elasticluster Documentation

Release 1.0

Grid Computing Competence Centre, University of Zurich

April 15, 2016

Contents

Introduction

elasticluster aims to provide a user-friendly command line tool to create, manage and setup computional clusters hosted on cloud infrastructures (like Amazon's Elastic Compute Cloud EC2) or a private OpenStack cloud). Its main goal is to get your own private cluster up and running with just a few commands; a YouTube video demoes the basic features of elasticluster.

This project is an effort of the Grid Computing Competence Center at the University of Zurich, licensed under the GNU General Public License version 3.

CHAPTER 2

Features

elasticluster is in active development, but the following features at the current state:

- Simple configuration file to define cluster templates
- Can start and manage multiple independent clusters at the same time
- **Automated cluster setup:**
  - use Debian GNU/Linux, Ubuntu, or CentOS as a base operating system
  - supports multiple batch systems, including SLURM, Grid Engine or Torque/PBS
  - supports Hadoop clusters
  - add useful tools like Ganglia for monitoring...
  - ...or anything that you can install with an Ansible playbook!
- Grow a running cluster

elasticluster is currently in active development: please use the GitHub issue tracker to file enhancement requests and ideas

CHAPTER 3

Architecture

The architecture of elasticluster is quite simple: the configuration file in `~/.elasticluster/config` defines a set of *cluster configurations* and information on how to access a specific cloud webservice (including access id and secret keys).

Using the command line (or, very soon, a simple API), you can start a cluster and override some of the default values, like the number of nodes you want to fire up. Elasticluster will use the boto library to connect to the desired cloud, start the virtual machines and wait until they are accessible via ssh.

After all the virtual machines are up and running, elasticluster will use ansible to configure them.

If you do a *resize* of the cluster (currently only growing a cluster is fully supported) new virtual machines will be created and again ansible will run for *all* of the virtual machines, in order to properly add the new hosts to the cluster.

CHAPTER 4

Table of Contents

4.1 Installation

elasticluster is a Python program; Python version 2.6 is required to run it.

The easiest way to install elasticluster is using pip, this will install the latest **stable** release from the PyPI website. The following section: *Installing from PyPI* will explain you how to do it.

If you instead want to test the *development* version, go to the *Installing from github* section.

4.1.1 Installing from PyPI

It's quite easy to install *elasticluster* using pip; the command below is all you need to install *elasticluster* on your system:

```
pip install elasticluster
```

If you want to run *elasticluster* from source you have to **install** Ansible **first:**

```
pip install ansible
python setup.py install
```

4.1.2 Installing from github

The source code of elasticluster is github, if you want to test the latest development version you can clone the github elasticluster repository.

You need the `git` command in order to be able to clone it, and we suggest you to use python virtualenv in order to create a controlled environment in which you can install elasticluster as normal user.

Assuming you already have `virtualenv` installed on your machine, you first need to create a virtualenv and install *ansible*, which is needed by elasticluster:

```
virtualenv elasticluster
. elasticluster/bin/activate
pip install ansible
```

Then you have to download the software. We suggest you to download it *within* the created virtualenv:

```
cd elasticluster
git clone git://github.com/gc3-uzh-ch/elasticluster.git src
cd src
git submodule init
git submodule update
python setup.py install
```

Now the `elasticluster` should be available in your current environment.

4.2 Configuration

All the information about how to access a cloud provider and how to setup a cluster is stored in a configuration file. The default configuration file is stored in your home directory: `~/.elasticluster/config` but you can specify a different location from the command line with the *-c* option.

When *elasticluster* is run for the first time, if no configuration file is found it will copy a template configuration file in `~/.elasticluster/config`. Such template is fully commented and self documented.

4.2.1 Basic syntax of the configuration file

The file is parsed by ConfigParser module and has a syntax similar to Microsoft Windows INI files.

It consists of *sections* led by a `[sectiontype/name]` header and followed by lines in the form:

```
key=value
```

Section names are in the form `[type/name]` wher *type* must be one of:

`cloud` define a cloud provider

`login` define a way to access a virtual machine

`setup` define a way to setup the cluster

`cluster` define the composition of a cluster. It contains references to the other sections.

