
StarCluster Documentation

Release 0.95.6

Justin Riley

March 04, 2015

1	Master Table of Contents	3
1.1	What is StarCluster?	3
1.2	Installing StarCluster	8
1.3	Quick-Start	10
1.4	StarCluster User Manual	12
1.5	StarCluster Guides	55
1.6	Plugin Documentation	61
1.7	Frequently Asked Questions	93
1.8	StarCluster Support	94
1.9	Contributing to StarCluster	95
1.10	Hacking	98
1.11	Features	98
1.12	StarCluster TODOs	99
1.13	Version History	100
1.14	License: LGPL (version 3)	117

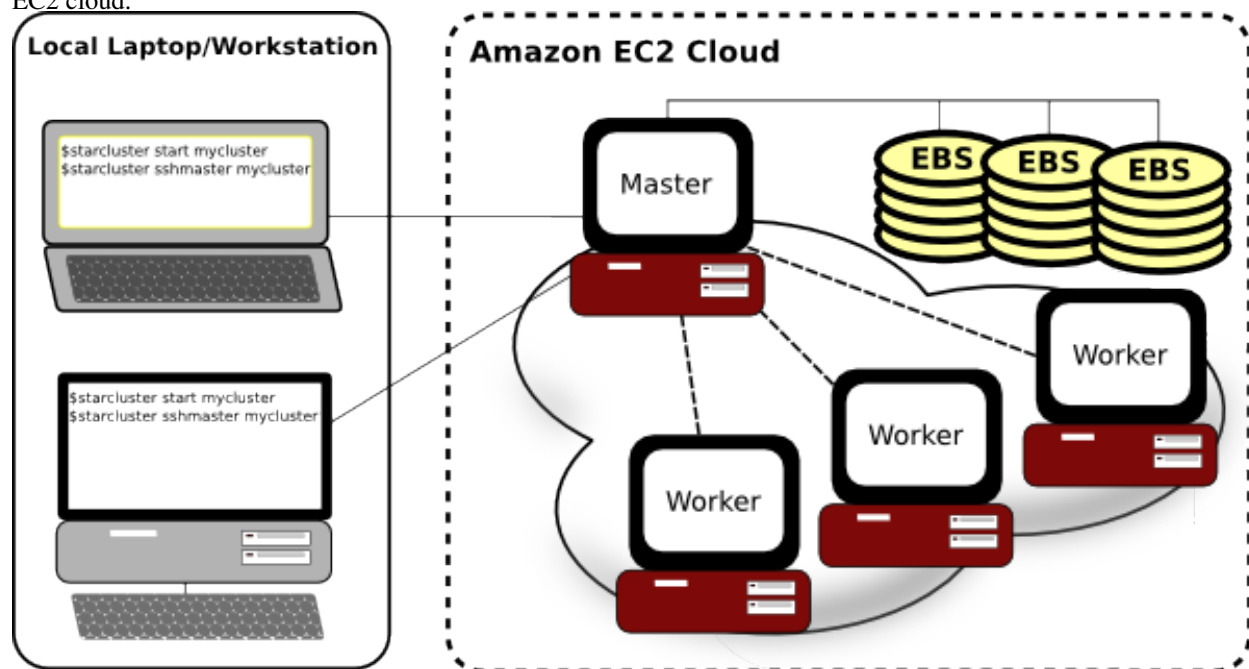
Contents:

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://star.mit.edu/cluster)
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-999999999 ec2-123-123-123-123.compute-1.amazonaws.com
    node001 running i-888888888 ec2-123-123-123-132.compute-1.amazonaws.com
    node002 running i-777777777 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 its 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* command:

```
$ 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.

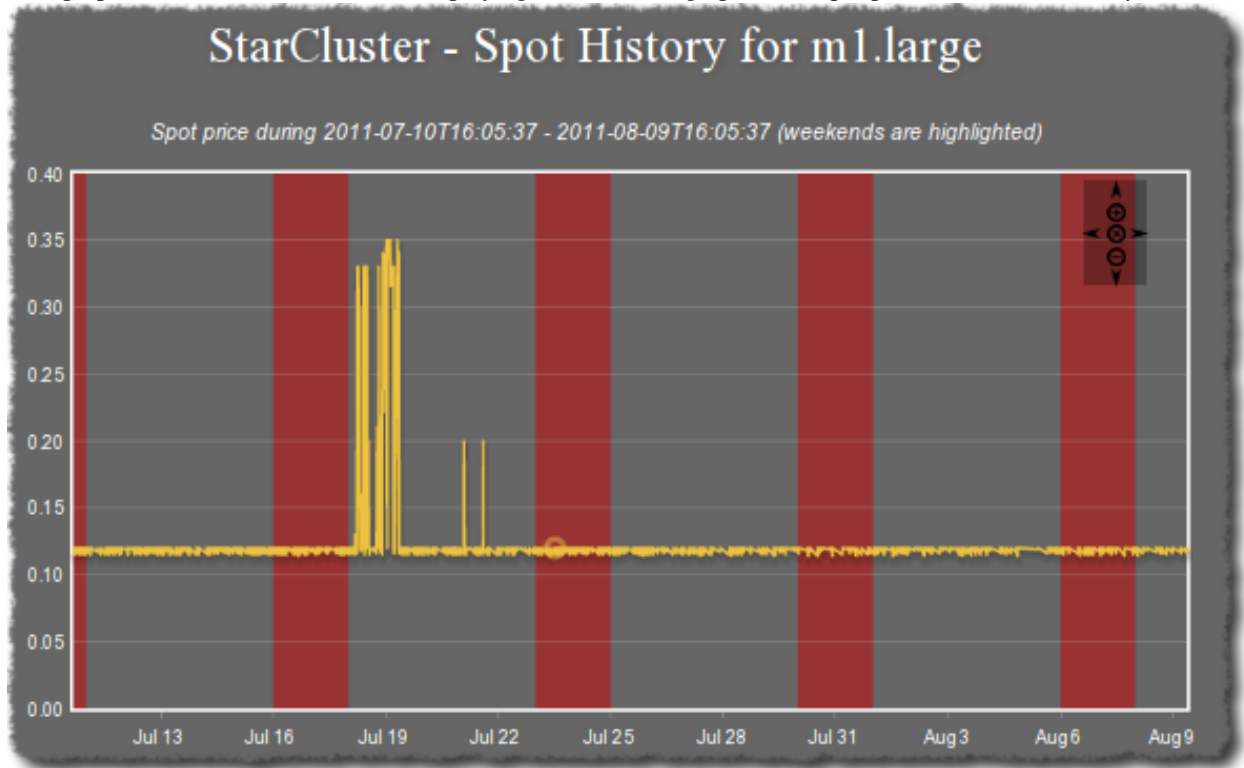
You can determine a decent spot bid to use by investigating the current, maximum, and average spot price using the *spothistory* command:

```
% starcluster spothistory -p m1.large
StarCluster - (http://star.mit.edu/cluster)
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
```

1.2.3 Installing on Windows

Before attempting to install StarCluster you first need to install Python 2.7 for Windows from python.org. You'll want to download the "Python 2.7.x Windows Installer".

Once you have Python 2.7 installed the next step is to download and install the Windows installers for the following dependencies:

- [setuptools 0.6rc11](#)
- [pycrypto 2.3](#)

Note: You will need to have your Python installation's Script directory (e.g. C:\Python27\Scripts) added to the end of your %PATH% variable in order to use both the `easy_install` and `starcluster` commands below

Once you've installed the above dependencies into your Python 2.7 installation you can now run:

```
c:\> easy_install StarCluster
```

Once the install is finished you're now ready to setup the configuration file:

```
c:\> 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://star.mit.edu/cluster)
$ 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://star.mit.edu/cluster)
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://star.mit.edu/cluster>

To learn more have a look at the rest of the documentation: <http://star.mit.edu/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 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]**, **[permission]**, 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 at the default config location (`~/.starcluster/config`), simply run "starcluster help" at the command-line. Provided that the configuration file does not exist, you will see the following:

```
$ starcluster help
StarCluster - (http://star.mit.edu/cluster)
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

Please 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` that is set as the default *cluster template*.

Storing the config in an alternate location

If you wish to store your StarCluster config in a location other than the default (`~/.starcluster/config`), you will need to set the `STARCLUSTER_CONFIG` environment variable to point to your file:

```
$ export STARCLUSTER_CONFIG="/path/to/starcluster/config"
```

After doing so, all StarCluster commands will use the config identified by `STARCLUSTER_CONFIG`.

Alternatively, you can specify the global `--config (-c)` option with every StarCluster command you use. For example:

```
$ starcluster -c /path/to/starcluster/config listclusters
```

In either case, if the config didn't exist at the specified path you would be prompted with the same menu above offering to generate a template at the specified path:

```
$ starcluster -c /path/to/nonexistent/config listclusters
StarCluster - (http://star.mit.edu/cluster)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu
```

```
cli.py:475 - ERROR - Config file /path/to/nonexistent/config does not exist
```

Options:

```
[1] Show the StarCluster config template
[2] Write config template to /path/to/nonexistent/config
[q] Quit
```

Please enter your selection:

Loading the config from the web

The config can also be loaded from a web url, however, if you choose to do so you should be *very* careful not to publicly host AWS credentials, keys, and other private information:

```
$ starcluster -c http://localhost/starcluster.cfg listclusters
```

See also:

See also: [Splitting the Config into Multiple Files](#)

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
```

All of the settings in the **[aws info]** section can be overridden by the environment. StarCluster will log a warning whenever it uses settings from the environment. For example:

```
$ export AWS_ACCESS_KEY_ID=your_aws_access_key_id
$ export AWS_SECRET_ACCESS_KEY=your_secret_access_key
$ starcluster listclusters
*** WARNING - Setting 'AWS_SECRET_ACCESS_KEY' from environment...
*** WARNING - Setting 'AWS_ACCESS_KEY_ID' from environment...
```

Amazon EC2 Regions

StarCluster uses the us-east-1 EC2 region by default. If you wish to permanently 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.

Switching Regions via Command Line

StarCluster also supports quickly switching between EC2 regions via the command line without having to change your config. To switch regions at the command line use the global `-r` (*-region*) option:

```
$ starcluster -r us-west-1 listpublic
```

The above example runs the **listpublic** command in the `us-west-1` region. Similarly, you will need to pass the global `-r` option to all of StarCluster's commands in order to switch regions via the command line.

See also:

See also: *Tips for Switching Regions*

Amazon S3 Region-Specific Endpoints

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.

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

Using a Proxy Host

StarCluster can also be configured to use a proxy host when connecting to AWS by specifying the following settings in your **[aws info]** section:

aws_proxy - The name of the proxy host to use for connecting to AWS.

aws_proxy_port - The port number to use to connect to the proxy host.

aws_proxy_user - The user name to use when authenticating with proxy host.

aws_proxy_pass - The password to use when authenticating with proxy host.

StarCluster will use the settings above when creating the `boto` connection object used to communicate with AWS. Example:

```
[aws info]
aws_proxy = your.proxyhost.com
aws_proxy_port = 8080
aws_proxy_user = yourproxyuser
aws_proxy_pass = yourproxypass
```

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 to obtain a list of your current EC2 keypairs. 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 template*'s **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 [AWS web console](#).

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:

```
[cluster smallcluster]
keyname = mykeypair1
cluster_size = 2
cluster_user = sgeadmin
cluster_shell = bash
master_image_id = ami-0330d16a
master_instance_type = m1.small
node_image_id = ami-0330d16a
node_instance_type = m1.small
```

Cluster Settings

The table below describes all required and optional settings for a cluster template in detail.

Setting	Required	Description
keyname	Yes	The keypair to use for the cluster (the keypair must be defined in a [keypair] section)
cluster_size	Yes	Number of nodes in the cluster (including master)
node_image_id	Yes	The AMI to use for worker nodes
node_instance_type	Yes	The instance type for worker nodes
cluster_user	No	The cluster user to create (defaults to sgeadmin)
cluster_shell	No	Sets the cluster user's shell (default: bash, options: bash, zsh, csh, ksh, tcsh)
dns_prefix	No	If True, prefixes the dns name of nodes with the cluster tag. For example: master → mycluster-master
master_image_id	No	The AMI to use for the master node. (defaults to node_image_id)
master_instance_type	No	The instance type for the master node. (defaults to node_instance_type)
userdata_scripts	No	List of userdata scripts to use when launching instances
volumes	No	List of EBS volumes to attach and NFS-share to the cluster (each volume must be defined in a [volume] section)
plugins	No	List of StarCluster plugins to use when launching the cluster (each plugin must be defined in a [plugin] section)
permissions	No	List of permissions to apply to the cluster's security group (each permission must be defined in a [permission] section)
user_data_scripts	No	List of user data scripts to run on boot for each instance in the cluster
spot_bid	No	Always use spot instances with this cluster template
force_spot_master	No	When requesting a spot cluster this setting forces the master node to also be a spot instance (default is for master not to be a spot instance for stability)
availability_zone	No	Launch all cluster instances in a single availability zone (defaults to any available zone)
disable_queue	No	Disables the setup and configuration of the Open Grid Scheduler (OGS, formerly SGE)
disable_cloudinit	No	Do not use cloudinit for cluster accounting (only required if using non- cloudinit enabled AMIs)
subnet_id	No	The VPC subnet to use when launching cluster instances
public_ips	No	Automatically assign public IP addresses to all VPC cluster instances. Default is <i>False</i> . WARNING: Enabling public IPs exposes your VPC cluster nodes to the internet which may not be desirable. This option also requires a special VPC configuration - see Connecting to a VPC Cluster

Using the Virtual Private Cloud (VPC)

From Amazon's [VPC page](#):

"Amazon Virtual Private Cloud (Amazon VPC) lets you provision a logically isolated section of the Amazon Web Services (AWS) Cloud where you can launch AWS resources in a virtual network that you define. You have complete control over your virtual networking environment, including selection of your own IP address range, creation of subnets, and configuration of route tables and network gateways."

New AWS accounts use VPC by default via the [default VPC](#) and StarCluster supports this configuration without user intervention. However, users that wish to launch clusters in a **non-default** VPC must also provide the **subnet_id** setting in their cluster template(s):

```
[cluster smallcluster]
keyname = mykeypair1
cluster_size = 2
node_image_id = ami-0330d16a
node_instance_type = m1.small
subnet_id = subnet-99999999
```

Alternatively, users can specify or override the subnet ID at runtime via the `--subnet-id (-N)` option to the `start` command:

```
$ starcluster start -N subnet-88888888 mycluster
```

Connecting to a VPC Cluster

By default StarCluster does **not** automatically assign a public IP address to all VPC cluster instances which means **you must be on a machine within the VPC in order to successfully create, connect, and configure a cluster in the VPC** - otherwise StarCluster will hang indefinitely trying to connect to the nodes. StarCluster does not assign public IPs by default for two reasons:

1. It opens the VPC to the internet which is a security risk
2. It requires a special VPC configuration before it can be used successfully

Specifically, your non-default VPC must have:

1. An internet gateway attached to the VPC
2. A route table entry linked to the internet gateway and associated with the cluster's VPC subnet that has a destination CIDR block of `0.0.0.0/0`

StarCluster will raise a validation error if public IPs are requested and these requirements are not met. Assuming you're aware of the risks and have configured your VPC as mentioned above you can enable public IP addresses by setting `public_ips=True` in your cluster config:

Warning: Enabling public IPs means that all VPC cluster nodes will be accessible from the internet which may not be desirable depending on your network/security requirements.

```
[cluster smallcluster]
keyname = mykeypair1
cluster_size = 2
node_image_id = ami-0330d16a
node_instance_type = m1.small
subnet_id = subnet-99999999
public_ips = True
```

This configuration will launch a cluster in a non-default VPC subnet and automatically assign a public IP address to all VPC cluster instances. You can also enable public IPs using the `--public-ips` option to the `start` command:

```
$ starcluster start -N subnet-88888888 --public-ips mycluster
```

Note: The `--public-ips` option only applies to **non-default** VPC clusters - this option is *not* needed for clusters using the default VPC or EC2 classic. Both the default VPC and EC2 classic assign public IPs automatically.

Once public IPs have been enabled you can launch a cluster inside the VPC from a machine (e.g. your laptop) outside the VPC.

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 (-c)` 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-c99999999` and `vol-c88888888`. 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.

See also:

See the *Using EBS Volumes for Persistent Storage* documentation to learn how to use StarCluster to easily create, format, and configure new EBS volumes.

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 (i.e. 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 [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, `myplug1_arg_one` and `myplug2_arg_one` are arguments to the plugin’s `setup_class`. The argument names were made up for this example. The names of a plugin’s arguments in general depends 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
# the rest 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:

Learn more about the [Plugin System](#)

Splitting the Config into Multiple Files

In some cases you may wish to split your configuration file into separate files for convenience. For example, you may wish to organize all keypairs, cluster templates, permissions, volumes, etc. into separate files to make it easier to access the relevant settings without browsing the entire config all at once. To do this, simply create a new set of files and put the relevant config sections into the files:

Note: The following list of files are just examples. You are free to create any number of files, name them anything you want, and distribute any of the sections in the config to these files in any way you see fit. The only exception is that the **[global]** section *must* live in either the default config `$HOME/.starcluster/config` or the config specified by the global `--config (-c)` command line option.

