 for deletion
```

After running *qdel* you'll notice the job is gone from the queue:

```
sgeadmin@master:~$ qstat
sgeadmin@master:~$
```

OpenMPI and Sun Grid Engine

Note: OpenMPI must be compiled with SGE support (–with-sge) to make use of the tight-integration between OpenMPI and SGE as documented in this section. This is the case on all of StarCluster's public AMIs.

OpenMPI supports tight integration with Sun Grid Engine. This integration allows Sun Grid Engine to handle assigning hosts to parallel jobs and to properly account for parallel jobs.

OpenMPI Parallel Environment

StarCluster by default sets up a parallel environment, called "orte", that has been configured for OpenMPI integration within SGE and has a number of *slots* equal to the total number of processors in the cluster. You can inspect the SGE parallel environment by running:

```
sgeadmin@ip-10-194-13-219:~$ qconf -sp orte
pe_name            orte
slots              16
user_lists         NONE
xuser_lists        NONE
start_proc_args    /bin/true
stop_proc_args     /bin/true
allocation_rule    $round_robin
control_slaves     TRUE
job_is_first_task  FALSE
urgency_slots      min
accounting_summary FALSE
```

This is the default configuration for a two-node, c1.xlarge cluster (16 virtual cores).

Round Robin vs Fill Up Modes

Notice the *allocation_rule* setting in the output of the *qconf* command in the previous section. This defines how to assign *slots* to a job. By default StarCluster configures *round_robin* allocation. This means that if a job requests 8 *slots* for example, it will go to the first machine, grab a single slot if available, move to the next machine and grab a single slot if available, and so on wrapping around the cluster again if necessary to allocate 8 *slots* to the job.

You can also configure the parallel environment to try and localize *slots* as much as possible using the *fill_up* allocation rule. With this rule, if a user requests 8 *slots* and a single machine has 8 *slots* available, that job will run entirely on one machine. If 5 *slots* are available on one host and 3 on another, it will take all 5 on that host, and all 3 on the other host. In other words, this rule will greedily take all *slots* on a given node until the slot requirement for the job is met.

You can switch between *round_robin* and *fill_up* modes using the following command:

```
$ qconf -mp orte
```

This will open up vi (or any editor defined in *EDITOR* env variable) and let you edit the parallel environment settings. To change from *round_robin* to *fill_up* in the above example, change the *allocation_rule* line from:

```
allocation_rule    $round_robin
```

to:

```
allocation_rule    $fill_up
```

After making the change and saving the file you can verify your settings using:

```
sgeadmin@ip-10-194-13-219:~$ qconf -sp orte
pe_name            orte
slots              16
user_lists         NONE
xuser_lists        NONE
start_proc_args    /bin/true
stop_proc_args     /bin/true
allocation_rule    $fill_up
control_slaves     TRUE
job_is_first_task  FALSE
urgency_slots      min
accounting_summary FALSE
```

Submitting OpenMPI Jobs using a Parallel Environment

The general workflow for running MPI code is:

1. Compile the code using mpicc, mpicxx, mpif77, mpif90, etc
2. Copy the resulting executable to the same path on all nodes or to an NFS-shared location on the master node

---

Note: It is important that the path to the executable is *identical* on all nodes for mpirun to correctly launch your parallel code. The easiest approach is to copy the executable somewhere under /home on the master node since /home is NFS-shared across all nodes in the cluster.

---

3. Run the code on *X* number of machines using:

```
$ mpirun -np X -hostfile myhostfile ./mpi-executable arg1 arg2 [...]
```

where the hostfile looks something like:

```
$ cat /path/to/hostfile
master  slots=2
node001 slots=2
node002 slots=2
node003 slots=2
```

However, when using an SGE parallel environment with OpenMPI **you no longer have to specify the -np, -hostfile, -host, etc options to mpirun**. This is because SGE will *automatically* assign hosts and processors to be used by OpenMPI for your job. You also do not need to pass the –byslot and –bynode options to mpirun given that these mechanisms are now handled by the *fill_up* and *round_robin* modes specified in the SGE parallel environment.

Instead of using the above formulation create a simple job script that contains a very simplified mpirun call:

```
$ cat myjobscript.sh
mpirun /path/to/mpi-executable arg1 arg2 [...]
```

Then submit the job using the *qsub* command and the *orte* parallel environment automatically configured for you by StarCluster:

```
$ qsub -pe orte 24 ./myjobscript.sh
```

The **-pe** option species which parallel environment to use and how many *slots* to request. The above example requests 24 *slots* (or processors) using the *orte* parallel environment. The parallel environment automatically takes care of distributing the MPI job amongst the SGE nodes using the *allocation_rule* defined in the environment's settings.

You can also do this without a job script like so:

```
$ cd /path/to/executable
$ qsub -b y -cwd -pe orte 24 mpirun ./mpi-executable arg1 arg2 [...]
```

1.6 Plugin Documentation

The links below are for plugin-specific docs. Please see the *plugin guide* for details on developing and using plugins.

1.6.1 IPython Cluster Plugin

To configure your cluster as an interactive IPython cluster use the ipcluster plugin:

```
[plugin ipcluster]
setup_class = starcluster.plugins.ipcluster.IPCluster
```

The next step is to put the ipcluster plugin in the list of plugins in one of your cluster templates:

```
[cluster smallcluster]
...
plugins = ipcluster
...
```

1.6.2 MySQL Cluster Plugin

This plugin automates the configuration and startup of MySQL Cluster on Ubuntu 10.04 Lucid Lynx. The mysql-cluster-server package suffers from an installation bug as of 7/21/10. This plugin works its way around that bug and results in an operational MySQL Cluster on initialization of the cluster.

Configuration Options

Here is an example of the mysqlcluster plugin section of the StarCluster config file:

```
[plugin mysqlcluster]
SETUP_CLASS = mysqlcluster.MysqlCluster
NUM_REPLICAS = 2
DATA_MEMORY = 80M
INDEX_MEMORY = 18M
DATA_DIR = /var/lib/mysqlcluster
BACKUP_DATA_DIR = /var/lib/mysqlcluster/
DEDICATED_QUERY = True
NUM_DATA_NODES = 2
```

NUM_REPLICAS: Specifies number of replicas for each table in the cluster, as well as the number of node groups. The maximum value is 4, and only values 1 and 2 are currently supported by MySQL. A value of 1 would indicate that that there is only one copy of your data. The loss of a single data note would therefore cause your cluster to fail, as there are no additional copies of that data. The number of replicas must also divide evenly into the number of data nodes. '2' is both the recommended and default setting for this parameter.

DATA_MEMORY: Amount of space (in bytes) available for storing database records. Suffixes K, M, and G can be used to indicated Kilobytes, Megabytes, or Gigabytes. Default value is 80M, minimum is 1M.

INDEX_MEMORY: Amount of storage used for hash indexes in the MySQL Cluster. Default value is 18M, minimum is 1M.

DATA_DIR: Specifies the directory where metadata, REDO logs, UNDO logs (for Disk Data tables), data files, trace files, log files, pid files and error logs are placed.

BACKUP_DATA_DIR: Specifies the directory in which to put the BACKUP directory. Defaults to DATA_DIR.

DEDICATED_QUERY: True indicates that the data nodes do not also function as query nodes, and there are instead dedicated nodes to accept queries. False indicates that all data nodes will also accept queries.

NUM_DATA_NODES: Number of data nodes if DEDICATED_QUERY is set to True. The remaining nodes in the cluster will be MySQL query nodes.

What This Plugin Does

1. Creates data and backup directories, changes ownership to mysql user

2. Generates /etc/mysql/ndb_mgmd.cnf configuration file on master.
3. Generates /etc/mysql/my.cnf configuration file on all nodes.
4. Kills mysql processes on all nodes.
5. Starts Management Client on master.
6. Starts mysql on query nodes
7. Starts mysql-ndb on data nodes.

Creating a Replicated Table

Here is an example of how to create a table that is replicated across the cluster. Do this on one of the data nodes:

```
mysql> create database testdb;
Query OK, 1 row affected (0.00 sec)
mysql> use testcluster;
Database changed
mysql> create table testtable (i int) engine=ndbcluster;
Query OK, 0 rows affected (0.71 sec)
mysql> insert into testtable values (1);
Query OK, 1 row affected (0.05 sec)
mysql> select * from testtable;
+------+
| i    |
+------+
|    1 |
+------+
1 row in set (0.03 sec)
```

Note that 'engine=ndbcluster' is what indicates that the table should be created in a cluster configuration. If it is not used, the table will not be replicated.

On another data node:

```
mysql> use testdb;
Database changed
mysql> select * from testtable;
+------+
| i    |
+------+
|    1 |
+------+
1 row in set (0.04 sec)
```

The table has been replicated, and the cluster is working.

Recommendations for Use

- Clusters should have three nodes at the very least.
- NUM_REPLICAS should probably stay at 2. Consequently, there should be an even number of data nodes.
- Set DATA_DIR and BACKUP_DATA_DIR to an EBS volume if you want the data to persist.

1.7 Frequently Asked Questions

TODO

1.8 StarCluster Support

The StarCluster project has several ways to get support: the user mailing list, the StarCluster github issue tracker, and the **#starcluster** IRC channel on freenode.

1.8.1 Submitting Bug Reports

If you've found a bug in StarCluster's code, please submit a bug report to the github issue tracker. In the case of an unrecoverable error, or crash, StarCluster will create a *crash file* containing debugging logs useful for diagnosing the issue. Below is an example of a crash due to a bug in the code:

```
% starcluster start -s 2 -v mycluster
StarCluster - (http://web.mit.edu/starcluster)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

>>> Using default cluster template: smallcluster
Traceback (most recent call last):
  File "/workspace/starcluster/starcluster/cli.py", line 155, in main
    sc.execute(args)
  File "/workspace/starcluster/starcluster/commands/start.py", line 180, in execute
    scluster = self.cm.get_cluster_template(template, tag)
  File "/workspace/starcluster/starcluster/cluster.py", line 82, in get_cluster_template
    ec2_conn=self.ec2)
  File "/workspace/starcluster/starcluster/config.py", line 589, in get_cluster_template
    clust = Cluster(ec2_conn, **kwargs)
  File "/workspace/starcluster/starcluster/cluster.py", line 315, in __init__
    blah = blah
UnboundLocalError: local variable 'blah' referenced before assignment