`cluster/<clustername>` override configuration for specific group of nodes within a cluster

You must define at least one for each section types in order to have a valid configuration file.

4.2.2 Cloud Section

A `cloud` section named `<name>` starts with:

```
[cloud/<name>]
```

The cloud section defines all properties needed to connect to a specific cloud provider.

You can define as many cloud sections you want, assuming you have access to different cloud providers and want to deploy different clusters in different clouds. The mapping between cluster and cloud provider is done in the *cluster* section (see later).

Currently two cloud providers are available: - boto: supports OpenStack and Amazon EC2 - google: supports Google Compute Engine Therefore the following configuration option needs to be set in the cloud section:

`provider`

  the driver to use to connect to the cloud provider. *boto* or *google*

Valid configuration keys for *boto*

`ec2_url`

the url of the EC2 endpoint. For Amazon is probably something like:

```
https://ec2.us-east-1.amazonaws.com
```

replace `us-east-1` with the zone you want to use while for OpenStack you can get it from the web interface

`ec2_access_key`

the access key (also known as access id) your cloud provider gave you to access its cloud resources.

`ec2_secret_key`

the secret key (also known as secret id) your cloud provider gave you to access its cloud resources.

`ec2_region`

the availability zone you want to use.

Valid configuration keys for *google*

`gce_client_id`

The API client id generated in the Google API Console

`gce_client_secret`

The API client secret generated in the Google API Console

`gce_project_id`

The project id of your Google Compute Engine project

Examples

For instance, to connect to the Hobbes private cloud of the University of Zurich you can use the following:

```
[cloud/hobbes]
provider=ec2_boto
ec2_url=http://hobbes.gc3.uzh.ch:8773/services/Cloud
ec2_access_key=****REPLACE WITH YOUR ACCESS ID****
ec2_secret_key=****REPLACE WITH YOUR SECRET KEY****
ec2_region=nova
```

For Amazon instead (region us-east-1) you can use:

```
[cloud/amazon-us-east-1]
provider=ec2_boto
ec2_url=https://ec2.us-east-1.amazonaws.com
ec2_access_key=****REPLACE WITH YOUR ACCESS ID****
ec2_secret_key=****REPLACE WITH YOUR SECRET KEY****
ec2_region=us-east-1
```

For Google Compute Engine you can use:

[cloud/google] provider=google gce_client_id=****REPLACE WITH YOUR CLIENT ID**** gce_client_secret=****REPLACE WITH YOUR SECRET KEY**** gce_project_id=****REPLACE WITH YOUR PROJECT ID****

OpenStack users

From the horizon web interface you can download a file containing your EC2 credentials by logging in in your provider web interface and clicking on:

"settings"

> **=> *"EC2 Credentials"*** => *"Download EC2 Credentials"*

The `ec2rc.sh` file will contain some values. Update the configuration file:

ec2_url using the value of the variable EC2_URL *ec2_access_key* using the value of the variable EC2_ACCESS_KEY *ec2_secret_key* using the value of the variable EC2_SECRET_KEY

Google Compute Engine users

To generate a client_id and client_secret to access the Google Compute Engine visit the following page: https://code.google.com/apis/console/ 1. Select the project defined as gce_project_id 2. Navigate to *API Access* 3. Click create another client id 4. Select *Installed Application* -> *Other* 5. After clicking the *Create* button you'll see your client_id and

> secret_key in the list of available client ids

Please note: you have to set your google username in the login section below as image_user to be able to use

> Google Compute Engine

4.2.3 Login Section

A `login` section named `<name>` starts with:

```
[login/<name>]
```

This section contains information on how to access the instances started on the cloud, including the user and the SSH keys to use.

Some of the values depend on the image you specified in the *cluster* section. Values defined here also can affect the *setup* section and the way the system is setup.

Mandatory configuration keys

`image_user`

> the remote user you must use to connect to the virtual machine. In case you're using Google Compute Engine you have to set your username
>
> > here. So if your gmail address is karl.marx@gmail.com, your username is karl.marx

`image_sudo`

> Can be *True* or *False*. *True* means that on the remote machine you can execute commands as root by running the *sudo* program.