File: `$HOME/.starcluster/awskeys`

```
[aws info]
aws_access_key_id = #your_aws_access_key_id
aws_secret_access_key = #your_secret_access_key
```

```
[key mykey1]
key_location=/path/to/key1
```

```
[key mykey2]
key_location=/path/to/key2
```

File: `$HOME/.starcluster/clusters`

```
[cluster smallcluster]
cluster_size = 5
keyname = mykey1
node_image_id = ami-999999999
```

```
[cluster largecluster]
extends = smallcluster
cluster_size = 50
node_image_id = ami-888888888
```

File: `$HOME/.starcluster/vols`

```
[key mykey]
key_location=/path/to/key
```

Then define the files in the config using the *include* setting in the **[global]** section of the default StarCluster config (`~/.starcluster/config`):

```
[global]
include = ~/.starcluster/awskeys, ~/.starcluster/clusters, ~/.starcluster/vols
```

Loading Configs from the Web

The files in the above example could also be loaded from the web. Let's say we've hosted, for example, the cluster templates in `~/.starcluster/clusters` on an http server at the url: `http://myhost/cluster.cfg`. To load these cluster templates from the web we just add the web address(es) to the list of includes:

```
[global]
include = ~/.starcluster/keys, http://myhost/cluster.cfg, ~/.starcluster/vols
```

Notice in the above example we only load the cluster templates from the web. The aws credentials, keypairs, volumes, etc. will all be loaded locally in this case.

StarCluster also supports loading the default config containing the **[global]** section from the web:

```
$ starcluster -c http://myhost/sc.cfg listvolumes
```

If you choose to load the default config from the web it's recommended that only a **[global]** section is defined that includes configs either locally, from the web, or both. It's also important

Tips for Switching Regions

Note: All examples in this section use `us-west-1` as the *target* region. You should replace `us-west-1` in these examples with your target region. Also, you do not need to pass the global `--region (-r)` flag if you've configured your **[aws info]** section to permanently use the target region.

In general, keypairs, AMIs, and EBS Volumes are all region-specific and must be recreated or migrated before you can use them in an alternate region. To create a new keypair in the target region, use the **createkey** command while passing the global `--region (-r)` flag:

```
$ starcluster -r us-west-1 createkey -o ~/.ssh/uswestkey.rsa myuswestkey
```

The above example creates a new keypair called *myuswestkey* in the `us-west-1` region and stores the key file in `~/.ssh/uswestkey.rsa`. Once you've created a new keypair in the target region you must define the new keypair in the config. For the above `us-west-1` example:

```
[key myuswestkey]
KEY_LOCATION = ~/.ssh/uswestkey.rsa
```

Similarly you can obtain a list of available StarCluster AMIs in the target region using:

```
$ starcluster -r us-west-1 listpublic
```

Finally, to (optionally) create new EBS volumes in the target region:

```
$ starcluster -r us-west-1 createvolume -n myuswestvol 10 us-west-1a
```

Given that a *cluster template* references these region-specific items you must either override the relevant settings at the command line using the *start* command's option flags or create separate *cluster templates* configured for each region you use. To override the relevant settings at the command line:

```
$ starcluster -r us-west-1 start -k myuswestkey -n ami-99999999
```

If you often use multiple regions you will most likely want to create separate *cluster templates* for each region by extending a common template, *smallcluster* for example, and overriding the relevant settings:

```
[key myuswestkey]
KEY_LOCATION = ~/.ssh/uswestkey.rsa

[volume myuswestvol]
VOLUME_ID = vol-99999999
MOUNT_PATH = /data

[cluster uswest-cluster]
EXTENDS = smallcluster
KEYNAME = uswestkey
# The AMI must live in the target region!
```

```
NODE_IMAGE_ID = ami-99999999
VOLUMES = myuswestvol
```

The above example extends the default cluster template *smallcluster* and overrides the relevant settings needed for the target region.

With the above template defined you can use the *start* command's *-c* (*-cluster-template*) option to use the new region-specific template to easily create a new cluster in the target region:

```
$ starcluster -r us-west-1 start -c uswest-cluster mywestcluster
```

1.4.2 Using EBS Volumes for Persistent Storage

StarCluster supports using Amazon's Elastic Block Storage (EBS) volumes for persistent storage. These volumes can be anywhere from 1GB to 1TB in size. StarCluster will attach each volume specified in a cluster template to the master node and then share the volume(s) to the rest of the nodes in the cluster via the network file system (NFS). Each volume will be mounted to the path specified by the `MOUNT_PATH` setting in the volume's configuration section.

For example, suppose we have the following configuration defined:

```
[vol myvol]
volume_id = vol-v99999999
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-v99999999` to the master node on `/data` and then NFS-share `/data` to all the nodes in the cluster.

It's also possible to use multiple EBS volumes by specifying a list of volumes in a cluster template:

Note: Each volume specified in a cluster template *must* have a unique `MOUNT_PATH` otherwise an error will be raised.

```
[vol cancerdata]
volume_id = vol-v88888888
mount_path = /data/cancer

[vol genomedata]
volume_id = vol-v99999999
mount_path = /data/genome

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

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.

Note: The **createvolume** command simply formats the *entire volume* using all of the space on the device rather than creating partitions. This makes it easier to resize the volume and expand the filesystem later on if you run out of disk space.

To create and format a new volume simply specify a volume size in GB and the availability zone to create the volume in:

```
$ starcluster createvolume --name=my-data 20 us-east-1c
```

The above 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. The `--name` option allows you name the volume for easy reference later on when using the **listvolumes** command or the [AWS web console](#).

If you wish to apply an arbitrary tag to the new volume use the `--tag` option:

```
$ starcluster createvolume --tag=mytag 20 us-east-1c
```

If you want to create a key/value tag:

```
$ starcluster createvolume --tag mytag=myvalue 20 us-east-1c
```

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: StarCluster does not terminate the host instance after creating a volume. This allows multiple volumes to be created in the same zone using a *single* host instance. You can pass the `--shutdown-volume-host` option to the **createvolume** command to if you'd rather automatically shutdown the volume host after creating the new volume.

Let's look at an example of creating a 20GB volume in us-east-1c:

```
$ starcluster createvolume --name=myvol --bid=0.50 20 us-east-1c
StarCluster - (http://star.mit.edu/cluster)
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
```

In the above example we name the volume `myvol` and use a spot instance for the volume host. 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://star.mit.edu/cluster)
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

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

```
volume_id: vol-be279s08
size: 5GB
status: available
availability_zone: us-east-1d
create_time: 2011-10-22 16:18:57
```

```
Total: 1
```

To list details for a single volume by name use the `--name (-n)` option:

```
$ starcluster listvolumes --name my-big-data
```

To list details for a single volume by id use the `--volume-id (-v)`:

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

If you'd like to see details for all volumes with a given tag use the `--tag (-t)` option:

```
$ starcluster listvolumes -t my-big-data
$ starcluster listvolumes -t mytag=myvalue
```

You can also filter the volumes by status using the `--status (-S)` flag:

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

and by volume size (in GB) using the `--size (-s)` option:

```
$ starcluster listvolumes -s 20
```

and also by attachment state using the `--attach-status (-a)` option:

```
$ 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-999999999
```

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-999999999`, to 20GB:

```
$ starcluster resizevolume vol-999999999 20
```

The above command will create a *new*, larger, 20GB volume containing the data from the original volume `vol-999999999`. 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-999999999 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 `--zone (-z)`, flag:

```
$ starcluster resizevolume -z us-east-1d vol-999999999 20
```

The above command will create a new 20GB volume in `us-east-1d` containing the data in `vol-999999999`. 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 Cluster 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://star.mit.edu/cluster)
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://star.mit.edu/cluster)
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
```

You can login as a different user using the `--user (-u)` option. For example, to login as the `sgadmin` user:

```
$ starcluster sshmaster -u sgadmin physicscluster
```

Logging into the worker nodes

To login to a worker node, say `node001` for example, as root:

```
$ starcluster sshnode physicscluster node001
```

You can also login as a different user using the `--user (-u)` option. For example, to login as the `sgadmin` user:

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

Running X11 (Graphical) Applications on the Cluster

If you have OpenSSH installed and an X server you can enable X11 forwarding over SSH using the `--forward-x11 (-X)` option. This allows you to run graphical applications on the cluster and display them on your local computer:

```
$ starcluster sshmaster -X mycluster
```

When you login you should be able to run graphical applications, for example `xterm`, on the cluster and display them on your local computer. The `sshnode` command also supports the `-X` option:

```
$ starcluster sshnode -X mycluster node001
```

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.

Terminating a Cluster

Warning: Once a cluster has been terminated, any data that was not saved either to S3 or an external EBS volume will be lost. Make sure you save any data you care to keep to S3 or to an external EBS volume.

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://star.mit.edu/cluster)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

Terminate cluster physicscluster (y/n)? y
>>> Shutting down i-999999999
>>> Shutting down i-888888888
>>> Waiting for cluster to terminate...
>>> Removing cluster security group @sc-physicscluster
```

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

Stopping an EBS-backed Cluster

Note: You can not stop S3-backed clusters - they can only be terminated.

If you used EBS-backed AMIs when creating a cluster, the cluster can optionally be stopped instead of terminated:

```
$ starcluster stop myebcluster
```

This will shutdown the entire cluster by putting all instances in a stopped state rather than terminating them. This allows you to preserve the local root volumes backing the nodes which would normally be destroyed if you used the **terminate** command. The cluster will continue to show up in the output of the **listclusters** command after being stopped, however, you will not be charged for stopped instance hours, only for the EBS volume storage backing the nodes:

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

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

Launch time: 2011-10-22T20:55:20.000Z
Uptime: 00:29:42
Zone: us-east-1c
Keypair: gsg-keypair
Cluster nodes:
    master stopped i-999999999
    node001 stopped i-888888888
Total nodes: 2
```

A stopped EBS-backed cluster can be started later on using:

```
$ starcluster start -x myebcluster
```

This will start all stopped nodes in the cluster and reconfigure the cluster. Once the cluster comes up all data previously stored on the root volumes backing the nodes before shutdown will be available.

To *completely* destroy an EBS-backed cluster use the **terminate** command:

```
$ starcluster terminate myebcluster
```

This will completely destroy the cluster nodes including the root volumes backing the nodes. As always, before terminating the cluster you should move any data you wish to keep either to an external EBS volume or to S3.

1.4.4 Running Remote Commands from Command Line

StarCluster's **sshmaster**, **sshnode**, and **sshinstance** commands now support executing remote commands on a cluster node without logging in interactively. This is especially useful for users looking to script these commands. To do so simply pass the command you wish to run remotely as an additional quoted argument to any of the **sshmaster**, **sshnode**, and **sshinstance** commands.

For example, to check which hosts are listed in the master node's `/etc/hosts` file:

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

10.10.10.1 master
10.10.10.2 node001
10.10.10.3 node002
```

or to quickly check the files in a given directory on node001:

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

drwxr-xr-x 4 root root 4096 Nov  5 00:06 data-2011
drwxr-xr-x 2 root root 4096 Nov  4 14:21 data-2010
drwxr-xr-x 2 root root 4096 Nov  4 14:23 data-2009
drwxr-xr-x 4 root root 4096 Oct 17 13:42 data-2008
```

Any command you can run while logged into a cluster interactively can be executed remotely from your local command line simply by quoting the remote command and passing it as an additional argument to any of the **sshmaster**, **sshnode**, and **sshinstance** commands as in the above examples.

If the remote command is successful the exit code of either the **sshmaster**, **sshnode**, or **sshinstance** command will be 0:

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

00:58:12 up 1 days,  9:49,  2 users,  load average: 0.02, 0.06, 0.12
$ echo $?
0
```

Otherwise, the exit code will be identical to the remote command's non-zero exit code:

```
$ starcluster sshmaster mycluster 'uptimes2'
StarCluster - (http://star.mit.edu/cluster)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

command not found: uptimes2

!!! ERROR - command 'uptimes2' failed with status 127
```

```
$ echo $?  
127
```

This allows you to use the **sshmaster**, **sshnode**, and **sshinstance** commands in scripts and check whether or not the remote command finished successfully.

Running X11 (Graphical) Applications

If you have OpenSSH installed and an X server you can enable X11 forwarding over SSH using the `--forward-x11 (-X)` option. This allows you to run graphical applications on the cluster and display them on your local computer. For example, to run *xterm* on the master node of *mycluster*:

```
$ starcluster sshmaster -X mycluster xterm
```

The **sshnode** command also supports the `-X` option:

```
$ starcluster sshnode -X mycluster node001 xterm
```

1.4.5 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.

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
```

To copy files as a different cluster user, use the `--user (-u)` option:

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

To copy files to a different cluster node, use the `--node (-n)` 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
```

To copy files as a different cluster user, use the `--user (-u)` option:

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

To copy files from a different cluster node, use the `--node (-n)` option:

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

1.4.6 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 *sgedadmin*. If you’ve specified a different **cluster_user**, please replace sgedadmin 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://star.mit.edu/cluster)
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://star.mit.edu/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://star.mit.edu/cluster)
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://star.mit.edu/cluster)
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://star.mit.edu/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 (i.e. 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 master,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:

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

1.4.7 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.

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> -m <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 (**-m <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 as the image host by passing **--bid (-b)** option:

```
$ starcluster start -o -s 1 -b 0.50 -I <INSTANCE-TYPE> -m <BASE-AMI-ID> imagehost
```

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 --show-ssh-status 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.

Note: 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 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 <i>every</i> 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-99999999 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-99999999 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*.

1.4.8 Listing the Official StarCluster AMIs

From time to time new StarCluster AMIs may become available. These AMIs might simply contain new OS versions or could fix a bug experienced by StarCluster users. In any case it's useful to look at the *latest* available StarCluster AMIs.

To look at a list of all currently available public StarCluster AMIs run the **listpublic** command:

```
$ starcluster listpublic
```

This will show all available AMIs in the current region. Unless you've specified a region in your config the default region is *us-east-1*. To view AMIs in other regions either specify a region in your `[aws info]` config or use the global **-r** option:

```
$ starcluster -r eu-west-1 listpublic
```

At the time of writing this document the output for *us-east-1* looks like:

```
$ starcluster listpublic
StarCluster - (http://star.mit.edu/cluster)
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
```

1.4.9 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.

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 constructor 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://star.mit.edu/cluster)
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 ssh
```

```
[~]|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. Test your plugin on your small test cluster using the **runplugin** command:

```
$ starcluster runplugin myplugin testcluster
```

Alternatively, you can also run your plugin using the development shell (requires [IPython](#)):

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

6. Fix any coding errors in order to get the plugin to run from start to finish using the **runplugin** command.
7. Login to the master node and verify that the plugin was successful:

```
$ starcluster sshmaster testcluster
```

1.4.10 Adding and Removing Nodes

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

To add nodes to a running cluster use 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://star.mit.edu/cluster)
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%
>>> 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
```

The `addnode` command auto-generates an alias for the new node(s). In the above example *mycluster* is a single node cluster. In this case `addnode` automatically added a new node and gave it an alias of *node001*. If we added additional nodes they would be named *node002*, *node003*, and so on.

If you'd rather manually specify an alias for the new node(s) use the `--alias (-a)` option:

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

It is also possible to add multiple nodes using the `--num-nodes (-n)` option:

```
$ starcluster addnode -n 5 mycluster
```

The above command will add five additional nodes to *mycluster* auto-generating the node aliases. To specify aliases for all five nodes simply specify a comma separated list to the `-a` option:

```
$ starcluster addnode -n 5 -a n1,n2,n3,n4,n5 mycluster
```

Once the `addnode` command has completed successfully the new nodes will show up in the output of the `listclusters` command:

```
$ starcluster listclusters mycluster
```

You can login directly to a new node by alias:

```
$ starcluster sshnode mycluster mynewnode
```

The `addnode` command has additional options for customizing the new node's instance type, AMI, spot bid, and more. See the help menu for a detailed list of all available options:

```
$ starcluster addnode --help
```

Re-adding a Node

If you've previously attempted to add a node and it failed due to a plugin error or other bug or if you used the `removenode` command with the `-k` option and wish to re-add the node to the cluster without launching a new instance you can use the `-x` option:

```
$ starcluster addnode -x -a node001 mycluster
```

Note: The `-x` option requires the `-a` option

This will attempt to add or re-add `node001` to `mycluster` using the existing instance rather than launching a new instance. If no instance exists with the alias specified by the `-a` option an error is reported. You can also do this for multiple nodes:

```
$ starcluster addnode -x -a mynode1,mynode2,mynode3 mycluster
```

Removing Nodes

To remove nodes from an existing cluster use 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://star.mit.edu/cluster)
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
>>> Canceling spot request sir-3567ba14
>>> Terminating node: node001 (i-8bec7ce5)
```

The above command removes `node001` from `mycluster` by removing the node from the Sun Grid Engine queuing system, from each node's `ssh known_hosts` files, from each node's `/etc/hosts` file, and from all NFS shares. If you're using plugins with your cluster they will be called to remove the node. Once the node has been removed from the cluster 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.

You can also remove multiple nodes by providing a list of aliases:

```
$ starcluster removenode mycluster node001 node002 node003
```

Remove Without Terminating

If you'd rather not terminate the node(s) after removing from the cluster to test plugins, for example, use the `--keep-instance (-k)` option:

```
$ starcluster removenode -k mycluster node001 node002 node003
```

This will remove the nodes from the cluster but leave the instances running. This can be useful, for example, when testing `on_add_node` methods in a StarCluster plugin.