!!! ERROR - Oops! Looks like you've found a bug in StarCluster
!!! ERROR - Crash report written to: /home/myuser/.starcluster/crash-report-6029.txt
!!! ERROR - Please remove any sensitive data from the crash report
!!! ERROR - and submit it to starcluster@mit.edu
```

Each time StarCluster encounters a crash, as in the example above, a new crash report will be written to $HOME/.starcluster/crash-report-$PID.txt where *$PID* is the process id of the buggy StarCluster session. When a crash report is generated users should first check the crash report for any sensitive data that might need to be removed and then submit a bug report with the crash report attached.

1.8.2 Issues and Questions

For all other issues, questions, feature requests, etc. you can either:

Note: It's highly preferred that bug reports are submitted to the github issue tracker if possible.

---

1. Submit a new issue to the StarCluster github issue tracker (recommended)

2. Send a report via email to the user mailing list (**please join the list!**)
3. Join the **#starcluster** IRC channel on freenode and ask your questions/issues there. If no one's around please post to the mailing list instead.

1.9 Contributing to StarCluster

Note: Prerequisites: You need to install git before following these instructions. You should also familiarize yourself with the basic use and work flow model of git before following these instructions. The folks over at github put together a good introduction to git that you can use to get started with git.

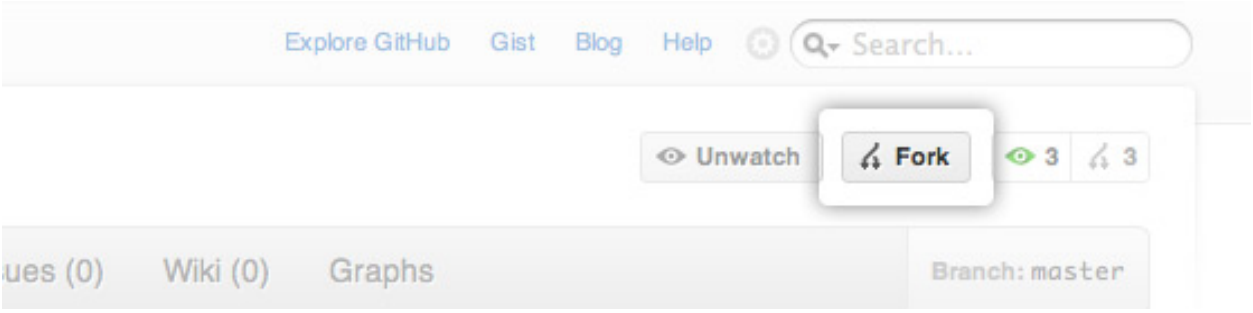
1.9.1 Overview

1.9.2 Sign-up for a github acount

StarCluster's source code is stored on github.com. It is preferred that you use github.com to submit patches and enhancements via pull requests. The first step is to sign up for a github account.

1.9.3 Fork the StarCluster project

Once you have a github account the next step is to fork the StarCluster github repository. To do this you must first login to github and then navigate to the StarCluster repository. Once there click on the **Fork** button towards the top right of the project page:



This will create your own copy of the StarCluster repository under your github account that you can modify and commit to. Having your own copy allows you to work on bug fixes, docs, new features, etc. without needing special commit access to the main StarCluster repository.

1.9.4 Setup a virtualenv for StarCluster development

When developing a Python project it's useful to work inside an isolated Python environment that lives inside your *$HOME* folder. This helps to avoid dependency version mismatches between projects and also removes the need to obtain root priviliges to install Python modules/packages for development.

Fortunately there exists a couple of projects that make creating and managing isolated Python environments quick and easy:

- virtualenv - Virtual Python Environment builder
- virtualenvwrapper - Shell enhancements for virtualenv

To get started you first need to install and configure virtualenv and virtualenvwrapper:

> **Warning:** You need *root* access to run the *sudo* commands below.

```
$ sudo easy_install virtualenv
$ sudo easy_install virtualenvwrapper
$ mkdir $HOME/.virtualenvs
$ echo "source /usr/local/bin/virtualenvwrapper.sh" >> $HOME/.bashrc
```

If you're using zsh then the last line should be changed to:

```
$ echo "source /usr/local/bin/virtualenvwrapper.sh" >> $HOME/.zshrc
```

Running these commands will install both virtualenv and virtualenvwrapper and configure virtualenvwrapper to use **$HOME/.virtualenvs** as the top-level virtual environment directory where all virtual environments are installed.

At this point you will either need to close your current shell and launch a new shell or *re-source* your shell's *rc* file:

```
$ source $HOME/.bashrc
```

This will reload your shell's configuration file and configure virtualenvwrapper. The next step is to create a new virtual environment called *starcluster* and change into that virtual environment:

```
$ mkvirtualenv --clear --no-site-packages --distribute starcluster
(starcluster)$ echo $PATH
/home/user/.virtualenvs/starcluster/bin:/usr/local/bin:/usr/bin:/bin:/opt/bin
```

Running this command will create a new folder *$HOME/.virtualenvs/starcluster* containing your new isolated Python environment for StarCluster. This command will also modify your current shell's environment to work with the StarCluster virtual environment. As you can see from the *echo $PATH* command above your PATH environment variable has been modified to include the virtual environment's *bin* directory at the front of the path. This means when you type *python* or other Python-related commands (e.g. easy_install, pip, etc) you will be using the virtual environment's isolated Python installation.

To see a list of your virtual environments:

```
$ workon
starcluster
```

To *activate* (or enter) a virtual environment:

```
$ workon starcluster
(starcluster)$ cdvirtualenv
(starcluster)$ pwd
/home/user/.virtualenvs/starcluster
(starcluster)$ ls
bin  build  include  lib  lib64  man  share
```

To *de-activate* (or leave) a virtual environment:

```
(starcluster)$ deactivate
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/opt/bin
```

Installing packages within your virtual environment is the same as outside the virtual environment except you don't need *root* privileges. You can use either *easy_install* or *pip* to install Python packages/modules. Since IPython is required to use StarCluster's development shell let's install IPython now to test this out:

```
(starcluster)$ pip install ipython
```

This will install IPython within the virtual environment. You can verify this using the following command:

```
(starcluster)$ which ipython
/home/user/.virtualenvs/starcluster/bin/ipython
```

1.9.5 Clone your fork

Now that you have a working virtual environment for StarCluster it's time to check out your fork of StarCluster:

Note: Replace **<user>** in the *git clone* command below with your github username.

```
$ workon starcluster
(starcluster)$ cdvirtualenv
(starcluster)$ git clone <user>@github.com:<user>/StarCluster.git starcluster
```

The *git clone* command above will checkout StarCluster's source files to $HOME/.virtualenvs/starcluster/starcluster. The next step is to configure StarCluster's Python source for development. To do this run the following command:

```
$ workon starcluster
(starcluster)$ cd $VIRTUAL_ENV/starcluster
(starcluster)$ python setup.py develop
```

The *python setup.py develop* command will install StarCluster into the virtual environment's site-packages in such a way that the sources are *linked* rather than copied to the site-packages directory.

Note: This has the benefit that as soon as a change is made in the StarCluster source files the changes will show up immediately in the virtual environment's site-packages directory. If you were to use *python setup.py install* you would instead need to re-install StarCluster each time you made a change for the changes to become active in the virtual environment's Python installation.

1.9.6 Code clean-up

Before committing any code please be sure to run the check.py script in the root of your StarCluster git repository. This script runs pep8 and pyflakes on all source files and outputs any errors it finds related to pep8 formatting, syntax, import errors, undefined variables, etc. Please fix any errors reported before committing.

PEP 8