`image_user_sudo`

> the login name of the administrator. Use *root* unless you know what you are doing...

`user_key_name`

name of the *keypair* to use on the cloud provider. If the keypair does not exist it will be created by elasticluster.

`user_key_private`

file containing a valid RSA or DSA private key to be used to connect to the remote machine. Please note that this must match the `user_key_public` file (RSA and DSA keys go in pairs). Also note that Amazon does not accept DSA keys but only RSA ones.

`user_key_public`

file containing the RSA/DSA public key corresponding to the `user_key_private` private key. See `user_key_private` for more details.

Examples

For a typical Ubuntu machine, both on Amazon and most OpenStack providers, these values should be fine:

```
[login/ubuntu]
image_user=ubuntu
image_user_sudo=root
image_sudo=True
user_key_name=elasticluster
user_key_private=~/.ssh/id_rsa
user_key_public=~/.ssh/id_rsa.pub
```

while for Hobbes appliances you will need to use the *gc3-user* instead:

```
[login/gc3-user]
image_user=gc3-user
image_user_sudo=root
image_sudo=True
user_key_name=elasticluster
user_key_private=~/.ssh/id_rsa
user_key_public=~/.ssh/id_rsa.pub
```

4.2.4 Setup Section

A `setup` section named `<name>` starts with:

```
[setup/<name>]
```

This section contain information on *how to setup* a cluster. After the cluster is started, elasticluster will run a `setup provider` in order to configure it.

Mandatory configuration keys

`provider`

the type of setup provider. So far, only *ansible* is supported.

Ansible-specific mandatory configuration keys

The following configuration keys are only valid if *provider* is *ansible*.

`<class>_groups`

Comma separated list of ansible groups the specific <class> will belong to. For each <class>_nodes in a [cluster/] section there should be a <class>_groups option to configure that specific class of nodes with the ansible groups specified.

If you are setting up a standard HPC cluster you probably want to have only two main groups: *frontend_groups* and *compute_groups*.

To configure a slurm cluster, for instance, you have the following available groups:

`slurm_master` configure this machine as slurm masternode

`slurm_clients` compute nodes of a slurm cluster

`ganglia_master` configure as ganglia web frontend. On the master, you probably want to define *ganglia monitor* as well

`ganglia_monitor` configure as ganglia monitor.

You can combine more groups together, but of course not all combinations make sense. A common setup is, for instance:

```
frontend_groups=slurm_master,ganglia_master,ganglia_monitor
compute_groups=slurm_clients,ganglia_monitor
```

This will configure the frontend node as slurm master and ganglia frontend, and the compute nodes as clients for both slurm and ganglia frontend.

A full list of the available groups is available at the *Playbooks distributed with elasticluster* page.

`<class>_var_<varname>`

an entry of this type will define a variable called `<varname>` for the specific `<class>` and add it to the ansible inventory file.

`playbook_path`

Path to the playbook to use when configuring the system. The default value printed here points to the playbook distributed with elasticluster. The default value points to the playbooks distributed with elasticluster.

Examples

Some (working) examples:

```
[setup/ansible-slurm]
provider=ansible
frontend_groups=slurm_master
compute_groups=slurm_clients

[setup/ansible-gridengine]
provider=ansible
frontend_groups=gridengine_master
compute_groups=gridengine_clients

[setup/ansible-pbs]
provider=ansible
frontend_groups=pbs_master,maui_master
compute_groups=pbs_clients

[setup/ansible_matlab]
# Please note that this setup assumes you already have matlab
# installed on the image that is being used.
```

```
provider=ansible
frontend_groups=mdce_master,mdce_worker,ganglia_monitor,ganglia_master
worker_groups=mdce_worker,ganglia_monitor
```

4.2.5 Cluster Section

A `cluster` section named `<name>` starts with:

```
[cluster/<name>]
```

The cluster section defines a *template* for a cluster. This section has references to each one of the other sections and define the image to use, the default number of compute nodes and the security group.