1.4.11 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

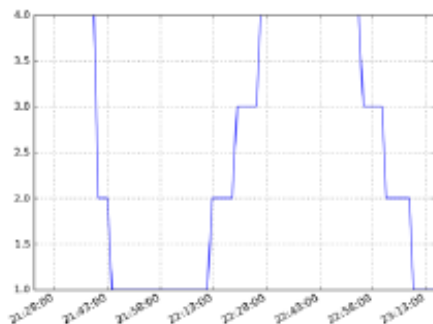
By default the load balancer will add a single node at a time when adding new nodes to the cluster. This allows the load balancer to gradually make a change and observe the impact on the queue without adding excessive resources. However, if you'd like to increase the number of nodes added when the load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

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

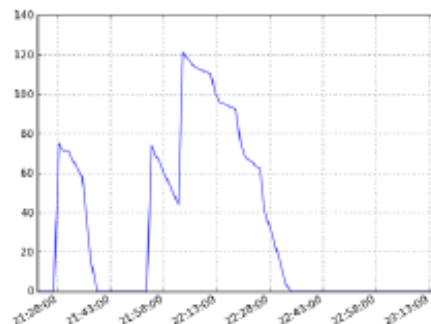
The above command will load balance *mycluster* up to a maximum of twenty nodes by adding two nodes at a time as necessary.

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 (maximum: 300s)
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” (minimum: 300s)
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-cluster`, option:

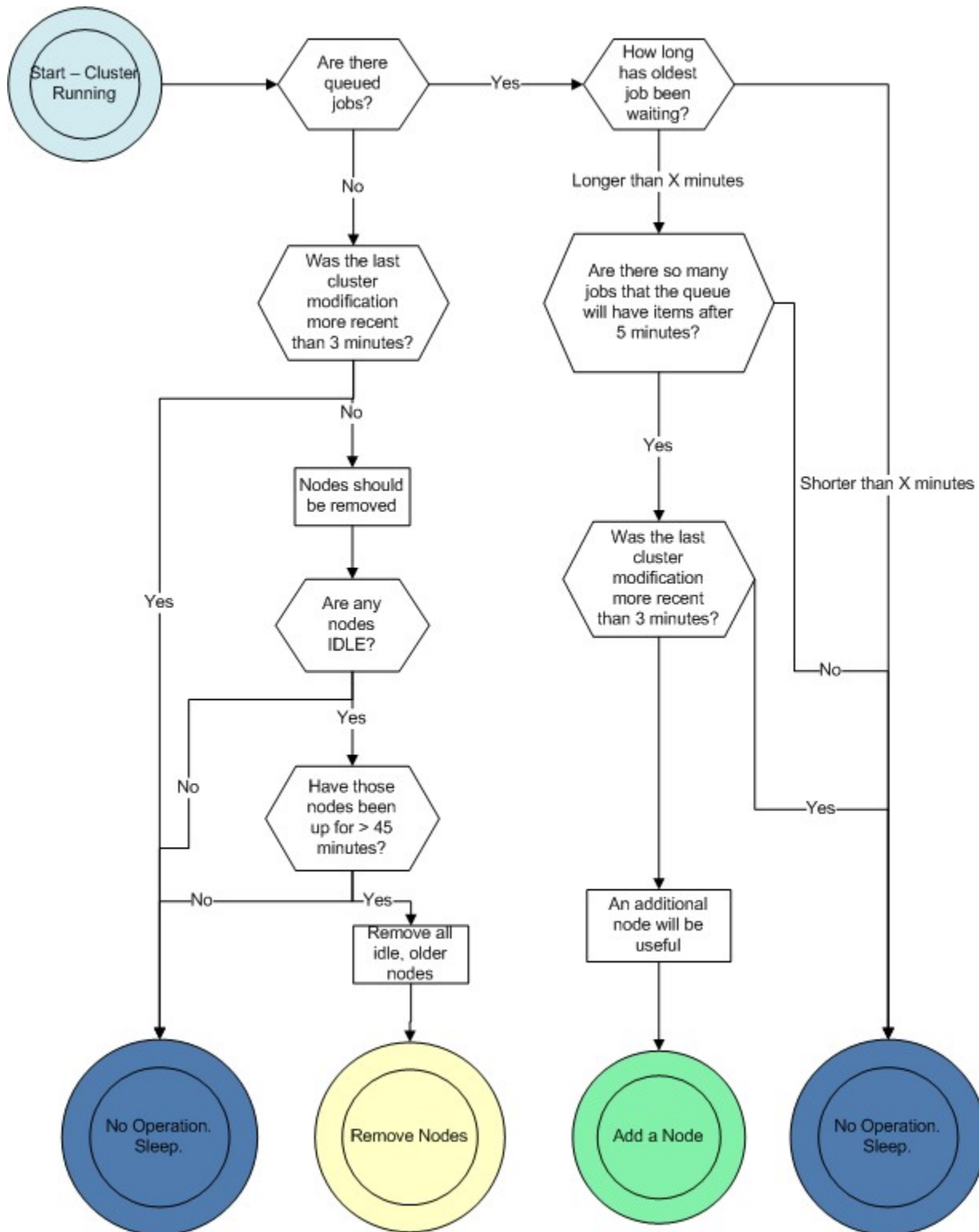
```
$ starcluster loadbalance --kill-cluster mycluster
```

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 cluster will 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 continuously 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 `sgc` 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 execute the `sgc` 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](#).

1.4.12 StarCluster Shell Completion (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 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 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:

```
sgadmin@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*:

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

```
sgeadmin@master:~$ qghost
HOSTNAME ARCH NCPU LOAD MEMTOT MEMUSE SWAPTO SWAPUS
-----
global - - - - -
master 1x24-x86 1 0.00 1.7G 62.7M 896.0M 0.0
node001 1x24-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 Sun Grid Engine Plugin

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.

Using the Plugin

Note: The SGE plugin is enabled by default for all clusters created by StarCluster - no additional configuration is required. The next section is for users that wish to customize the SGE install beyond the defaults.

Advanced Options

The SGE plugin has advanced options that some users may wish to tune for their needs. In order to use these advanced options you must first define the SGE plugin in your config:

```
[plugin sge]
setup_class = starcluster.plugins.sge.SGEPlugin
```

Once the plugin has been defined the next step is to add the plugin to the `plugins` list in one of your cluster templates:

```
[cluster smallcluster]
plugins = sge
```

The final step is to set `disable_queue=True` in your cluster template. This tells StarCluster *not* to run the SGE plugin internally by default. This is needed given that we're configuring the plugin manually ourselves:

Warning: It is important to set `disable_queue=True` when manually configuring the SGE plugin. If you don't the SGE plugin will run **twice** because the SGE plugin is currently a *default* in StarCluster.

```
[cluster smallcluster]
disable_queue = True
plugins = sge
```

Disabling Job Execution on Master Node By default StarCluster configures the master node as an *execution host* which means that the master node can accept and run jobs. In some cases you may not wish to run jobs on the master due to resource constraints. For example, if you're generating a lot of NFS traffic in your jobs you may wish to completely dedicate the master to serving NFS rather than both running jobs and serving NFS.

To disable the master node being used as an *execution host* set `master_is_exec_host=False` in your sge plugin config:

```
[plugin sge]
setup_class = starcluster.plugins.sge.SGEPlugin
master_is_exec_host = False
```

Now whenever a new cluster is created with the SGE plugin enabled the master will *not* be configured as an *execution host*.

Setting the Number of Slots Per Host By default StarCluster configures each execution host in the cluster with a number of job ‘slots’ equal to the number of processors on the host. If you’d like to manually set the number of slots on each execution host set `slots_per_host=<num_slots_per_host>` in your SGE plugin config:

```
[plugin sge]
setup_class = starcluster.plugins.sge.SGEPlugin
slots_per_host = 10
```

Whenever a new cluster is created with the above configuration each execution host in the cluster will be assigned 10 slots.

Disabling SGE

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, if you do not want or need SGE on your cluster simply set `disable_queue=True` in one of your cluster templates:

```
[cluster smallcluster]
disable_queue = True
```

This will skip the SGE install phase when creating a new cluster using the `smallcluster` template.

Sun Grid Engine Quick-Start

The following sections give an overview of how to submit jobs, monitor job and host status, and how to use the SGE parallel environment.

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:

```
sgadmin@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:

```
sgedadmin@master:~$ qhost
HOSTNAME ARCH NCPU LOAD MEMTOT MEMUSE SWAPTO SWAPUS
```

```
-----
global - - - - -
master 1x24-x86 1 0.00 1.7G 62.7M 896.0M 0.0
node001 1x24-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*:

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

```
-----
all.q@master.c BIP 0/0/1 0.00 1x24-x86
-----
all.q@node001.c BIP 0/0/1 0.00 1x24-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/sgedadmin/jobscript.sh on your StarCluster and execute the following as the sgedadmin user:

```
sgedadmin@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:

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

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

sgedadmin@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    $fill_up
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).

Parallel Environment Allocation Rule Notice the *allocation_rule* setting in the output of the *qconf* command in the previous section. This rule defines how to assign *slots* to a job. By default StarCluster uses the *fill_up* allocation rule. This rule causes SGE to greedily take all available slots on as many cluster nodes as needed to fulfill the slot requirements of a given job. For example, if a user requests 8 *slots* and a single node has 8 *slots* available, that job will run entirely on one node. If 5 *slots* are available on one node and 3 on another, it will take all 5 on that node, and all 3 on the other node.

The allocation rule can also be configured to distribute the slots around the cluster as evenly as possible by using the *round_robin* *allocation_rule*. For example, if a job requests 8 *slots*, it will go to the first node, grab a slot if available, move to the next node and grab a single slot if available, and so on wrapping around the cluster nodes again if necessary to allocate 8 *slots* to the job.

Finally, setting the *allocation_rule* to an integer number will cause the parallel environment to take a fixed number of slots from each host when allocating the job by specifying an integer for the *allocation_rule*. For example, if the *allocation_rule* is set to 1 then all slots have to reside on different hosts. If the special value *\$pe_slots* is used then all slots for the parallel job must be allocated entirely on a single host in the cluster.

You can change the allocation rule for the *orte* parallel environment at any time using:

```
$ qconf -mp orte
```

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

```
allocation_rule    $fill_up
```

to:

```
allocation_rule    $round_robin
```

You can also change the rule to the *pe_slots* mode:

```
allocation_rule    $pe_slots
```

or specify a fixed number of slots per host to assign when allocating the job:

```
allocation_rule    1
```

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    $round_robin
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 specifies 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 [...]
```

1.6.2 Create Users Plugin

This plugin creates one or more cluster users and configures passwordless SSH access between cluster nodes for each user.

Configuration

To use the `users` plugin add a **[plugin]** section to your starcluster config file:

```
[plugin createusers]
setup_class = starcluster.plugins.users.CreateUsers
num_users = 30
```

The above config will create and configure 30 cluster users along with passwordless SSH access on the cluster for each user. In this case usernames are auto-generated to be `user001`, `user002`, etc. If you'd prefer to provide the usernames instead of auto-generating them use the following config instead:

```
[plugin createusers]
setup_class = starcluster.plugins.users.CreateUsers
usernames = linus, tux, larry
```

Next update the `PLUGINS` setting of one or more of your cluster templates to include the `createusers` plugin:

```
[cluster mycluster]
plugins = createusers
```

The next time you start a cluster using the `mycluster` template the plugin will automatically be executed and users will be created. If you need to download the SSH keys for each cluster user in order to distribute them to users that do not have access to your AWS account please read the next subsection.

Downloading User SSH Keys

Warning: These settings will download the SSH keys for each user to your local computer. Please be aware that these keys allow remote access to your cluster(s). Please use caution when storing and distributing these keys.

If you'd prefer the plugin to automatically archive and download the SSH keys for each cluster user add the following to your **[plugin]** section:

```
[plugin createusers]
setup_class = starcluster.plugins.users.CreateUsers
num_users = 30
download_keys = True
```

By default this will create and download an archive containing all users' SSH keys to `$HOME/.starcluster/user_keys/<cluster>-<region>.tar.gz`. If you'd prefer to store the archive in an alternate location specify the `download_keys_dir` setting:

```
[plugin createusers]
setup_class = starcluster.plugins.users.CreateUsers
num_users = 30
download_keys = True
download_keys_dir = /path/to/keys/dir/
```

The above example will download the SSH keys archive to `/path/to/keys/dir/<cluster>-<region>.tar.gz`.

Usage

Once the plugin has been configured it will automatically run the next time you start a new cluster using the appropriate cluster template. If you already have a cluster running that didn't originally have the plugin in its plugin list at creation time you will need to manually run the plugin:

```
% starcluster runplugin createusers mycluster
StarCluster - (http://star.mit.edu/cluster)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

>>> Running plugin createusers
>>> Creating 30 cluster users
3/3 ||||| 100%
>>> Configuring passwordless ssh for 30 cluster users
50/50 ||||| 100%
>>> Configuring scratch space for user(s): user001, user002,
>>> user003, user004, user005, user006, user007, user008,
>>> user009, user010, user011, user012, user013, user014,
>>> user015, user016, user017, user018, user019, user020,
>>> user021, user022, user023, user024, user025, user026,
>>> user027, user028, user029, user030
3/3 ||||| 100%
```

After the plugin has finished executing you can easily login to the cluster as any one of these users on any node in the cluster:

```
% starcluster sshmaster -u user007 mycluster
% starcluster sshnode -u user007 mycluster node007
```

Using the SSH Keys Archive

Warning: Please use caution when storing and distributing these keys - they allow remote access to your cluster.

If you specified `download_keys = True` in your config then the plugin will create and download a gzipped tar archive containing the RSA SSH keys for each user to your local computer:

```
>>> Tarring all SSH keys for cluster users...
>>> Copying cluster users SSH keys to: mycluster-us-east-1.tar.gz
mycluster-us-east-1.tar.gz 100% ||||| Time: 00:00:00 963.07 K/s
```

If you did not specify `download_keys_dir` in your config then the tar archive will be saved to `$HOME/.starcluster/user_keys/<cluster>-<region>.tar.gz` by default. The archive contains all of the RSA SSH keys for each cluster user it created:

```
% cd $HOME/.starcluster/user_keys
% tar -tf mycluster-us-east-1.tar.gz
./user001.rsa
./user002.rsa
./user003.rsa
./user004.rsa
./user005.rsa
./user006.rsa
./user007.rsa
./user008.rsa
./user009.rsa
./user010.rsa
./user011.rsa
./user012.rsa
./user013.rsa
./user014.rsa
./user015.rsa
...
```

These keys can be distributed to non-AWS users to allow remote cluster access *without AWS credentials*. You will also need to pick a cluster node for users to login to and then distribute that node's public DNS name to your non-AWS users. You can use the **listclusters** command to list the public DNS names of all nodes in your cluster:

```
$ starcluster listclusters mycluster
StarCluster - (http://star.mit.edu/cluster)
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
Uptime: 00:29:42
Zone: us-east-1c
Keypair: mykeypair
Cluster nodes:
    master running i-999999999 ec2-123-123-123-121.compute-1.amazonaws.com
    node001 running i-888888888 ec2-123-123-123-122.compute-1.amazonaws.com
Total nodes: 2
```

As an example, let's choose the `master` node in the above output to be the 'login' node. Now suppose we distribute `user007`'s username and SSH key to a non-AWS user and also give them the public DNS name of the login node. In this case the non-AWS user can connect to the cluster's master node as `user007` using:

```
% ssh -i /path/to/user007.rsa user007@ec2-123-123-123-121.compute-1.amazonaws.com
```

1.6.3 IPython Cluster Plugin

Note: These docs are for [IPython 0.13+](#) which is installed in the latest StarCluster 12.04 Ubuntu-based AMIs. See `starcluster listpublic` for a list of available AMIs.

To configure your cluster as an [interactive IPython cluster](#) you must first define the `ipcluster` plugin in your config file:

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

If you'd like to use the new IPython web notebook (highly recommended!) you'll also want to add the following settings:

```
[plugin ipcluster]
setup_class = starcluster.plugins.ipcluster.IPCluster
enable_notebook = True
notebook_directory = notebooks
# set a password for the notebook for increased security
notebook_passwd = a-secret-password
```

After defining the plugin in your config, add the ipcluster plugin to the list of plugins in one of your cluster templates:

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

Using the IPython Cluster

To use your new IPython cluster log in directly to the master node of the cluster as the `CLUSTER_USER` and create a parallel client:

```
$ starcluster sshmaster mycluster -u myuser
$ ipython
[~]> from IPython.parallel import Client
[~]> rc = Client()
```

Once the client has been started, create a 'view' over the entire cluster and begin running parallel tasks. Below is an example of performing a parallel map across all nodes in the cluster:

```
[~]> view = rc[:]
[~]> results = view.map_async(lambda x: x**30, range(8))
[~]> print results.get()
[0,
 1,
1073741824,
205891132094649L,
1152921504606846976L,
931322574615478515625L,
221073919720733357899776L,
22539340290692258087863249L]
```

See also:

See the [IPython parallel docs](#) (0.13+) to learn more about the IPython parallel API

Connecting from your Local IPython Installation

Note: You must have IPython 0.13+ installed to use this feature

If you'd rather control the cluster from your *local* IPython installation use the `shell` command and pass the `--ipcluster` option:

```
$ starcluster shell --ipcluster=mycluster
```

This will start StarCluster's development shell and configure a remote parallel session for you automatically. StarCluster will create a parallel client in a variable named `ipclient` and a corresponding view of the entire cluster in a variable named `ipview` which you can use to run parallel tasks on the remote cluster:

```
$ starcluster shell --ipcluster=mycluster
[~]> ipclient.ids
[0, 1, 2, 3]
[~]> res = ipview.map_async(lambda x: x**30, range(8))
[~]> print res.get()
```

Using IPython Parallel Scripts with StarCluster

If you wish to run parallel IPython scripts from your local machine that run on the remote cluster you will need to use the following configuration when creating the parallel client in your code:

```
from IPython.parallel import Client
rc = Client('~/.starcluster/ipcluster/<cluster>-<region>.json'
           sshkey='/path/to/cluster/keypair.rsa')
```

For example, let's say we started a cluster called 'mycluster' in region 'us-east-1' with keypair 'mykey' stored in /home/user/.ssh/mykey.rsa. In this case the above config should be updated to:

```
from IPython.parallel import Client
rc = Client('/home/user/.starcluster/ipcluster/mycluster-us-east-1.json'
           sshkey='/home/user/.ssh/mykey.rsa')
```

Note: it is possible to dynamically add new nodes with the `starcluster addnode` command to a pre-existing cluster. New IPython engines will automatically be started and connected to the controller process running on master. This means that existing `Client` and `LoadBalancedView` instance will automatically be able to leverage the new computing resources to speed-up ongoing computation.

Configuring a custom packer

The default message packer for `IPython.parallel` is based on the JSON format which is quite slow but will work out of the box. It is possible to instead configure the faster 'pickle' packer:

```
[plugin ipcluster]
setup_class = starcluster.plugins.ipcluster.IPCluster
enable_notebook = True
notebook_directory = notebooks
# set a password for the notebook for increased security
notebook_passwd = a-secret-password
packer = pickle
```

When using IPython 0.13 this will require to pass an additional `packer='pickle'`. For instance if running the client directly from the master node:

```
$ starcluster sshmaster mycluster -u myuser
$ ipython
[~]> from IPython.parallel import Client
[~]> rc = Client(packer='pickle')
```

If the `msgpack-python` package is installed on all the cluster nodes and on the client, it is possible to get even faster serialization of the messages with:

```
[plugin ipcluster]
setup_class = starcluster.plugins.ipcluster.IPCluster
enable_notebook = True
notebook_directory = notebooks
# set a password for the notebook for increased security
```

```
notebook_passwd = a-secret-password
packer = msgpack
```

And then from the client:

```
$ starcluster sshmaster mycluster -u myuser
$ ipython
[~]> from IPython.parallel import Client
[~]> rc = Client(packer='msgpack.packb', unpacker='msgpack.unpackb')
```

Note: from IPython 0.14 and on the client will automatically fetch the packer configuration from the controller configuration without passing an additional constructor argument to the `Client` class.

Restarting All the Engines at Once

Sometimes some IPython engine processes become unstable (non-interruptable, long running computation or memory leaks in compiled extension code for instance).

In such a case it is possible to kill all running engine processes and start new ones automatically connected to the existing controller by adding a some configuration for the the `IPClusterRestartEngines` plugin in your `.starcluster/config` file:

```
[plugin ipclusterrestart]
SETUP_CLASS = starcluster.plugins.ipcluster.IPClusterRestartEngines
```

You can then trigger the restart manually using:

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

>>> Running plugin ipclusterrestart
>>> Restarting 23 engines on 3 nodes
3/3 ||||| 100%
```

Using the IPython HTML Notebook

The IPython cluster plugin comes with support for the new [IPython web notebook](#). As mentioned in the intro section, you will need to specify a few extra settings in the IPython cluster plugin's config in order to use the web notebook:

```
[plugin ipcluster]
setup_class = starcluster.plugins.ipcluster.IPCluster
enable_notebook = True
notebook_directory = notebooks
# set a password for the notebook for increased security
notebook_passwd = a-secret-password
```

The `notebook_passwd` setting specifies the password to set on the remote IPython notebook server. If you do not specify the `notebook_passwd` setting the plugin will randomly generate a password for you. You will be required to enter this password in order to login and use the notebook server on the cluster. In addition to enforcing a notebook password, StarCluster also enables SSL in the notebook server in order to secure the transmission of your password when logging in.

The `notebook_directory` setting makes it possible to use a custom folder on the master node. The path can be relative to the user home folder or be absolute. If left blank, the notebooks are stored directly in the home folder. If `notebook_directory` does not exist it automatically created at cluster start-up time.

Once you have these settings in the plugin's config simply start a cluster and let the plugin configure your IPython cluster:

```
$ starcluster start -s 3 iptest
StarCluster - (http://star.mit.edu/cluster)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

... (abbreviated output)
>>> Running plugin ipcluster
>>> Writing IPython cluster config files
>>> Starting the IPython controller and 7 engines on master
>>> Waiting for JSON connector file...
/home/user/.starcluster/ipcluster/SecurityGroup:@sc-iptest-us-east-1.json 100% || Time: 00:00:00 37
>>> Authorizing tcp ports [1000-65535] on 0.0.0.0/0 for: IPython controller
>>> Adding 16 engines on 2 nodes
2/2 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Setting up IPython web notebook for user: myuser
>>> Creating SSL certificate for user myuser
>>> Authorizing tcp ports [8888-8888] on 0.0.0.0/0 for: notebook
>>> IPython notebook URL: https://ec2-184-72-131-236.compute-1.amazonaws.com:8888
>>> The notebook password is: XXXXXXXXXXXX
*** WARNING - Please check your local firewall settings if you're having
*** WARNING - issues connecting to the IPython notebook
>>> IPCluster has been started on SecurityGroup:@sc-iptest for user 'myuser'
with 23 engines on 3 nodes.
```

To connect to cluster from your local machine use:

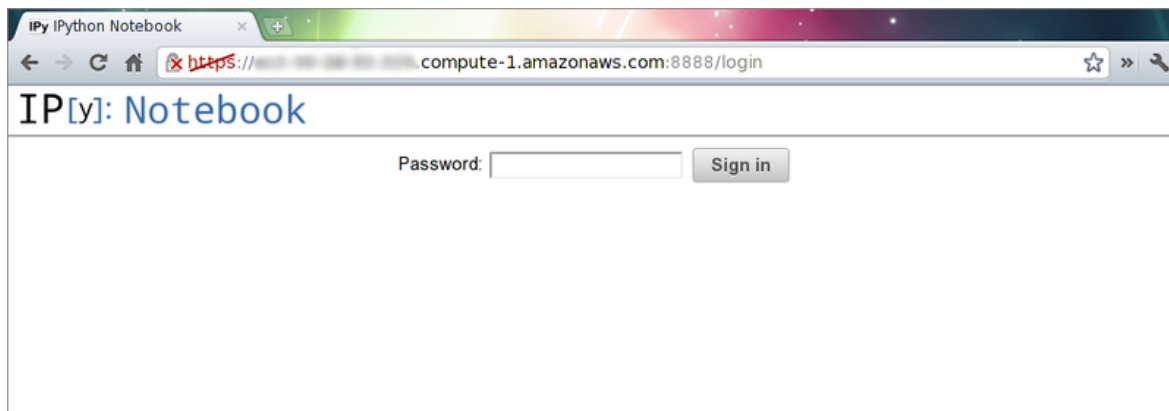
```
from IPython.parallel import Client
client = Client('/home/user/.starcluster/ipcluster/SecurityGroup:@sc-iptest-us-east-1.json', sshkey=

See the IPCluster plugin doc for usage details:
http://star.mit.edu/cluster/docs/latest/plugins/ipython.html
>>> IPCluster took 0.738 mins
```

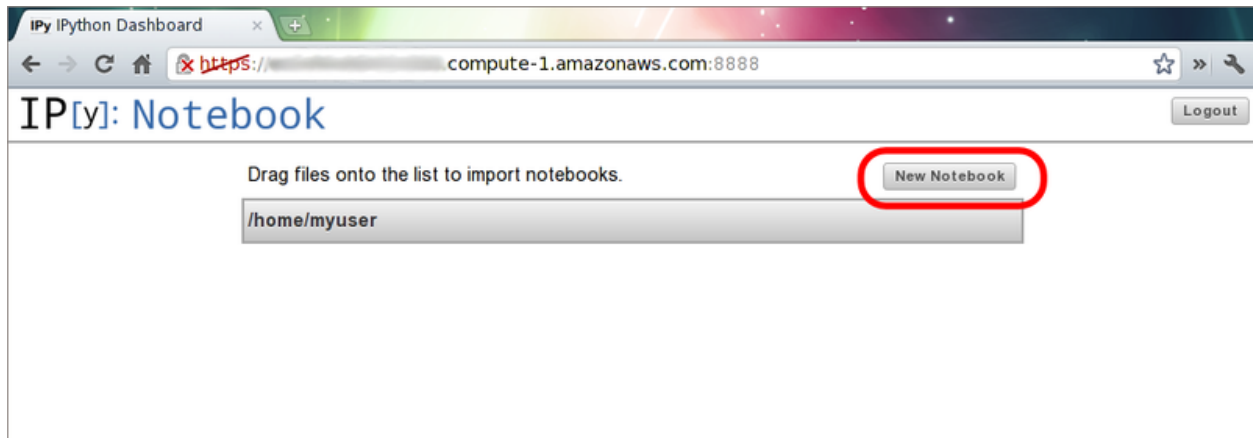
Pay special attention to the following two lines as you'll need them to login to the cluster's IPython notebook server from your web browser:

```
>>> IPython notebook URL: https://ec2-XXXX.compute-1.amazonaws.com:8888
>>> The notebook password is: XXXXXXXXXXXX
```

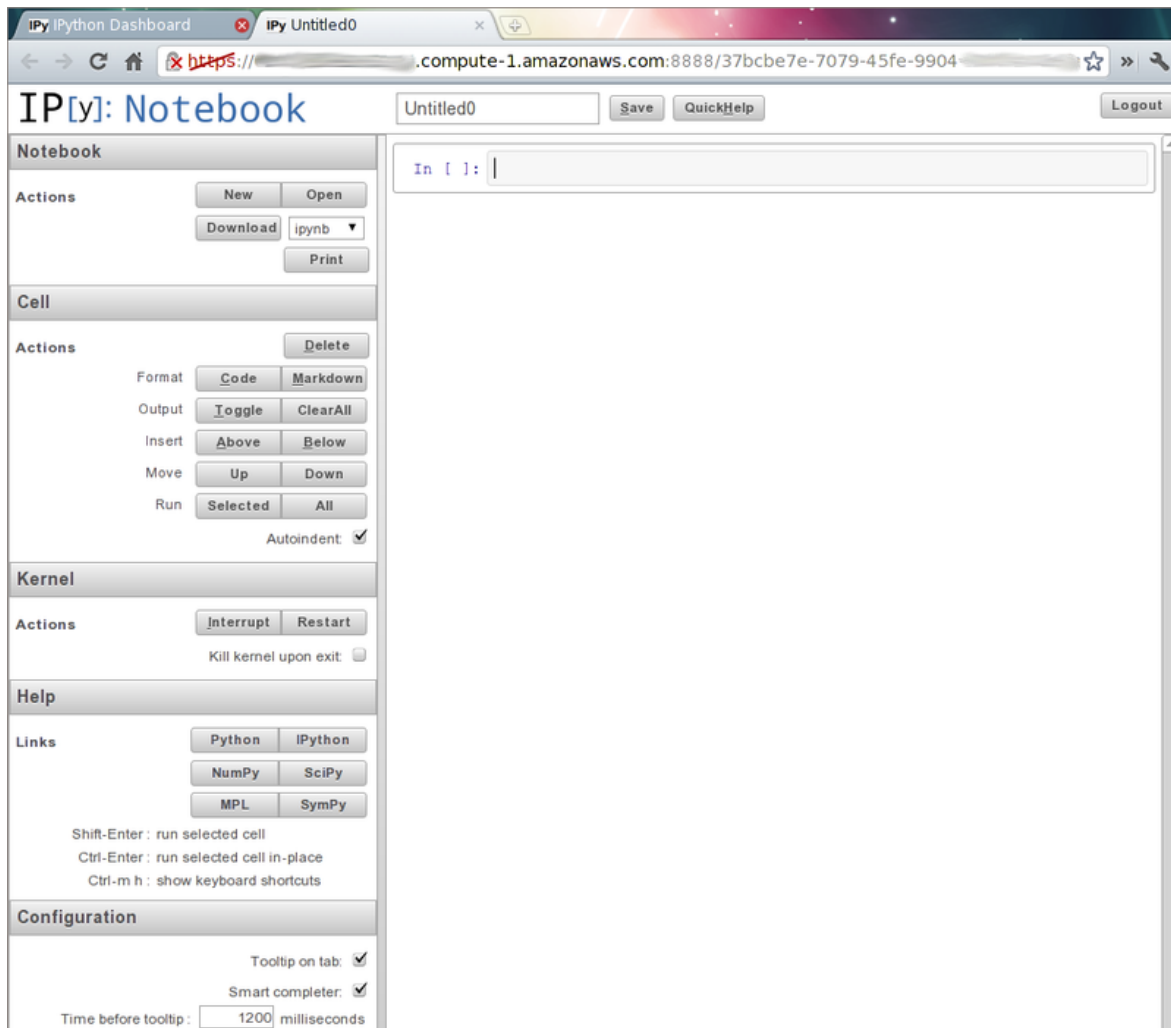
Navigate to the given *https* address and use the password to login:



After you've logged in you should be looking at IPython's dashboard page:

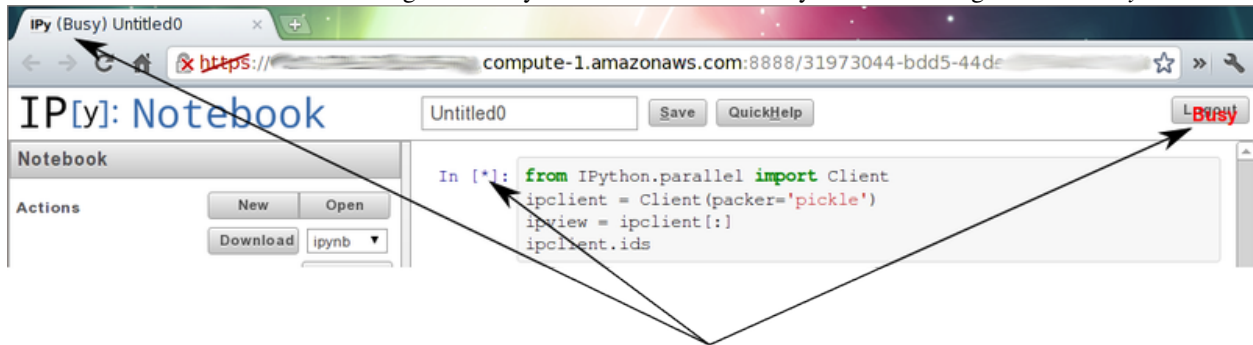


Since this is a brand new cluster there aren't any existing IPython notebook's to play with. Click the New Notebook button to create a new IPython notebook:



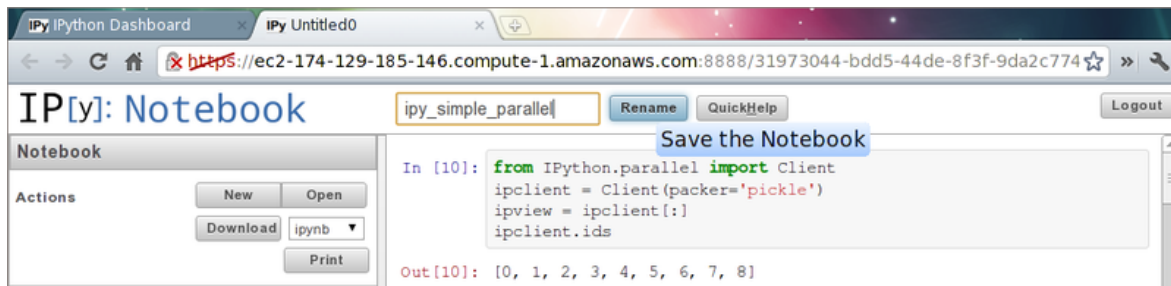
This will create a new blank IPython notebook. To begin using the notebook, click inside the first input cell and begin typing some Python code. You can enter multiple lines of code in one cell if you like. When you're ready to execute your code press `shift-enter`. This will execute the code in the current cell and show any output in a new *output* cell below.