```
$ cd $STARCLUSTER_REPO
$ pip install pep8
$ pip install pyflakes
$ ./check.py
>>> Running pyflakes...
>>> Running pep8...
>>> Clean!
```

1.9.7 Submit your changes upstream

Once you've finished fixing bugs or adding features you're now ready to submit a pull request so that the changes can be merged upstream and be included in the next stable release.

1.10 Hacking

TODO

1.11 Features

- Simple configuration with sensible defaults
- Single "start" command to automatically launch and configure one or more clusters on EC2
- Support for attaching and NFS-sharing Amazon Elastic Block Storage (EBS) volumes for persistent storage across a cluster
- Comes with a publicly avilable Amazon Machine Image (AMI) configured for scientific computing
- AMI includes OpenMPI, ATLAS, Lapack, NumPy, SciPy, and other useful libraries
- Clusters are automatically configured with NFS, Sun Grid Engine queuing system, and password-less ssh between machines
- Supports user-contributed "plugins" that allow users to perform additional setup routines on the cluster after StarCluster's defaults
- EBS-Backed Clusters - Added support for starting/stopping EBS-backed clusters on EC2.
- Cluster Compute Instances - Added support for the new Cluster Compute instance type. Thanks to Fred Rotbart for his contributions.
- Ability to Add/Remove Nodes- Added new addnode and removenode commands for adding/removing nodes to a cluster and removing existing nodes from a cluster.
- Restart command - Added new restart command that reboots the cluster and reconfigures the cluster.
- Create Keypairs - Added ability to add/list/remove keypairs
- Elastic Load Balancing - Support for shrinking/expanding clusters based on Sun Grid Engine queue statistics. This allow the user to start a single-node cluster (or larger) and scale the number of instances needed to meet the current queue load. For example, a single-node cluster can be launched and as the queue load increases new EC2 instances are launched, added to the cluster, used for computation, and then removed when they're idle. This minimizes the cost of using EC2 for an unknown and on-demand workload. This feature is now available in the latest github code thanks to Rajat Banerjee (rqbanerjee).
- Security Group Permissions - Added ability to specify permission settings to be applied automatically to a cluster's security group after it's been started.
- Multiple Instance Types - Added support for specifying instance types on a per-node basis. Thanks to Dan Yamins for his contributions.
- Unpartitioned Volumes - StarCluster now supportsboth partitioned and unpartitioned EBS volumes.
- New Plugin Hooks - Plugins can now play a part when adding/removing a node as well as when restarting/shutting down the entire cluster by implementing the on_remove_node/on_add_node/on_shutdown/on_reboot methods

1.12 StarCluster TODOs

Below are the current feature requests for the StarCluster project that need implementing:

1.12.1 Config Includes

Allow including multiple files in the StarCluster config file. This would allow users to separate their *private* credentials from their cluster templates. This is really useful if you want to, say, share a common config between members of a group. Each member would have their credentials stored in a separate credentials file, say *$HOME/.starcluster/creds*, and could then easily pull in the latest config updates from the web, git, etc. For example:

```
[include ~/.starcluster/creds]
```

This line would pull in any config sections defined in *$HOME/.starcluster/creds*. Here's an example *creds* file:

```
[aws info]
aws_access_key_id = #your aws access key id
aws_secret_access_key = #your aws secret access key
aws_user_id = #your aws user id
```

The main config file, *$HOME/.starcluster/config*, would then include this file to pick up the credentials:

```
[global]
default_template = smallcluster

[include ~/.starcluster/creds]

[cluster smallcluster]
cluster_size = 4
keypair = mykey
node_instance_type = m1.large
node_image_id = ami-0af31963
```

As you can see this removes the private sections from the main StarCluster config file which in turn makes the main config file much easier to share.

1.12.2 Support for Amazon VPC

Add the ability to configure StarCluster to launch the cluster instances inside of an Amazon VPC. Should just involve adding a *SubnetId* parameter to the run_instances boto call. Feature requested by Adam Kraut.

1.12.3 Improved Eucalyptus Support

Need to subclass starcluster.cluster.Cluster and starcluster.awsutils.EasyEC2 for Eucalyptus to handle the lack of the following API features in ECC:

- DescribeInstanceAttribute
- Tags API
- Filters API

1.12.4 Add More Options to "addnode" command

The following options need to be added to the addnode command:

- –image-id (-i)
- –instance-type (-I)
- –availability-zone (-z)

- –bid (-b)

Dan Yamins has a pull request for this that needs to be merged.

1.12.5 Add Support for Load Balancing a Given SGE Queue

Load balancer should support balancing Sun Grid Engine queues other than just all.q. This is useful if you want to load balance many different queues with varying configurations. In this case you can launch a separate load-balancer process for each queue.

Dan Yamins has a pull request for this that needs to be merged.

1.12.6 Add HTTP Proxy Settings for Boto to StarCluster Config

It looks like boto supports using an HTTP proxy: http://code.google.com/p/boto/wiki/BotoConfig

Need to add a *[proxy]* section to the starcluster config that gets passed on to the boto connection.

INDEX

P