Mandatory configuration keys

`cloud`

the name of a valid *cloud* section. For instance *hobbes* or *amazon-us-east-1*

`login`

the name of a valid *login* section. For instance *ubuntu* or *gc3-user*

`setup_provider`

the name of a valid *setup* section. For instance, *ansible-slurm* or *ansible-pbs*

`image_id`

image id in *ami* format. If you are using OpenStack, you need to run *euca-describe-images* to get a valid *ami-** id.

`flavor`

the image type to use. Different cloud providers call it differently, could be *instance type*, *instance size* or *flavor*.

`security_group`

Security group to use when starting the instance.

`<class>_nodes`

the number of nodes of type `<class>`. These configuration options will define the composition of your cluster. A very common configuration will include only two group of nodes:

`frontend_nodes` the queue manager and frontend of the cluster. You probably want only one.

`compute_nodes` the worker nodes of the cluster.

Each `<class>_nodes` group is configured using the corresponding `<class>_groups` configuration option in the `[setup/...]` section.

`ssh_to`

ssh and *sftp* nodes will connect to only one node. This is the first of the group specified in this configuration option, or the first node of the first group in alphabetical order. For instance, if you don't set any value for *ssh_to* and you defined two groups: *frontend_nodes* and *compute_nodes*, the ssh and sftp command will connect to *compute001* which is the first *compute_nodes* node. If you specify *frontend*, instead, it will connect to *frontend001* (or the first node of the *frontend* group).

Optional configuration keys

`image_userdata`

shell script to be executed (as root) when the machine starts. This is usually not needed because the *ansible* provider works on *vanilla* images, but if you are using other setup providers you may need to execute some command to bootstrap it.

Examples

Some (working) examples:

```
[cluster/slurm]
cloud=hobbes
login=gc3-user
setup_provider=ansible-slurm
security_group=default
# Ubuntu image
image_id=ami-00000048
flavor=m1.small
frontend_nodes=1
compute_nodes=2
frontend_class=frontend

[cluster/torque]
cloud=hobbes
frontend_nodes=1
compute_nodes=2
frontend_class=frontend
security_group=default
# CentOS image
image_id=ami-0000004f
flavor=m1.small
login=gc3-user
setup_provider=ansible-pbs

[cluster/aws-slurm]
cloud=amazon-us-east-1
login=ubuntu
setup_provider=ansible-slurm
security_group=default
# ubuntu image
image_id=ami-90a21cf9
flavor=m1.small
frontend=1
compute=2

[cluster/matlab]
cloud=hobbes
setup_provider=ansible_matlab
security_group=default
image_id=ami-00000099
flavor=m1.medium
frontend_nodes=1
worker_nodes=10
image_userdata=
ssh_to=frontend
```

4.2.6 Cluster node section

A *cluster node* for the node type `<nodetype>` of the cluster `<name>` starts with:

```
[cluster/<name>/<nodetype>]
```

This section allows you to override some configuration values for specific group of nodes. Assume you have a standard slurm cluster with a frontend which is used as manager node and nfs server for the home directories, and a set of compute nodes.

You may want to use different flavors for the frontend and the compute nodes, since for the first you need more space and you don't need many cores or much memory, while the compute nodes may requires more memory and more cores but are not eager about disk space.

This is achieved defining, for instance, a *bigdisk* flavor (the name is just fictional) for the frontend and *8cpu32g* for the compute nodes. Your configuration will thus look like:

```
[cluster/slurm]
...
flavor=8cpu32g
frontend_nodes=1
compute_nodes=10

[cluster/slurm/frontend]
flavor=bigdisk
```

4.3 Usage

The syntax of the elasticluster command is:

```
elasticluster [-v] [-s PATH] [-c PATH] [subcommand] [subcommand args and opts]
```

The following options are general and are accepted by any subcommand:

`-h, --help` Show an help message and exits.

`-v, --verbose` Adding one or more *-v* will increase the verbosity accordingly.

`-s PATH, --storage PATH`

Path to the storage folder. This directory is used to store information about the cluster which are running. By default this is ``~/.elasticluster/storage`

WARNING: If you delete this directory elasticluster will not be able to access the cluster anymore!