You can modify existing cells simply by clicking in the cell, changing some text, and pressing `shift-enter` again to re-run the cell. While a cell is being executed you will notice that the IPython notebook goes into a *busy* mode:

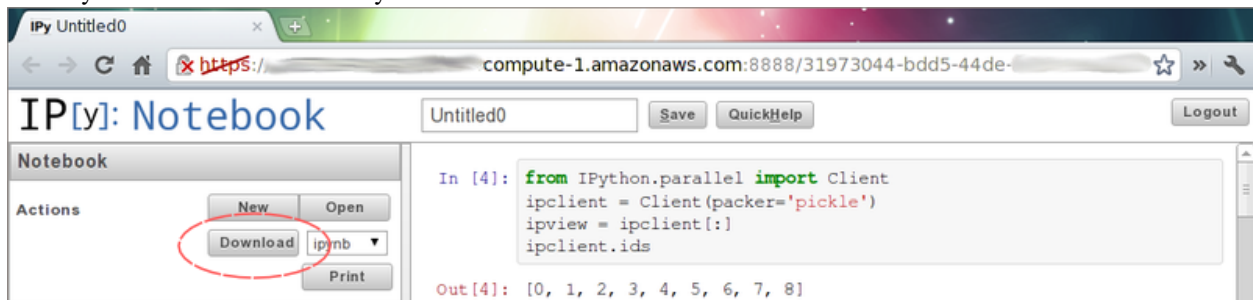


You can keep adding and executing more cells to the notebook while in *busy* mode, however, the cells will run in the order they were executed one after the other. Only one cell can be running at a time.

Once you've finished adding content to your notebook you can save your work to the cluster by pressing the `save` button. Since this is a new notebook you should also change the name before saving which will temporarily change the `save` button to `rename`:



This will save the notebook to `<notebook title>.ipynb` in your `CLUSTER_USER`'s home folder. If you've configured StarCluster to mount an EBS volume on `/home` then these notebook files will automatically be saved to the EBS volume when the cluster shuts down. If this is not the case you will want to download the notebook files before you terminate the cluster if you wish to save them:



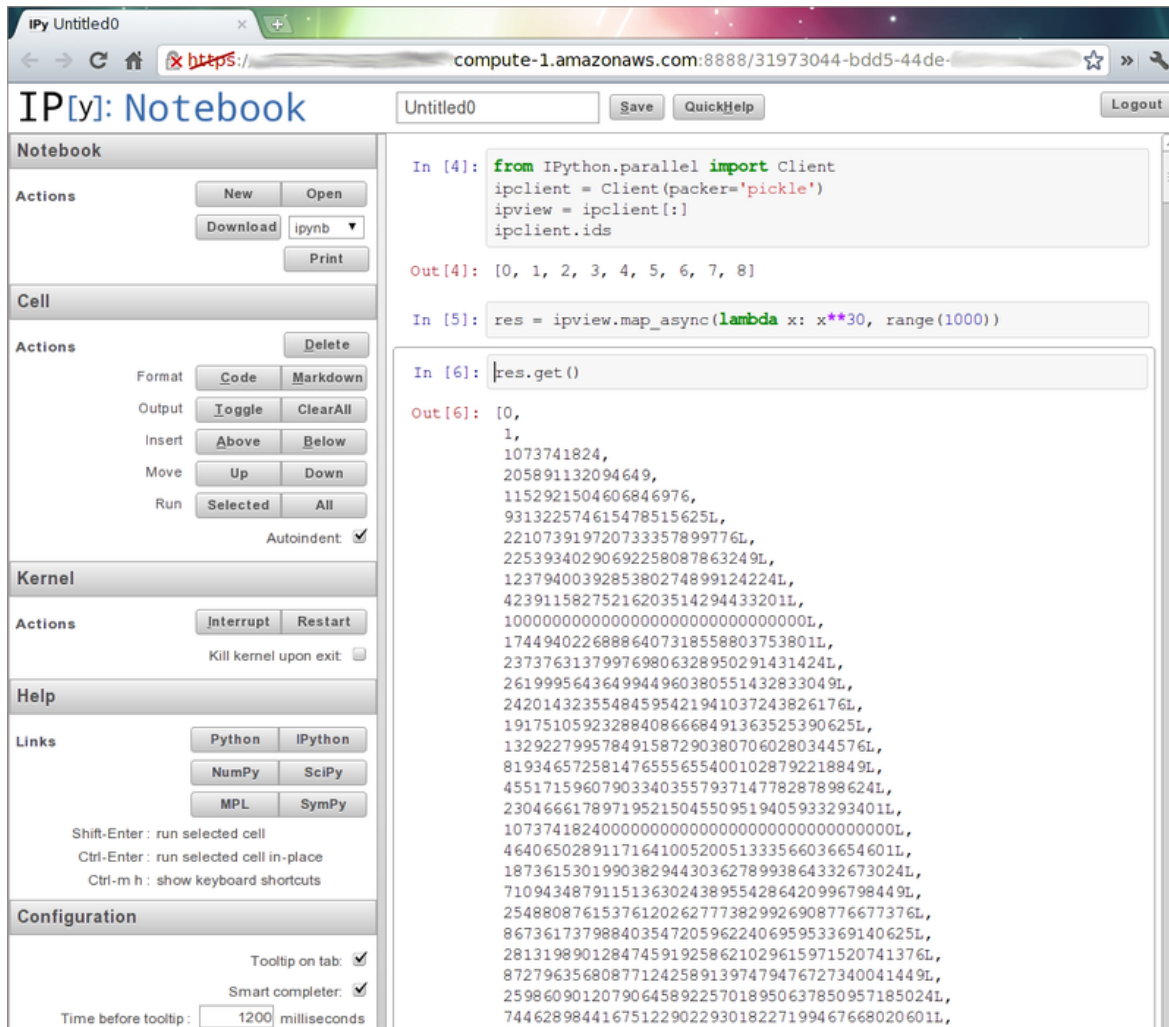
Press `ctrl-m h` within the web notebook to see all available keyboard shortcuts and commands

See also:

See the [official IPython notebook docs](#) for more details on using the IPython notebook

Using Parallel IPython in the IPython Notebook

It's also very easy to combine the notebook with IPython's parallel framework running on StarCluster to create an HPC-powered notebook. Simply use the same commands described in the [Using the IPython Cluster](#) section to set up a parallel client and view in the notebook:



1.6.4 Boto Plugin

This plugin configures a `$HOME/.boto` config file for the `CLUSTER_USER`. It supports both copying an existing boto config file and auto-generating a boto config file using the currently active StarCluster credentials.

To use this plugin you must first define it in the StarCluster config file:

```
[plugin boto]
setup_class = starcluster.plugins.boto.BotoPlugin
```

By default the plugin auto-generates a boto config file for you using the currently active AWS credentials at the time the plugin is executed. If you'd prefer to copy your own boto config file instead add the `boto_cfg` setting:

```
[plugin boto]
setup_class = starcluster.plugins.boto.BotoPlugin
boto_cfg = /path/to/your/boto/config
```

Once you've defined the plugin in your config, add the boto plugin to the list of plugins in one of your cluster templates:

```
[cluster smallcluster]
plugins = boto
```

Now whenever you start a cluster using that cluster template the `CLUSTER_USER` will automatically be configured to use boto.

1.6.5 TMUX Plugin

TMUX is a terminal multiplexer. It enables a number of terminals (or windows), each running a separate program, to be created, accessed, and controlled from a single screen. tmux may be detached from a screen and continue running in the background, then later reattached.

TMUX is especially useful when used on remote systems given that it will automatically detach and background itself if the remote SSH connection breaks saving your remote terminal sessions and running programs. You can then reconnect later on and reattach your terminals and running programs.

This plugin will configure a TMUX session for your `CLUSTER_USER` that contains a terminal session on each node in a separate TMUX window.

Usage

To use the TMUX plugin add a plugin section to your starcluster config file:

```
[plugin tmux]
setup_class = starcluster.plugins.tmux.TmuxControlCenter
```

Next update the `PLUGINS` setting in one or more of your cluster templates to include the TMUX plugin:

```
[cluster mycluster]
plugins = tmux
```

The next time you start a cluster the TMUX plugin will automatically be executed. If you already have a cluster running that didn't originally have tmux in its plugin list you can manually run the plugin using:

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

>>> Running plugin tmux
>>> Starting TMUX Control Center...
>>> Creating TMUX Control Center for user 'myuser'
>>> Creating TMUX Control Center for user 'root'
```

Next login as root or `CLUSTER_USER` and use your cluster TMUX session:

```
$ starcluster sshmaster mycluster -u myuser
myuser@master $ tmux a
```

This will attach your terminal to the remote TMUX session. It should look something like this:

```

myuser@master:~$
myuser@node001:~$

Open Grid Scheduler/Condor cheat sheet:

* qstat/condor_q - show status of batch jobs
* qhost/condor_status - show status of hosts, queues, and jobs
* qsub/condor_submit - submit batch jobs (e.g. qsub -cwd ./jobscript.sh)
* qdel/condor_rm - delete batch jobs (e.g. qdel 7)
* qconf - configure Open Grid Scheduler system

Current System Stats:

System load: 0.0      Processes: 78
Usage of /: 31.4% of 9.84GB  Users logged in: 2
Memory usage: 26%      IP address for eth0: 10.243.115.201
Swap usage: 0%

myuser@node002:~$
[starclust 0:all0* 1:master 2:node001 3:node002- "master" 17:22 27-Dec-11]

```

You'll notice at the bottom there are 4 windows open. Since this is a 3-node cluster the first window is split into 3 panes with each pane logged into a separate node. The remaining 3 windows are all individual SSH sessions to each node in the cluster. If the cluster is large, you will notice multiple `all` windows each containing a group of panes logged into a subset of the cluster nodes. This avoids the `all` windows being split so much that the terminals are unusable. Read the next section to learn how to use navigate these TMUX windows/panes.

Basic TMUX Usage

In order to take full advantage of TMUX you need to become familiar with some of the basic keyboard shortcut commands. It is highly recommended that you read this [blog post](#) for an excellent tutorial on how to use TMUX. Below are the basic keyboard shortcuts for reference:

Note: If you're used to screen or don't care for prefixing commands with `Ctrl-b` you can change this by putting the following in your `$HOME/.tmux.conf`:

```
set -g prefix Ctrl-a
```

You can then replace `Ctrl-b` with `Ctrl-a` in the shortcuts listed below.

Command	Description
Ctrl-b c	Create new window
Ctrl-b d	Detach current client
Ctrl-b l	Move to previously selected window
Ctrl-b n	Move to the next window
Ctrl-b p	Move to the previous window
Ctrl-b &	Kill the current window
Ctrl-b ,	Rename the current window
Ctrl-b %	Split the current window into two panes
Ctrl-b q	Show pane numbers (used to switch between panes)
Ctrl-b o	Switch to the next pane
Ctrl-b ?	List all keybindings

See also:

See the official [TMUX documentation](#) for more details.

1.6.6 Condor Plugin

Note: Condor is only available on the latest StarCluster 11.10 Ubuntu-based AMIs and above. See *starcluster listpublic* for a list of available AMIs.

To configure a condor pool on your cluster you must first define the `condor` plugin in your config file:

```
[plugin condor]
setup_class = starcluster.plugins.condor.CondorPlugin
```

After defining the plugin in your config, add the `condor` plugin to the list of plugins in one of your cluster templates:

```
[cluster smallcluster]
plugins = condor
```

Using the Condor Cluster

Condor jobs cannot be submitted by the `root` user. Instead you must login to the cluster as the normal `CLUSTER_USER`:

```
$ starcluster sshmaster mycluster -u myuser
```

Submitting Jobs

Warning: The “parallel” universe currently does not work. This should be resolved in a future release.

To submit a job you must first create a job script. Below is a simple example that submits a job which sleeps for 5 minutes:

```
Universe      = vanilla
Executable   = /bin/sleep
Arguments     = 300
Log           = sleep.log
Output        = sleep.out
Error         = sleep.error
Queue
```

The above job will run `/bin/sleep` passing 300 as the first argument. Condor messages will be logged to `$PWD/sleep.log` and the job’s standard output and standard error will be saved to `$PWD/sleep.out` and `$PWD/sleep.error` respectively where `$PWD` is the directory from which the job was originally submitted. Save the job script to a file, say `job.txt`, and use the `condor_submit` command to submit the job:

```
$ condor_submit job.txt
Submitting job(s).
1 job(s) submitted to cluster 8.
```

From the output above we see the job has been submitted to the cluster as job 8. Let’s submit this job once more in order to test that multiple jobs can be successfully distributed across the cluster by Condor:

```
$ condor_submit job.txt
Submitting job(s).
1 job(s) submitted to cluster 9.
```

The last job was submitted as job 9. The next step is to monitor these jobs until they're finished.

Monitoring Job Status

To monitor the status of your Condor jobs use the `condor_q` command:

```
$ condor_q
-- Submitter: master : <10.220.226.138:52585> : master
  ID      OWNER      SUBMITTED      RUN_TIME ST PRI  SIZE CMD
    8.0    myuser      12/12 21:31    0+00:06:40 R  0   0.0  sleep 300
    9.0    myuser      12/12 21:31    0+00:05:56 R  0   0.0  sleep 300

2 jobs; 0 idle, 2 running, 0 held
```

From the output above we see that both jobs are currently running. To find out which cluster nodes the jobs are running on pass the `-run` option:

```
$ condor_q -run
-- Submitter: master : <10.220.226.138:52585> : master
  ID      OWNER      SUBMITTED      RUN_TIME HOST(S)
    8.0    myuser      12/12 21:31    0+00:05:57 master
    9.0    myuser      12/12 21:31    0+00:05:13 node001
```

Here we see that job 8 is running on the master and job 9 is running on node001. If your job is taking too long to run you can diagnose the issue by passing the `-analyze` option to `condor_q`:

```
$ condor_q -analyze
```

This will give you verbose output showing which scheduling conditions failed and why.

Canceling Jobs

In some cases you may need to cancel queued or running jobs either because of an error in your job script or simply because you wish to change job parameters. Whatever the case may be you can cancel jobs by passing the job ids to `condor_rm`:

```
$ condor_rm 9
Cluster 9 has been marked for removal.
```

The above example removes job 9 from the condor queue.

1.6.7 Hadoop Plugin

From the [Hadoop](#) homepage:

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using a simple programming model. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

This plugin will automatically configure the Hadoop framework on your cluster(s). It will also configure [dumbo](#) which provides a convenient Python API for writing MapReduce programs in Python as well as useful tools that make it easier to manage HDFS.

See also:

Learn more about Hadoop at it's [project page](#)

Usage

To use this plugin add a plugin section to your starcluster config file:

```
[plugin hadoop]
setup_class = starcluster.plugins.hadoop.Hadoop
```

Next update the PLUGINS setting of one or more of your cluster templates to include the hadoop plugin:

```
[cluster mycluster]
plugins = hadoop
```

The next time you start a cluster the hadoop plugin will automatically be executed on all nodes. If you already have a cluster running that didn't originally have hadoop in its plugin list you can manually run the plugin using:

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

>>> Running plugin hadoop
>>> Configuring Hadoop...
>>> Adding user myuser to hadoop group
3/3 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Installing configuration templates...
3/3 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring environment...
3/3 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring MapReduce Site...
3/3 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring Core Site...
3/3 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring HDFS Site...
3/3 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring masters file...
3/3 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring slaves file...
3/3 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring HDFS...
3/3 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring dumbo...
3/3 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Starting namenode...
>>> Starting secondary namenode...
>>> Starting datanode on master...
>>> Starting datanode on node001...
>>> Starting datanode on node002...
3/3 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Starting jobtracker...
>>> Starting tasktracker on master...
>>> Starting tasktracker on node001...
```

```
>>> Starting tasktracker on node002...
3/3 ||||| 100%
>>> Job tracker status: http://ec2-XXXX.compute-1.amazonaws.com:50030
>>> Namenode status: http://ec2-XXXX.compute-1.amazonaws.com:50070
>>> Shutting down threads...
20/20 ||||| 100%
```

Once the plugin has completed successfully you should be able to login and begin using HDFS and the Hadoop framework tools. Hadoop's home directory, where all of the Hadoop jars live, is `/usr/lib/hadoop`.