`-c PATH, --config PATH`

Path to the configuration file. By default this is `~/.elasticluster/config`

elasticluster provides multiple *subcommands* to start, stop, resize, inspect your clusters. The available subcommands are:

start Create a cluster using one of the configured cluster tmplate.

stop Stop a cluster and all associated VM instances.

list List all clusters that are currently running.

list-nodes Show information about the nodes in a specific started cluster.

list-templates Show the available cluster configurations, as defined in the configuration file.

setup Run ansible to configure the cluster.

resize Resize a cluster by adding or removing nodes.

ssh Connect to the frontend of the cluster using the *ssh* command.

sftp Open an SFTP session to the cluster frontend host.

An help message explaining the available options and subcommand of *elasticluster* is available by running:

```
elasticluster -h
```

Options and arguments accepted by a specific subcommand *<cmd>* is available by running:

```
elasticluster <cmd> -h
```

4.3.1 The `start` command

This command will start a new cluster using a specific cluster configuration, defined in the configuration file. You can start as many clusters you want using the same cluster configuration, by providing different `--name` options.

Basic usage of the command is:

```
usage: elasticluster start [-h] [-v] [-n CLUSTER_NAME]
                           [--nodes N1:GROUP[,N2:GROUP2,...]] [--no-setup]
                           cluster
```

`cluster` is the name of a *cluster* section in the configuration file. For instance, to start the cluster defined by the section `[cluster/slurm]` you must run the command:

```
elasticluster start slurm
```

The following options are available:

`-h, --help` Show an help message and exits.

`-v, --verbose` Adding one or more *-v* will increase the verbosity accordingly.

`-n CLUSTER_NAME, --name CLUSTER_NAME` Name of the cluster. By default this is the same as the cluster configuration name.

`--nodes N1:GROUP[,N2:GROUP2,...]`

This option allow you to override the values stored in the configuration file, by starting a different number of hosts fore each group.

Assuming you defined, for instance, a cluster with the following type of nodes in the configuration file:

```
hadoop-data_nodes=4
hadoop-task_nodes=4
```

and you want to run instead 10 data nodes and 10 task nodes, you can run elasticluster with option:

```
elasticluster ... --nodes 10:hadoop-data,10:hadoop-task
```

`--no-setup` By default elasticluster will automatically run the **setup** command after all the virtual machines are up and running. This option prevent the *setup* step to be run and will leave the cluster unconfigured.

When you start a new cluster, elasticluster will:

- create the requested/configured number of virtual machines.
- wait until *all* the virtual machines are started.

- wait until *elasticluster* is able to connect to *all* the virtual machines using *ssh*.
- run ansible on all the virtual machines (unless `--no-setup` option is given).

This process can take several minutes, depending on the load of the cloud, the configuration of the cluster and your connection speed. *Elasticluster* usually print very few information on what's happening, if you run it with *-v* it will display a more verbose output (including output of ansible command) to help you understanding what is actually happening.

After the setup process is done a summary of the created cluster is printed, similar to the following:

```
Cluster name:     slurm
Cluster template: slurm
Frontend node: frontend001
- compute nodes: 2
- frontend nodes: 1

To login on the frontend node, run the command:

    elasticluster ssh slurm

To upload or download files to the cluster, use the command:

    elasticluster sftp slurm
```

The first line tells you the name of the cluster, which is the one you are supposed to use with the **stop**, **list-nodes**, **resize**, **ssh** and **sftp** commands.

The second line specifies the cluster configuration section used to configure the cluster (in this case, for instance, the section `[cluster/slurm]` has been used)

The `Frontend node` line shows which node is used for the **ssh** and **sftp** commands, when connecting to the cluster.

Then a list of how many nodes of each type have been started

The remaining lines describe how to connect to the cluster either by opening an interactive shell to run commands on it, or an sftp session to upload and download files.

4.3.2 The `stop` command

The **stop** command will terminate all the instances running and delete all information related to the cluster saved on the local disk.

WARNING: elasticluster doesn't do any kind of test to check if the cluster is *used*!

Basic usage of the command is:

```
usage: elasticluster stop [-h] [-v] [--force] [--yes] cluster
```

Like the **start** command, `cluster` is the name of a *cluster* section in the configuration file.

The following options are available:

`-h, --help` Show an help message and exits.

`-v, --verbose` Adding one or more *-v* will increase the verbosity accordingly.