Advanced Configuration

If you'd like to change the hadoop temporary directory (ie `hadoop.tmp.dir`) from the default `/mnt/hadoop` update the `[plugin]` section above with:

```
[plugin hadoop]
setup_class = starcluster.plugins.hadoop.Hadoop
hadoop_tmpdir = /path/to/hadoop/temp
```

This plugin creates a custom `mapred-site.xml` file for each node that, among other things, configures the `mapred.tasktracker.{map,reduce}.tasks.maximum` parameters based upon the node's CPU count. The plugin uses a simple heuristic (similar to the one used in Amazon's Elastic Map Reduce hadoop configs) that assigns 1 map per CPU and ~1/3 reduce per CPU by default in order to use all CPUs available on each node. You can tune these ratios by updating the `[plugin]` section above with the following settings:

```
[plugin hadoop]
setup_class = starcluster.plugins.hadoop.Hadoop
map_to_proc_ratio = 1.0
reduce_to_proc_ratio = 0.3
```

Hadoop Web Interfaces

The Hadoop plugin will launch two web-based interfaces that you can access via your web browser. These web interfaces give you real-time stats for the Hadoop job tracker and namenode. The urls for the job tracker and namenode are given at the end of the output of the plugin:

```
>>> Job tracker status: http://ec2-XXXX.compute-1.amazonaws.com:50030
>>> Namenode status: http://ec2-XXXX.compute-1.amazonaws.com:50070
```

Here's what the job tracker page should look like:

master Hadoop Map/Reduce Administration [Quick Links](#)

State: RUNNING
 Started: Tue Dec 27 19:55:23 UTC 2011
 Version: 0.20.2-cdh3u2, 95a824e4005b2a94fe1c11f1ef9db4c672ba43cb
 Compiled: Thu Oct 13 21:52:18 PDT 2011 by root from Unknown
 Identifier: 201112271955

Cluster Summary (Heap Size is 9.06 MB/966.69 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Excluded Nodes
0	0	0	3	0	0	0	0	6	6	4.00	0	0

Scheduling Information

Queue Name	State	Scheduling Information
default	running	N/A

Filter (Jobid, Priority, User, Name)
 Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

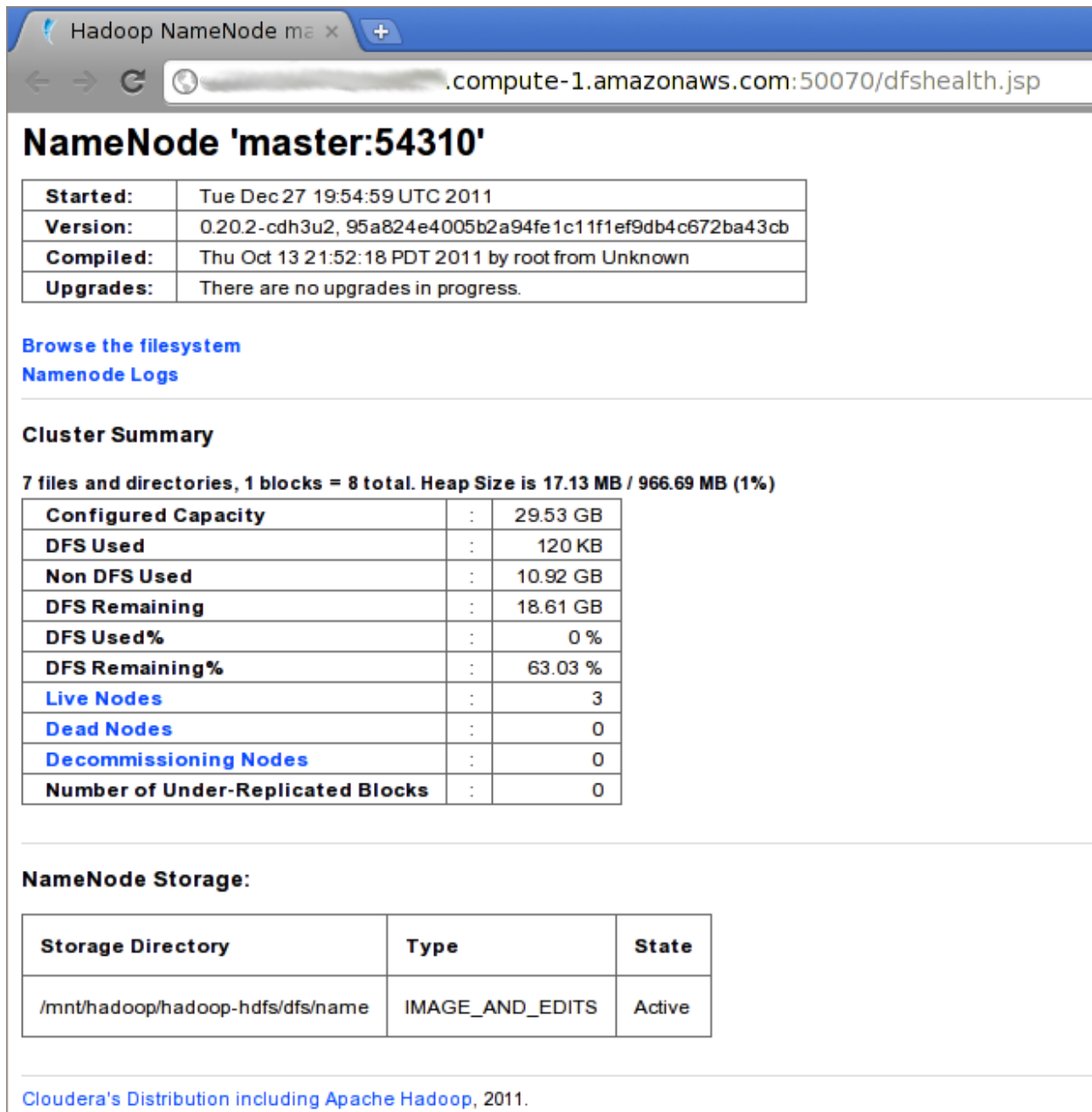
Retired Jobs

Local Logs

[Log](#) [directory](#), [Job Tracker History](#)

Cloudera's Distribution including Apache Hadoop, 2011.

Here's what the namenode page should look like



NameNode 'master:54310'

Started:	Tue Dec 27 19:54:59 UTC 2011
Version:	0.20.2-cdh3u2, 95a824e4005b2a94fe1c11f1ef9db4c672ba43cb
Compiled:	Thu Oct 13 21:52:18 PDT 2011 by root from Unknown
Upgrades:	There are no upgrades in progress.

[Browse the filesystem](#)
[Namenode Logs](#)

Cluster Summary

7 files and directories, 1 blocks = 8 total. Heap Size is 17.13 MB / 966.69 MB (1%)

Configured Capacity	:	29.53 GB
DFS Used	:	120 KB
Non DFS Used	:	10.92 GB
DFS Remaining	:	18.61 GB
DFS Used%	:	0 %
DFS Remaining%	:	63.03 %
Live Nodes	:	3
Dead Nodes	:	0
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	0

NameNode Storage:

Storage Directory	Type	State
/mnt/hadoop/hadoop-hdfs/dfs/name	IMAGE_AND_EDITS	Active

[Cloudera's Distribution including Apache Hadoop](#), 2011.

Using Dumbo to Drive the Hadoop Framework

If you are familiar with the core Hadoop framework you should be able to get started quickly using the default Hadoop tool suite. However, if you're a new user or if you're tired of the verbosity of the core Hadoop framework, the Hadoop plugin also configures `dumbo` on your cluster. Dumbo provides a convenient Python API for writing MapReduce programs and in general makes things much easier when working with the Hadoop framework.

Note: Every `dumbo` command run must include the option `-hadoop starcluster` in order to run on the cluster using Hadoop/HDFS. Without this flag `dumbo` will run using the local environment instead of the Hadoop cluster.

Managing HDFS

You can quickly browse your Hadoop HDFS on any node using dumbo:

```
$ dumbo ls / -hadoop starcluster
```

To upload files to your Hadoop HDFS:

```
$ dumbo put /path/to/file /HDFS/path -hadoop starcluster
```

If you'd rather quickly view a file or set of files on HDFS without downloading:

```
$ dumbo cat /HDFS/path/to/file/or/dir -hadoop starcluster
```

To copy files from your Hadoop HDFS:

```
$ dumbo get /HDFS/path/to/file /local/destination/path -hadoop starcluster
```

You can also remove files and directories from your Hadoop HDFS:

```
$ dumbo rm /HDFS/path/to/file/or/dir -hadoop starcluster
```

Using the Streaming API

Writing Hadoop mappers and reducers with dumbo is very easy. Here's an example for a simple word count:

```
def mapper(key, value):
    for word in value.split():
        yield word, 1

def reducer(key, values):
    yield key, sum(values)

if __name__ == "__main__":
    import dumbo
    dumbo.run(mapper, reducer)
```

Let's assume this is saved to `$HOME/wordcount.py` and we're currently in the `$HOME` directory. To run this example we first upload a text file to HDFS:

```
$ dumbo put /path/to/a/textfile.txt in.txt -hadoop starcluster
```

Next we run the `wordcount.py` example using the `in.txt` file we just put on HDFS:

```
$ dumbo start wordcount.py -input in.txt -output out -hadoop starcluster
```

This will run the word count example using the streaming API and dump the results to a new `out` directory on HDFS. To view the results:

```
$ dumbo cat out/part* -hadoop starcluster
```

If you'd rather download the entire results directory instead:

```
$ dumbo get out out -hadoop starcluster
```

See also:

Have a look at [Dumbo's documentation](#) for more details

1.6.8 MPICH2 Plugin

By default StarCluster includes OpenMPI. However, for either performance or compatibility reasons, you may wish to use MPICH2 which provides an alternate MPI implementation. The MPICH2 plugin will install and configure MPICH2 on your clusters.

Usage

To use this plugin add a plugin section to your starcluster config file:

```
[plugin mpich2]
setup_class = starcluster.plugins.mpich2.MPICH2Setup
```

Next update the PLUGINS setting of one or more of your cluster templates to include the MPICH2 plugin:

```
[cluster mycluster]
plugins = mpich2
```

The next time you start a cluster the MPICH2 plugin will automatically be executed and MPICH2 will be installed and configured on all nodes. If you already have a cluster running that didn't originally have MPICH2 in its plugin list you can manually run the plugin using:

```
$ starcluster runplugin mpich2 mycluster
StarCluster - (http://star.mit.edu/cluster)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

>>> Running plugin mpich2
>>> Setting up MPICH2 hosts file on all nodes
3/3 | 100%
>>> Setting MPICH2 as default MPI on all nodes
3/3 | 100%
>>> MPICH2 is now ready to use
>>> Use mpicc, mpif90, mpirun, etc. to compile and run your MPI apps
```

Building and Running MPI Programs

To build your MPI code use the standard procedure of compiling with mpicc, mpic++, or mpif90:

```
$ starcluster sshmaster mycluster -u myuser
myuser@master $ cat << EOF > mpihelloworld.c
/* C Example */
#include <stdio.h>
#include <mpi.h>
#include <unistd.h>
int main (argc, argv)
    int argc;
    char *argv[];
{
    char hostname[1024];
    gethostname(hostname, 1024);
    int rank, size;
    MPI_Init (&argc, &argv); /* starts MPI */
    MPI_Comm_rank (MPI_COMM_WORLD, &rank); /* get current process id */
    MPI_Comm_size (MPI_COMM_WORLD, &size); /* get number of processes */
    printf("Hello world from process %d of %d on %s\n", rank, size, hostname);
}
```

```

    MPI_Finalize();
    return 0;
}
EOF
myuser@master $ mpicc -o mpihw mpihelloworld.c

```

Note: You should put the MPI binary in your \$HOME folder on the cluster in order for the binary to be NFS-shared to the rest of the nodes in the cluster.

To run the code simply use `mpirun` and pass in the number of processors you wish to use on the cluster via the `-np` option:

```

myuser@master $ mpirun -np 3 ./mpihw
Hello world from process 0 of 3 on master
Hello world from process 2 of 3 on node002
Hello world from process 1 of 3 on node001

```

This will use 3 processors to run the hello world MPI code above. You do not need to create or specify a hosts file. The hostfile is automatically created and set as the default `MPICH2` hosts file by the plugin.

1.6.9 MySQL Cluster Plugin

This plugin automates the configuration and startup of a MySQL Cluster on StarCluster.

Configuration Options

To use this plugin add the following to your StarCluster config file: file:

```

[plugin mysqlcluster]
setup_class = starcluster.plugins.mysql.MySqlCluster
num_replicas = 2
data_memory = 80M
index_memory = 18M
dump_file = test.sql
dump_interval = 60
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 there is only one copy of your data. The loss of a single data node 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 indicate 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.

DUMP_FILE: Path to sql file on the cluster to back databases to. Will be created if it does not exist along with all parent directories.

DUMP_INTERVAL: How often, in minutes, to backup databases to `DUMP_FILE`.

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

Creates data and backup directories, changes ownership to mysql user # Generates /etc/mysql/ndb_mgmd.cnf configuration file on master. # Generates /etc/mysql/my.cnf configuration file on all nodes. # Kills mysql processes on all nodes. # Starts Management Client on master. # Starts mysql on query nodes # 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.

1.6.10 Package Installer Plugin

The `PackageInstaller` plugin installs a list of Ubuntu packages on all nodes in parallel.

Usage

To use this plugin add the following to your starcluster config file:

```
[plugin pkginstaller]
setup_class = starcluster.plugins.pkginstaller.PackageInstaller
packages = mongodb, python-pymongo
```

The packages setting specifies the list of Ubuntu packages to install on each node. The above example will install mongodb and python-pymongo on all nodes in the cluster.

Once you've configured the PackageInstaller plugin the next step is to add it to the plugins list in one of your cluster templates in the config:

```
[cluster mycluster]
plugins = pkginstaller
```

If you already have a cluster running that didn't originally include the PackageInstaller plugin in its config you can manually run the plugin on the cluster using:

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

>>> Running plugin pkginstaller
>>> Installing the following packages on all nodes:
mongodb, python-pymongo
2/2 ||||| 100%
```

1.6.11 Python Package Installer Plugin

The PyPkgInstaller plugin installs a list of Python packages on all nodes in parallel using `pip` (by default).

Usage

To use this plugin add the following to your starcluster config file with the list of packages to install. For instance to install libraries to build an HTTP API with a database:

```
[plugin webapp-packages-installer]
setup_class = starcluster.plugins.pypkginstaller.PyPkgInstaller
packages = flask, SQLAlchemy
```

The packages setting specifies the list of Python packages to install on each node.

Once you've configured the PyPkgInstaller plugin the next step is to add it to the plugins list in one of your cluster templates in the config:

```
[cluster mycluster]
plugins = webapp-packages-installer
```

If you already have a cluster running that didn't originally include the PyPkgInstaller plugin in its config you can manually run the plugin on the cluster using:

```
$ starcluster runplugin webapp-packages-installer mycluster
StarCluster - (http://star.mit.edu/cluster)
Software Tools for Academics and Researchers (STAR)
```

Please submit bug reports to starcluster@mit.edu

```
>>> Running plugin webapp-packages-installer
>>> Installing Python packages on all nodes:
>>> $ pip install flask
>>> $ pip install SQLAlchemy
2/2 ||||| 100%
>>> PyPkgInstaller took 0.317 mins
```

Installing Unreleased Software

pip can also install the development branch of software project directly from source code repositories such as github. For instance the following configuration makes it possible to install the master branch of IPython. If this plugin is configured to run before *IPython Cluster Plugin*, this makes it possible to test yet unreleased features of IPython.parallel and notebook:

```
[plugin ipython-dev]
setup_class = starcluster.plugins.py_pkginstaller.PyPkgInstaller
packages = pyzmq,
           python-msgpack,
           git+http://github.com/ipython/ipython.git
```

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

```
[cluster mycluster]
plugins = ipython-dev, ipcluster
```

1.6.12 Xvfb Plugin

Xvfb, or X virtual framebuffer, is an X11 server that performs all graphical operations in memory without showing any screen output. This plugin configures an Xvfb server on all nodes in the cluster and sets the `DISPLAY` variable on the nodes accordingly.

Usage

To use this plugin add a plugin section to your starcluster config file:

```
[plugin xvfb]
setup_class = starcluster.plugins.xvfb.XvfbSetup
```

Next update the `PLUGINS` setting of one or more of your cluster templates to include the xvfb plugin:

```
[cluster mycluster]
plugins = xvfb
```

The next time you start a cluster the Xvfb plugin will automatically be executed and your `DISPLAY` setting will be set to `:1` on all nodes. If you already have a cluster running that didn't originally have Xvfb in its plugin list you can manually run the plugin using:

```
$ starcluster runplugin xvfb mycluster
StarCluster - (http://star.mit.edu/cluster)
Software Tools for Academics and Researchers (STAR)
```


Please submit bug reports to starcluster@mit.edu

```
>>> Running plugin xvfb
>>> Installing Xvfb on all nodes
3/3 ||||| 100%
>>> Launching Xvfb Server on all nodes
3/3 ||||| 100%
```

Now anytime you login and launch a graphical application on the nodes the application will be started and rendered within the virtual framebuffer on `DISPLAY=:1`

1.7 Frequently Asked Questions

Below are answers to some frequently asked questions on the StarCluster mailing list.

1.7.1 Garbage Packet Received

If you're using StarCluster and encounter the following error you most likely are not using a compatible StarCluster AMI:

```
Traceback (most recent call last):
  File "/usr/lib/python2.6/site-packages/starcluster/threadpool.py", line 32, in run
    job.run()
  File "/usr/lib/python2.6/site-packages/starcluster/threadpool.py", line 59, in run
    r = self.method(*self.args, **self.kwargs)
  File "/usr/lib/python2.6/site-packages/starcluster/node.py", line 661, in set_hostname
    hostname_file = self.ssh.remote_file("/etc/hostname", "w")
  File "/usr/lib/python2.6/site-packages/starcluster/ssh.py", line 284, in remote_file
    rfile = self.sftp.open(file, mode)
  File "/usr/lib/python2.6/site-packages/starcluster/ssh.py", line 174, in sftp
    self._sftp = ssh.SFTPCClient.from_transport(self.transport)
  File "/usr/lib/python2.6/site-packages/ssh/sftp_client.py", line 106, in from_transport
    return cls(chan)
  File "/usr/lib/python2.6/site-packages/ssh/sftp_client.py", line 87, in __init__
    server_version = self._send_version()
  File "/usr/lib/python2.6/site-packages/ssh/sftp.py", line 108, in _send_version
    t, data = self._read_packet()
  File "/usr/lib/python2.6/site-packages/ssh/sftp.py", line 179, in _read_packet
    raise SFTPError('Garbage packet received')
SFTPError: Garbage packet received
```

In this case you should update your `NODE_IMAGE_ID` setting in one of your cluster templates in the config to point to a StarCluster AMI. You can get a list of currently available StarCluster AMIs using the `listpublic` command:

```
$ starcluster listpublic
```

You can also list available StarCluster AMIs in other regions:

```
$ starcluster -r sa-east-1 listpublic
```

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://star.mit.edu/cluster)
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/logs/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 `$HOME/.starcluster/logs/crash-report-$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.