`--force`

If some of the virtual machines fail to terminate (for instance because they have been terminated already not by elasticluster), this command will ignore these errors and will force termination of all the other instances.

`--yes`

Since stopping a cluster is a possibly desruptive action, elasticluster will always ask for confirmation before doing any modification, unless this option is given.

4.3.3 The `list` command

The **list** command print a list of all the cluster that have been started. For each cluster, it will print a few information including the cloud used and the number of nodes started for each node type:

```
$ elasticluster list

The following clusters have been started.
Please note that there's no guarantee that they are fully configured:

centossge
---------
  name:           centossge
  template:       centossge
  cloud:          hobbes
  - frontend nodes: 1
  - compute nodes: 2

slurm
-----
  name:           slurm
  template:       slurm
  cloud:          hobbes
  - frontend nodes: 1
  - compute nodes: 2

slurm13.04
----------
  name:           slurm13.04
  template:       slurm13.04
  cloud:          hobbes
  - frontend nodes: 1
  - compute nodes: 2
```

4.3.4 The `list-nodes` command

The **list-nodes** command print information on the nodes belonging to a specific cluster.

Basic usage of the command is:

```
usage: elasticluster list-nodes [-h] [-v] [-u] cluster
```

`cluster` is the name of a cluster that has been *started* previously.

The following options are available:

`-h, --help` Show an help message and exits.

`-v, --verbose` Adding one or more *-v* will increase the verbosity accordingly.

`-u, --update`

By default `elasticluster list-nodes` will not contact the EC2 provider to get up-to-date information, unless *-u* option is given.

Example:

```
$ elasticluster list-nodes centossge

Cluster name:     centossge
Cluster template: centossge
Frontend node: frontend001
- frontend nodes: 1
- compute nodes: 2

To login on the frontend node, run the command:

    elasticluster ssh centossge

To upload or download files to the cluster, use the command:

    elasticluster sftp centossge

frontend nodes:

  - frontend001
    public IP:   130.60.24.61
    private IP:  10.10.10.36
    instance id: i-0000299f
    instance flavor: m1.small

compute nodes:

  - compute001
    public IP:   130.60.24.44
    private IP:  10.10.10.17
    instance id: i-0000299d
    instance flavor: m1.small

  - compute002
    public IP:   130.60.24.48
    private IP:  10.10.10.29
    instance id: i-0000299e
    instance flavor: m1.small
```

4.3.5 The `list-templates` command

The **list-templates** command print a list of all the available templates defined in the configuration file with a few information for each one of them.

Basic usage of the command is:

```
usage: elasticluster list-templates [-h] [-v] [clusters [clusters ...]]
```

clusters is used to limit the clusters to be listed and uses a globbing-like pattern matching. For instance, to show all the cluster templates that contains the word `slurm` in their name you can run the following:

```
$ elasticluster list-templates *slurm*
11 cluster templates found.

name:     aws-slurm
cloud:     aws
compute nodes: 2
```

```
frontend nodes: 1

name:     slurm
cloud:     hobbes
compute nodes: 2
frontend nodes: 1

name:     slurm_xl
cloud:     hobbes
compute nodes: 2
frontend nodes: 1

name:     slurm13.04
cloud:     hobbes
compute nodes: 2
frontend nodes: 1
```

4.3.6 The `setup` command

The **setup** command will run *ansible* on the desired cluster once again. It is usually needed only when you customize and update your playbooks, in order to re-configure the cluster, since the **start** command already run *ansible* when all the machines are started.

Basic usage of the command is:

```
usage: elasticluster setup [-h] [-v] cluster
```

`cluster` is the name of a cluster that has been *started* previously.

The following options are available:

-h, --help Show an help message and exits.

-v, --verbose Adding one or more *-v* will increase the verbosity accordingly.

4.3.7 The `resize` command

The **resize** command allow you to add or remove nodes from a started cluster. Please, be warned that **this feature is still experimental**, and while adding nodes is usually safe, removing nodes can be desruptive and can leave the cluster in an unknwonw state.

Moreover, there is currently no way to decide *which nodes* can be removed from a cluster, therefore if you shrink a cluster **you must ensure** that any node of that type can be removed safely and no job is running on it.