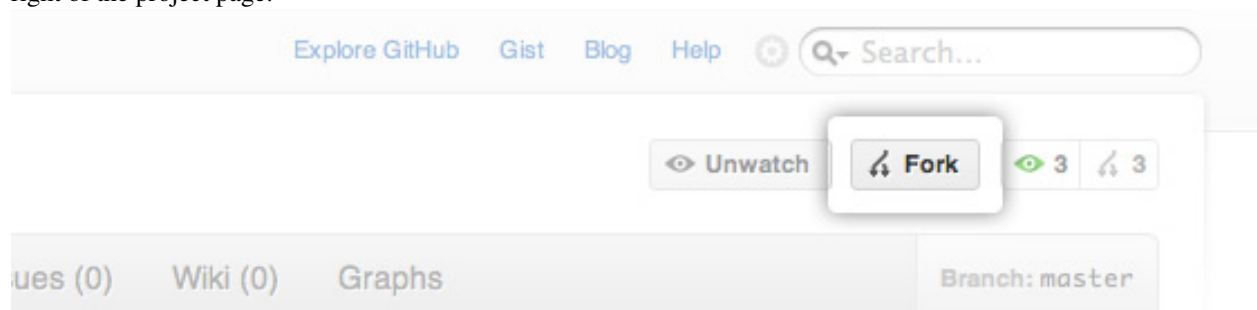
1.9.1 Overview

1.9.2 Sign-up for a github account

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 privileges 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 git@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!
```

You can and should also set up a git hook to do this automatically prior to every commit:

```
$ cd $STARCLUSTER_REPO
$ ln -s $PWD/git-hooks/pre-commit $PWD/.git/hooks/pre-commit
```

Now whenever you run `git commit` the `check.py` script will check the files to be committed for errors, e.g.:

```
$ git commit
>>> Running pyflakes...
starcluster/awsutils.py:333: undefined name 'blah'
ERROR: pyflakes failed on some source files
ERROR: please fix the errors and re-run this script
```

1.9.7 Running the Tests

Before submitting a pull request please make sure that your changes pass StarCluster's current tests. All of the tests run locally and do not require an AWS/EC2 account. To run the tests simply install *nose* and then run *nosetests*:

```
$ pip install nose
$ cd $STARCLUSTER_REPO
$ nosetests
.....
-----
Ran 50 tests in 18.222s

OK
```

If you get an *OK* at the end all is well. Otherwise your changes have introduced errors that need to be fixed before submitting a PR.

1.9.8 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 publicly available Ubuntu-based Amazon Machine Images (AMI) configured for distributed and parallel computing.
- AMI includes OpenMPI, OpenBLAS, Lapack, NumPy, SciPy, and other useful scientific libraries.
- Clusters are automatically configured with NFS, Open Grid Scheduler (formerly SGE) queuing system, and password-less ssh between machines
- Plugin System - plugin framework that enables users to customize the cluster. Plugins are executed when adding/removing a node as well as when restarting/shutting down the entire cluster. StarCluster has many *built-in plugins* ready to use out-of-the-box. Users can also implement their own plugins to further customize the cluster for their needs.
- EBS-Backed Clusters - Support for starting/stopping EBS-backed clusters on EC2.
- Cluster Compute Instances - Support for all cluster compute instance types.
- 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.
- Security Group Permissions - ability to specify permission settings to be applied automatically to a cluster's security group after it's been started.
- Multiple Instance Types - support for specifying instance types on a per-node basis.
- Unpartitioned Volumes - StarCluster supports both partitioned and unpartitioned EBS volumes.
- Numerous helper commands for common EC2/S3 operations:

```
listinstances: List all running EC2 instances
listspots: List all EC2 spot instance requests
listimages: List all registered EC2 images (AMIs)
listpublic: List all public StarCluster images on EC2
listkeypairs: List all EC2 keypairs
createkey: Create a new Amazon EC2 keypair
removekey: Remove a keypair from Amazon EC2
s3image: Create a new instance-store (S3) AMI from a running EC2 instance
ebsimage: Create a new EBS image (AMI) from a running EC2 instance
showimage: Show all AMI parts and manifest files on S3 for an instance-store AMI
downloadimage: Download the manifest.xml and all AMI parts for an instance-store AMI
removeimage: Deregister an EC2 image (AMI)
createvolume: Create a new EBS volume for use with StarCluster
listvolumes: List all EBS volumes
resizevolume: Resize an existing EBS volume
removevolume: Delete one or more EBS volumes
spothistory: Show spot instance pricing history stats (last 30 days by default)
showconsole: Show console output for an EC2 instance
listregions: List all EC2 regions
listzones: List all EC2 availability zones in the current region (default: us-east-1)
listbuckets: List all S3 buckets
showbucket: Show all files in an S3 bucket
```

1.12 StarCluster TODOs

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

1.12.1 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`

1.12.2 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.

1.12.3 Use PyStun to Restrict Cluster Access to User's IP-address

StarCluster should support restricting ssh access to the user's current ip-address when creating a new cluster. This feature will need to use the [pystun](#) project to correctly determine the user's public ip behind firewalls and NAT. StarCluster's ssh* commands will also need to be modified to check that the user's current ip has been allowed to access the cluster in the case that they use StarCluster from multiple machines.

1.13 Version History

- 0.95.6
 - News
 - Features
 - Bug Fixes
 - Enhancements
- 0.95.5
 - News
 - Features
 - Bug Fixes
 - Enhancements
- 0.95.4
 - News
 - Features
 - Bug Fixes
 - Enhancements
- 0.95.3
 - News
 - Features
 - Bug Fixes
- 0.95.2
 - News
 - Bug Fixes
- 0.95.1
 - News
 - Bug Fixes
- 0.95
 - News
 - Features
 - Enhancements
 - Bug Fixes
- 0.94.3
 - News
 - Bug Fixes
- 0.94.2
 - News
 - Bug Fixes
- 0.94.1
 - News
 - Features
 - Bug Fixes
- 0.94
 - News
 - Features
 - Bug Fixes
- 0.93.3
 - News
 - Bug Fixes
- 0.93.2
 - News
 - Features
 - Bug Fixes
- 0.93.1
 - News
 - Features
 - Bug Fixes
- 0.93
 - News
 - Features
 - Bug Fixes

- News
- Features
 - * New Plugins
- Bug Fixes

1.13.1 0.95.6

News

Version 0.95.6 of StarCluster is a bug fix release. The biggest fix is for a change in recent versions of boto that prevents launching all clusters (#455). The sections below contain the full list of changes. This version is *not* compatible with clusters started with older versions of StarCluster. Please terminate all clusters before upgrading to 0.95.6.

Features

- Add support for T2 instance types (#407)

Bug Fixes

- awsutils: set encrypted flag to None in custom block device mappings (#455)
- tests: include pytest.ini in MANIFEST file (#454)
- tests: fix for pytest coverage plugin
- sshutils: pass socket timeout down to SCPClient

Enhancements

- node: dont crash on socket.errors (#465)

1.13.2 0.95.5

News

Version 0.95.5 of StarCluster is a bug fix release. The sections below contain the full list of changes. This version is *not* compatible with clusters started with older versions of StarCluster. Please terminate all clusters before upgrading to 0.95.5.

Features

- Add support for R3 instance types (#392, #393)

Bug Fixes

- cluster: fix bug when creating a flat rate cluster with master_instance_type or master_image_id specified

Enhancements

- docs: update image host command to use -I,-m options (#388)

1.13.3 0.95.4

News

Warning: This release fixes a show-stopping issue for users launching spot clusters on a default-VPC account without explicitly specifying a VPC/subnet. It is recommended that you upgrade as soon as possible.

Version 0.95.4 of StarCluster is a bug fix release. The sections below contain the full list of changes. This version is *not* compatible with clusters started with older versions of StarCluster. Please terminate all clusters before upgrading to 0.95.4.

Features

- sge: new `slots_per_host` setting for specifying a custom number of slots per execution host in the cluster (see [Setting the Number of Slots Per Host](#))

Bug Fixes

- cluster: fix `spot_requests` property to correctly account for spot clusters that get automatically assigned to the default VPC by EC2. ([#377](#))
- loadbalance: fetch and use the remote time zone from the master node rather than assuming the timezone is always set to UTC. ([#385](#))
- sge: fix parallel environment slot allocation to respect both the `master_is_exec_host` and the new `slots_per_host` setting.

Enhancements

- awsutils: show progress bar when waiting for spot requests/instances to propagate
- listspots: show detailed status and status message
- sge: remove redundant admin/submit host setup

1.13.4 0.95.3

News

Version 0.95.3 of StarCluster is a bug fix release. StarCluster no longer assigns public IPs for non-default VPC clusters by default now because of the following issues:

1. It opens up the VPC to the internet by default which is a security risk
2. It requires a special VPC configuration (internet gateway attached to the VPC and a route to the gateway with dest CIDR block 0.0.0.0/0 associated with the VPC subnet). Configuring this automatically (which does not happen currently) would be a security risk and without this configuration StarCluster cannot connect to the VPC nodes even though they've been assigned a public IP. See [Connecting to a VPC Cluster](#) for more details.

Users can still enable public IPs for their non-default VPC clusters via the `PUBLIC_IPS` config setting or the new `--public-ips` option to the `start` command assuming they've properly configured their VPC to use public IPs. If the VPC hasn't been properly configured for public IPs an error will be raised during validation. The `--no-public-ips` option has been preserved so that users can disable the config setting (`PUBLIC_IPS`) from

the `start` command. Please see [Using the Virtual Private Cloud \(VPC\)](#) and [Connecting to a VPC Cluster](#) for more details.

Please note that this version is *not* compatible with clusters started with older versions of StarCluster. Please terminate all clusters before upgrading to 0.95.3.

Features

- start: add `-Q`, `-p`, and `--no-spot-master` options
- cluster: update 'Creating cluster' log msg to indicate VPC

Bug Fixes

- users: pass `-n` to `echo` to avoid trailing newlines (#375)
- cluster: do not use public IPs with VPC by default (#372)
- cluster: fix VPC public IPs validation

1.13.5 0.95.2

News

Version 0.95.2 of StarCluster is a bug fix release. Please note that this version is *not* compatible with clusters started with older versions of StarCluster. Please terminate all clusters before upgrading to 0.95.2.

Bug Fixes

- cluster: sort chunked cluster tags before loading (#371)
- config: set `public_ip` setting type to boolean (#366)

1.13.6 0.95.1

News

Version 0.95.1 of StarCluster is a bug fix release. Please note that this version is *not* compatible with clusters started with older versions of StarCluster. Please terminate all clusters before upgrading to 0.95.1.

Bug Fixes

- cluster: fix max length error when applying cluster tags (#348)
- start: fix default value for `--no-public-ips` option (#366)
- cluster: fix for custom security group permissions (#356)

1.13.7 0.95

News

Version 0.95 of StarCluster is a feature release with many enhancements and bug fixes. This release has support for Amazon's [Virtual Private Cloud \(VPC\)](#). See [Using the Virtual Private Cloud \(VPC\)](#) for details on using StarCluster with VPC. This release comes with new Ubuntu 13.04 AMIs for 32bit and 64bit platforms as well as an HVM AMI that comes with drivers for GPU and enhanced networking. The HVM AMI also comes with the CUDA toolkit, PyCuda, MAGMA, and OGS GPU sensor. These new AMIs are available in all AWS regions. Please use the **listpublic** command to find them:

```
$ starcluster listpublic
```

The sections below list the full set of features, enhancements, and bug fixes for this release. Please note that this version is *not* compatible with clusters started with older versions of StarCluster. Please terminate all clusters before upgrading to 0.95.

Features

- **Add Virtual Private Cloud (VPC) support**
- **Update base StarCluster AMI ids to 13.04**
- Add support for new C3 instance types
- Add support for new GRID GPU g2.2xlarge (non-GPGPU) instance type
- Add support for m3.{medium,large} instance types
- Add support for new I2 instance family
- start: add `--subnet (-N)` and `--no-public-ips` option (VPC-only)
- start: add `--dns-prefix (-P)` option to prefix hostnames with the cluster tag
- removenode: added `--force (-f)` option to removenode
- removenode: prompt for confirmation and add `--confirm (-c)` option
- sshmaster, sshnode: add `--pseudo-tty (-t)` option
- createkey: add `--import-key (-i)` option
- cluster: validate imported keypair fingerprints
- spothistory: add `--vpc` and `--classic` options
- spothistory: use classic or vpc prices based on default vpc
- hadoop: tune number of map/reduce tasks to number of CPUs
- hadoop: document all advanced settings
- listclusters, listinstances: show vpc and subnet ids

Enhancements

- removenode: update usage to mimic addnode (ie added `-n` and `-a` options similar to addnode) - old usage works but is deprecated
- Removed internal scp module in favor of @jbardin's scp module

- awsutils: wait for placement and security group deletion propagation in `ec2.delete_group`
- node: don't force NFSv3 - use OS default (NFSv4 on recent AMIs)
- ebsimage: reduce rsync output by switching `-v` to `-q` (`s3->ebs`)
- ebsimage: log the ebs volume formatting step
- volume: log the exception when volume creation fails
- loadbalance: set UTC tzinfo in all datetime objects
- utils: use UTC timezone for all datetime objects
- cluster: deprecated all `get_node_by*` calls in favor of `cluster.get_node`
- awsutils: raise exc in `EasyEC2.wait_for_propagation` after `max_retries`
- Use `pytest` to run tests

Bug Fixes

- node: fix stale NFS entries from force-terminated spot nodes
- node: remove 'user' setting from NFS mount options
- node: add work-around for bug in debian nfs-kernel-server init script in Ubuntu 13.04+
- node: use `apt-get` instead of `apt` in `package_provider` property
- node: run `apt-get update` before `install` in `apt_install`
- cluster: always use master's zone if not specified
- cluster: store `availability_zone` in core cluster settings tag
- cluster: raise error if `MOUNT_PATH=/`
- cluster: retry deleting placement group until successful
- cluster: check for master in `remove_nodes` prior to removing any nodes
- loadbalance: add nodes when below minimum
- loadbalance: raise an exception if `min_nodes > max_nodes` in `loadbalancer`
- loadbalance: fix crash in `get_all_stats()` when `arr` is empty
- ebsimage: set fs label = / on root vol when converting `s3->ebs`
- ebsimage: fix for `bmap` when converting `s3->ebs`
- ebsimage: only set 1 ephemeral drive when converting `s3->ebs`
- volume: only detach if `vol != available` in `_delete_new_volume`
- volume: remove duplicates from volume hosts list
- spothistory: improve accuracy by not using a possibly out-of-sync local time stamp for the end date as long as `-e` is not specified
- spothistory: set UTC tzinfo on all datetime objects
- awsutils: raise exception if keypair exists in `create_keypair`
- awsutils: use instance store `bmap` for 32bit `m1.small`
- awsutils: dont specify ephemeral drives w/ `t1.micro`

- cli: raise parser error if tag is incorrectly specified
- condor: fetch & create condor dirs from config
- condor: set SCHEDD_NAME on each host
- condor: improve dedicated scheduler config

1.13.8 0.94.3

News

Version 0.94.3 of StarCluster is a bug fix release. See the “Bug Fixes” sections below for details. In general it’s recommended to terminate all clusters before upgrading to new versions of StarCluster although this is not always strictly required.

Bug Fixes

- Fixed ‘InvalidSnapshot.NotFound’ errors on cluster launch
- Fixed issue with Server.InternalError: Internal error on launch when using m1.small instance type with a 32bit AMI.
- Updated default AMIs in config template to 64bit AMIs

1.13.9 0.94.2

News

Version 0.94.2 of StarCluster is a bug fix release. See the “Bug Fixes” sections below for details. In general it’s recommended to terminate all clusters before upgrading to new versions of StarCluster although this is not always strictly required.

Bug Fixes

- node: fixed regression where the instance’s connection object was not being reused which was causing issues for folks using regions other than the default us-east-1.

1.13.10 0.94.1

News

Version 0.94.1 of StarCluster is a bug fix release. See the “Features” and “Bug Fixes” sections below for details. In general it’s recommended to terminate all clusters before upgrading to new versions of StarCluster although this is not always strictly required.

Features

- restart: add `--reboot-only` option
- awsutils: force `delete_on_termination` on root volume when launching instances

- delay using optional imports in `starcluster.utils` until needed (reduces the time it takes to load StarCluster)

Bug Fixes

- cluster: wait for all instance and spot requests to propagate (fixes ‘instance not found’ errors and ‘missing’ spot requests when waiting for cluster to come up)
- cluster: terminate any instance that’s been pending for more than 15 minutes (fixes infinite wait due to a forever pending node)
- s3image: fix block device map issue when host instance is EBS-backed
- awsutils: fix custom block device map for S3 instances/AMIs
- sshnode: fix keypair validation bug
- node: fix for duplicate entries in `known_hosts` file
- node: fix bug where credentials attributes were missing from `node.ec2` object
- loadbalancer: always pass `--utc` when calling `date` command
- loadbalancer: set matplotlib backend to Agg - allows `--plot` option to work without a display
- config: convert `EC2_CERT` and `EC2_PRIVATE_KEY` in `[aws]` section to absolute path
- config: fix typo in config template (`PROTOCOL` -> `IP_PROTOCOL`)
- tmux plugin: fix race condition where `new-window -t` is called before `new-session` has finished
- spothistory: update help docs for `--plot` option

1.13.11 0.94

News

Version 0.94 of StarCluster is a feature release that fixes *many* bugs. See the “Features” and “Bug Fixes” sections below for details. This release is not compatible with any active clusters that were created using any previous version of StarCluster. Please terminate all existing clusters before upgrading to 0.94.

Features

- Support for `hs1.8xlarge`, `hi1.4xlarge`, and `m3` instance types
- Added support for userdata scripts (new cluster setting: `userdata_scripts`)
- Add *all* ephemeral drives for any instance type when launching instances
- Updated base AMIs to latest 12.04 versions
- OGS (formerly SGE): set default parallel environment `allocation_rule` to `$fill_up`
- Added authentication agent forwarding to `sshmaster`, `sshnode`, and `sshinstance`
- Support for loading config based on `$STARCLUSTER_CONFIG` environment variable
- New plugin to install python packages with `pip` (`pypkginstaller` plugin)
- New plugin that either copies or autogenerates a boto config for the cluster user
- Added `--force` option to `terminate`/`stop` commands

- Add tags and virtualization type to listinstance output
- Add `--zone (-z)` option to spothistory command

Bug Fixes

- sshutils: source `/etc/profile` by default
- awsutils: validate AWS SSL certificates by default
- awsutils: refetch group before calling authorize
- awsutils: use 3 decimal places in spot history prices
- cluster: fix thread leak when running plugins
- cluster: store cluster metadata in `userdata/tags`
- cluster: dont launch master as spot when `CLUSTER_SIZE=1`
- cluster: reraise uncaught exceptions in `run_plugins`
- cluster: fix instance filtering for default VPC
- cluster: only clean-up placement groups in supported regions
- clustersetup: ebs volume mounting improvements
- clustersetup: raise exception if user's `$HOME` is owned by root
- improve ebs volume device and partition detection
- config: allow inline comments in the config file
- config: support `[vol]` shortcut in config
- config: document/warn where/when env overrides `cfg`
- threadpool: clear the exceptions queue in `wait()`
- node: fix `Node.root_device_name` for misconfigured AMIs
- node: fix `~/ssh` ownership in `generate_key_for_user`
- sge: Export `$DRMAA_LIBRARY_PATH` for Python DRMAA
- cli: use logging module to log/show exceptions
- cli: write header to `sys.stderr`
- logger: move console logs from `stderr` to `stdout`
- logger: output error messages to `stderr`
- logger: remove in-memory log handler (memory-leak fix)
- fix for `DependencyViolation` when deleting security group
- replace `group-name` filter with `instance.group-name`
- spothistory: show *current* spot price instead of oldest

resizevol/createvol:

- call `e2fsck` before `resize2fs` if needed
- cleanup vol on failure in `resize()`
- fix bug in `_warn_about_volume_hosts`

- fix bug in sfdisk repartitioning
- set default instance type to t1.micro for volume hosts
- fix bug when cleaning up failed volumes
- use force_spot_master when launching cluster

ebsimage/s3image:

- fetch root device from instance metadata
- only wait for snapshot if root device is in bmap
- show progress bar for CreateImage snapshot
- remove ssh host keys in clean_private_data
- use ec2.get_snapshot to fetch snapshots

loadbalance:

- collect all queue info from qstat
- fix logic in SGStats._count_tasks
- fix node count in _eval_remove_node
- get slot calculations from all.q
- handle qacct error on fresh SGE install
- improve qstat parsing
- maintain a single instance of SGStats
- move static settings to __init__
- remove bounds on interval and wait_time
- remove estimated time to completion calcs
- replace 'qlen > ts' condition in _eval_add_node
- require positive ints for numeric options
- take slots into account in _eval_add_node
- update sge tests for latest parser changes
- use # of nodes for max_nodes check (not exec hosts)
- don't wait for cluster to stabilize to add a node when there are no slots
- immediately add a node when jobs are in the queue and no slots exists

IPython plugin:

- Add section on engine restart + note on growing new engines
- ipcluster: raise exception if IPython isnt installed
- Better docstrings for the stop and restart plugins
- Better packer configuration support + updated doc
- Make it possible to configure a custom notebook folder
- Make it possible to restart engine, better logs, make msgpack setup explicit
- Open the controller ports for IPython 0.13.1

- Use the cluster thread pool to start the engines on the nodes
- Update the documentation according to tests done on 0.13.1

1.13.12 0.93.3

News

Version 0.93.3 of StarCluster is a patch release that fixes several bugs. See the “Bug Fixes” section below for details.

- Bumped boto dep to 2.3.0
- Updated and improved default config template

Bug Fixes

- sge: fix bug where master could never be an exec host even when toggling the `master_is_exec_host` plugin setting (issue 89)
- fix bug when setting custom SSH permission without `cidr_ip` (issue 91)
- start-cmd: fix bug where `refresh_interval` is always `None` if `[global]` is not defined in the config (issue 90)
- dont allow `cluster_user=root` when validating cluster settings (issue 92)
- fix another textwrap error with python 2.5

1.13.13 0.93.2

News

Version 0.93.2 of StarCluster fixes several bugs and adds a couple new features. See the “Bug Fixes” section below for details.

- Amazon recently announced a new instance type `m1.medium` which is now supported. Amazon also enabled the `m1.small` and `c1.medium` instance types to be compatible with both 32bit *and* 64bit AMIs now. StarCluster now supports the new `m1.medium` instance type as well as the new changes to `m1.small` and `c1.medium`.
- Switched from paramiko to actively maintained fork called `ssh` (1.7.13)
- loadbalancer: `--kill-master` option renamed to `--kill-cluster`
- Bumped boto dep to 2.2.2
- Bumped jinja dep to 2.6

Features

- SGE setup code has been moved to a plugin that is executed by default (**no configuration changes required**). This plugin supports an optional setting `master_is_exec_host` which controls whether or not the master node executes jobs. This is set to `True` by default. To change this you will need to set `disable_queue=True` in your cluster config and then manually define and add the SGE plugin in order to configure the new plugin’s settings. See [Sun Grid Engine Plugin](#) for more details.
- Added the following new options to `addnode` command: `--image-id`, `--instance-type`, `--availability-zone`, `--bid`. Thanks to Dan Yamins for his contributions. (issue 76)

- New `CreateUsers` plugin that creates multiple cluster users and configures passwordless SSH. Either number of users or a list of usernames can be specified. Also has options to tar and download all SSH keys for generated users. See *Create Users Plugin* for more details.
- Added `-X` option to `sshmaster` and `sshnode` commands for forwarding X11. Can be used both when logging in interactively and when executing remote commands.

Bug Fixes

- catch *Garbage packet received* exception and warn users about incompatible AMI
- `createvolume`: support new `/dev/xvd*` device names
- fix textwrap error with python 2.5 (issue 85)
- `sshmaster`, `sshnode`: load `/etc/profile` by default when executing remote cmds
- `terminate`: handle Ctrl-D gracefully when prompting for input (issue 80)
- `terminate`: ignore keys when terminating manually
- validate `key_location` is readable (issue 77)
- relax validation constraints for running instances when using `starcluster start -x` (issue 78)
- fix cluster validation for *t1.micro* AMIs/instance types
- `loadbalance`: fix `_get_stats` when no exec hosts
- `loadbalance`: respect `min_nodes` when removing nodes (issue 75)
- `loadbalance`: fix `-K` option to kill cluster when queue is empty (issue 86)
- `loadbalance`: retry fetching stats 5x before failing (issue 65)
- skip NFS exports setup when there are no worker nodes
- `sge`: configure SGE profile on all nodes concurrently
- `sge`: use `qconf's -*attr` commands to update cluster config (replaced a bunch of ugly code that parsed SGE's console output)
- add Changelog to `MANIFEST.in` in order to include Changelog in future release dists

1.13.14 0.93.1

News

Warning: It is recommended that you upgrade as soon as possible.

Version 0.93.1 of StarCluster is largely a patch release that fixes several bugs. See the “Bug Fixes” section below for details.

Also, the `put` and `get` commands have been marked stable which means they no longer require `ENABLE_EXPERIMENTAL=True` in the config in order to use them.

Features

- Added `sys.argv` to crash reports for better details when things go wrong
- Implemented `on_add_node` and `on_remove_node` for TMUX plugin to add/remove nodes from the TMUX control-center sessions for both `root` and `CLUSTER_USER`
- Implemented `on_add_node` for the Ubuntu package installer plugin (`pkginstaller`) to ensure new nodes also install the required packages
- Implemented `on_add_node` for the MPICH2 plugin to ensure new nodes are configured for MPICH2 and also added to the hosts file. Moved the MPICH2 hosts file to an NFS-shared location (`/home/mpich2.hosts`) for faster configuration
- Implemented `on_add_node` for the Xvfb plugin that installs and runs Xvfb on newly added nodes

Bug Fixes

- Fixed major NFS bug in `addnode` command where `/etc/exports` was being wiped each time a new node was added. This caused the last-to-be-added node to be the only node correctly configured for NFS in the entire cluster.
- Updated all plugins which subclass `DefaultClusterSetup` to implement `on_add_node` and `on_remove_node` regardless of whether or not they actually use them to avoid the ‘default’ plugin’s implementation being incorrectly called multiple times by plugins when adding/removing nodes using the `addnode` and `removenode` commands
- Fixed string formatting bug in Condor plugin

1.13.15 0.93

News

New StarCluster Ubuntu 11.10 EBS AMIs available including a new HVM AMI for use with CC/CC2/CG instance types! Includes OpenGridScheduler (to replace SGE), Condor, Hadoop/Dumbo, MPICH2, OpenMPI, IPython 0.12 (parallel+notebook), and more. Run `dpkg -l` on the AMIs for the full list of packages.

The new StarCluster Ubuntu 11.10 EBS HVM AMI additionally comes with NVIDIA driver, CUDA Toolkit/SDK, PyCuda, PyOpenCL, and Magma for GPU computing.

These new AMIs are available in all AWS regions. The new `us-east-1` 11.10 EBS AMIs are now the defaults in the StarCluster config template. Use the `listpublic` command to obtain a list of available StarCluster AMIs in other regions:

```
$ starcluster -r sa-east-1 listpublic
```

Note: The HVM AMI is currently only available in the `us-east-1` region until other regions support HVM.

Features

- Added support for new CC2 instance types
- Added support for specifying a proxy host to use when connecting to AWS:

```
[aws info]
aws_proxy = your.proxy.com
aws_proxy_port = 8080
aws_proxy_user = yourproxyuser
aws_proxy_pass = yourproxypass
```

See *Using a Proxy Host* for more details

- Updated IPCluster plugin to support IPython 0.12 with *parallel* and *notebook* support (SSL+password auth). See *IPython Cluster Plugin* for more details
- Keypairs are now validated against EC2 fingerprint which notifies users ahead of time if the file they specified in KEY_LOCATION doesn't match the EC2 keypair specified
- Support for Windows (tested on 32bit Windows 7). See *Installing on Windows* for more details

New Plugins

- **Condor** - experimental support for creating a Condor pool on StarCluster. See *Condor Plugin* for more details
- **Hadoop** - support for creating a Hadoop cluster with StarCluster See *Hadoop Plugin* for more details
- **MPICH2** - configure cluster to use MPICH2 instead of OpenMPI See *MPICH2 Plugin* for more details
- **TMUX** - configures cluster SSH “dashboard” in tmux See *TMUX Plugin* for more details
- **Package installer** - install Ubuntu packages on all nodes concurrently See *Package Installer Plugin* for more details
- **Xvfb** - install and configure an Xvfb session on all nodes (`sets DISPLAY=:1`) See *Xvfb Plugin* for more details
- **MySQL** - install and configure a MySQL cluster on StarCluster. See *MySQL Cluster Plugin* for more details

Bug Fixes

- Fix bug in `put` command where in random cases the last few bytes of the file weren't being transferred
- Raise an error if placement group creation fails
- Fix bug in `terminate` command in the case that nodes are already shutting-down when the `terminate` command is executed
- Fix bug in `s3image` when destination S3 bucket already exists
- Fix `known_hosts` warnings when `ssh`'ing from node to node internally
- Fix trivial logging bug in `createvolume` command

1.13.16 0.92.1

release-date 11-05-2011

Features

- Support for splitting the config into an arbitrary set of files:

```
[global]
include=~/.starcluster/awscreds, ~/.starcluster/myconf
```

See *Splitting the Config into Multiple Files* for more details

- createvolume: support naming/tagging newly created volumes:

```
$ starcluster createvolume --name mynewvol 30 us-east-1d
```

See *Create and Format a new EBS Volume* for more details

- listvolumes: add support for filtering by tags:

```
$ starcluster listvolumes --name mynewvol
$ starcluster listvolumes --tag mykey=myvalue
```

See *Managing EBS Volumes* for more details

- sshmaster, sshnode, sshinstance: support for running remote commands from command line:

```
$ starcluster sshmaster mycluster 'cat /etc/fstab'
$ starcluster sshnode mycluster node001 'cat /etc/fstab'
$ starcluster sshinstance i-999999999 'cat /etc/hosts'
```

See *Running Remote Commands from Command Line* for more details

Bug Fixes

The following bugs were fixed in this release:

spothistory command

- add package_data to sdist in order to include the necessary web media and templates needed for the `--plot` feature. The previous 0.92 version left these out and thus the `--plot` feature was broken. This should be fixed.
- fix bug when launching default browser on mac

start command

- fix bug in option completion when using the start command's `--cluster-template` option

terminate command

- fix bug in terminate cmd when region != us-east-1

listkeypairs command

- fix bug in list_keypairs when no keys exist

Improvements

- listinstances: add 'state_reason' msg to output if available
- add system info, Python info, and package versions to crash-report
- listregions: sort regions by name
- improved bash/zsh completion support. completion will read from the correct config file, if possible, in the case that the global `-c` option is specified while completing.
- separate the timing of cluster setup into time spent on waiting for EC2 instances to come up and time spent configuring the cluster after all instances are up and running. this is useful when profiling StarCluster's performance on large (100+ node) clusters.

1.13.17 0.92

release-date 10-17-2011

News

Note: Before upgrading please be aware that rc1 does not support clusters created with previous versions and will print a warning along with instructions on how to proceed if it finds old clusters. With that said, it's best not to upgrade until you've terminated any currently running clusters created with an old version. Also, the @sc-masters group is no longer needed and can be removed after upgrading by running "starcluster terminate masters"

You can find the new docs for 0.92rc1 here:

<http://star.mit.edu/cluster/docs/0.92rc1/>

NOTE: these docs are still very much a work in progress. If anyone is interested in helping to improve the docs, please fork the project on github and submit pull requests. Here's a guide to get you started:

<http://star.mit.edu/cluster/docs/0.92rc1/contribute.html>

Please upgrade at your leisure, test the new release candidate, and submit any bug reports preferably on StarCluster's github issue tracker or this mailing list.

Features

- Cluster Compute/GPU Instances- Added support for the new Cluster Compute/GPU instance types. Thanks to Fred Rotbart for his contributions
- EBS-Backed Clusters- Added support for starting/stopping EBS-backed clusters on EC2. The `stop` command now stops (instead of terminate) an EBS-backed cluster and terminates an S3-backed cluster. Added a `terminate` command that terminates *any* cluster
- Improved performance - using workerpool library (<http://pypi.python.org/pypi/StarCluster>) to configure nodes concurrently
- New `ebsimage` and `s3image` commands for easily creating new EBS-backed AMIs. Each command supports creating a new AMI from S3 and EBS-backed image hosts respectively. (NOTE: `createimage` has been renamed to `s3image`. you can still call `createimage` but this will go away in future releases)
- 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 SGE 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. 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 supports both partitioned and unpartitioned EBS volumes
- New Plugin Hooks - Plugins can now play a part when adding or removing a node as well as when restarting or shutting down the entire cluster by implementing the `on_remove_node`, `on_add_node`, `on_shutdown`, and `on_reboot` methods respectively
- Added a `runplugin` command and fixed the `run_plugin()` method in the development shell. You can now run a plugin in the dev shell like so:

```
~[1]> cm.run_plugin('myplugin', 'mycluster')
```

- Added support for easily switching regions (without config changes) using a global `-z` option. For example:

```
$ starcluster -r eu-west-1 listclusters
StarCluster - (http://star.mit.edu/cluster) (v. 0.92rc1)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu
```

```
>>> No clusters found...
```

- Added new `resizevolume` command for easily resizing existing EBS volumes
- Added `listregions/listzones` commands

1.14 License: LGPL (version 3)

StarCluster is licensed under LGPLv3. Below is a copy of the license.

GNU LESSER GENERAL PUBLIC LICENSE (Version 3, 29 June 2007)

Copyright (C) 2008 Free Software Foundation, Inc. <<http://fsf.org/>> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

- a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
- b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
- c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

4. Do one of the following:

- o) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

118 Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

- e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute

P

Python Enhancement Proposals

PEP 8, [97](#)