When adding nodes, you have to specify the *type* of the node and the number of node you want to add. Then, elasticluster will basically re-run the *start* and *setup* steps:

- create the requested/configured number of virtual machines.
- wait until *all* the virtual machines are started.
- wait until *elasticluster* is able to connect to *all* the virtual machines using *ssh*.
- run ansible on all the virtual machines, including the virtual machines already configured (unless `--no-setup` option is given).

Growing a cluster (adding nodes to the cluster) should be supported by all the playbooks included in the elasticluster package.

Basic usage of the command is:

```
usage: elasticluster resize [-h] [-a N1:GROUP1[,N2:GROUP2]]
                            [-r N1:GROUP1[,N2:GROUP2]] [-v] [--no-setup]
                            [--yes]
                            cluster
```

`cluster` is the name of a cluster that has been *started* previously.

The following options are available:

-h, --help Show an help message and exits.

-v, --verbose Adding one or more *-v* will increase the verbosity accordingly.

`-a N1:GROUP1[,N2:GROUP2], --add N1:GROUP1[,N2:GROUP2]`

This option allow you to specify how many nodes for a specific group you want to add. You can specify multiple nodes separated by a comma.

Assuming you started, for instance, a cluster named *hadoop* using the default values stored in the configuration file:

```
hadoop-data_nodes=4
hadoop-task_nodes=4
```

and assuming you want to *add* 5 more data nodes and 10 more task nodes, you can run:

```
elasticluster resize -a 5:hadoop-data,10:hadoop-task
```

`-r N1:GROUP1[,N2:GROUP2], --remove N1:GROUP1[,N2:GROUP2]`

This option allow you to specify how many nodes you want to remove from a specific group. It follows the same syntax as the `--add` option.

WARNING: elasticluster pick the nodes to remove at random, so **you have to be sure that any of the nodes can be removed**. Moreover, not all the playbooks support shrkinging!

`--no-setup`

By default elasticluster will automatically run the **setup** command after starting and/or stopping the virtual machines. This option prevent the *setup* step to be run. **WARNING**: use this option wisely: depending on the cluster configuration it is impossible to know in advance what the status of the cluster will be after resizing it and NOT running the *setup* step.

`--yes`

Since resizing a cluster, especially shrinking, is a possibly desruptive action and is not supported by all the distributed playbooks, elasticluster will always ask for confirmation before doing any modification, unless this option is given.

4.3.8 The `ssh` command

After a cluster is started, the easiest way to login on it is by using the **ssh** command. This command will run the *ssh* command with the correct options to connect to the cluster using the configured values for user and ssh key to use.

If no `ssh_to` option is specified in the configuration file, the **ssh** command will connect to the first host belonging to the type which comes first in alphabetic order, otherwise it will connect to the first host of the group specified by the `ssh_to` option of the `cluster` section. However, running the command `elasticluster list-nodes <cluster>` will show which host will be used as frontend node.

The usage of the *ssh* command is as follow:

```
elasticluster ssh <clustername> [ -- ssh arguments]
```

All the options and arguments following the `--` characters will be passed directly to the `ssh` command.

For instance, if you just want to run the `hostname -f` command on the frontend of the cluster you can run:

```
elasticluster ssh <clustername> -- hostname -f
```

Note that since the IP address of the virtual machines are likely to be reused by different virtual machines, in order to avoid annoying warning messages from ssh elasticluster will add the following options to the *ssh* command line:

`-o UserKnownHostsFile=/dev/null` Use an empty virtual file to check the host key of the remote machine.

`-o StrictHostKeyChecking=no` Disable check of the host key of the remove machine, without prompting to ask if the key can be accepted or not.

4.3.9 The `sftp` command

After a cluster is started, the easiest way to upload or download files to and from the cluster is by using the **sftp** command. This command will run the *sftp* command with the correct options to connect to the cluster using the configured values for user and ssh key to use.

If no `ssh_to` option is specified in the configuration file, the **sftp** command will connect to the first host belonging to the type which comes first in alphabetic order, otherwise it will connect to the first host of the group specified by the `ssh_to` option of the `cluster` section. However, running the command `elasticluster list-nodes <cluster>` will show which host will be used as frontend node.

The usage of the *sftp* command is as follow:

```
elasticluster sftp <clustername> [ -- sftp arguments]
```

All the options and arguments following the `--` characters will be passed directly to the `sftp` command.

Note that since the IP address of the virtual machines are likely to be reused by different virtual machines, in order to avoid annoying warning messages from ssh elasticluster will add the following options to the *sftp* command line:

`-o UserKnownHostsFile=/dev/null` Use an empty virtual file to check the host key of the remote machine.

`-o StrictHostKeyChecking=no` Disable check of the host key of the remove machine, without prompting to ask if the key can be accepted or not.

4.4 Playbooks distributed with elasticluster

After the requested number of Virtual Machines have been started, elasticluster uses Ansible to configure them based on the configuration options defined in the configuration file.

We distribute a few playbooks together with elasticluster to configure some of the most wanted clusters. The playbooks are available at the `share/elasticluster/providers/ansible-playbooks/` directory inside your virtualenv if you installed using pip, or in the `elasticluster/providers/ansible-playbooks` directory of the github source code. You can copy, customize and redistribute them freely under the terms of the GPLv3 license.

A list of the most used playbooks distributed with elasticluster and some explanation on how to use them follows.

4.4.1 Slurm

Tested on:

- Ubuntu 12.04
- Ubuntu 13.04
- Debian 7.1 (GCE)

ansible groups	role
`slurm_master`	Act as scheduler and submission host
`slurm_clients`	Act as compute node

This playbook will install the SLURM queue manager using the packages distributed with Ubuntu and will create a basic, working configuration.

You are supposed to only define one `slurm_master` and multiple `slurm_clients`. The first will act as login node and will run the scheduler, while the others will only execute the jobs.

The `/home` filesystem is exported *from* the slurm server to the compute nodes.

A *snippet* of a typical configuration for a slurm cluster is:

```
[cluster/slurm]
frontend_nodes=1
compute_nodes=5
ssh_to=frontend
setup_provider=ansible_slurm
...

[setup/ansible_slurm]
frontend_groups=slurm_master
compute_groups=slurm_clients
...
```

You can combine the slurm playbooks with ganglia. In this case the `setup` stanza will look like:

```
[setup/ansible_slurm]
frontend_groups=slurm_master,ganglia_master
compute_groups=slurm_clients,ganglia_monitor
...
```

4.4.2 Gridengine

Tested on:

- Ubuntu 12.04
- CentOS 6.3 (except for GCE images)
- Debian 7.1 (GCE)

ansible groups	role
`gridengine_master`	Act as scheduler and submission host
`gridengine_clients`	Act as compute node

This playbook will install Grid Engine using the packages distributed with Ubuntu or CentOS and will create a basic, working configuration.

You are supposed to only define one `gridengine_master` and multiple `gridengine_clients`. The first will act as login node and will run the scheduler, while the others will only execute the jobs.

The `/home` filesystem is exported *from* the gridengine server to the compute nodes. If you are running on a CentOS, also the `/usr/share/gridengine/default/common` directory is shared from the gridengine server to the compute nodes.

A *snippet* of a typical configuration for a gridengine cluster is:

```
[cluster/gridengine]
frontend_nodes=1
compute_nodes=5
ssh_to=frontend
setup_provider=ansible_gridengine
...

[setup/ansible_gridengine]
frontend_groups=gridengine_master
compute_groups=gridengine_clients
...
```

You can combine the gridengine playbooks with ganglia. In this case the `setup` stanza will look like:

```
[setup/ansible_gridengine]
frontend_groups=gridengine_master,ganglia_master
compute_groups=gridengine_clients,ganglia_monitor
...
```

Please note that Google Compute Engine provides Centos 6.2 images with a non-standard kernel which is **unsupported** by the gridengine packages.

4.4.3 Ganglia

Tested on:

- Ubuntu 12.04
- CentOS 6.3
- Debian 7.1 (GCE)
- CentOS 6.2 (GCE)

ansible groups	role
`ganglia_master`	Run gmetad and web interface. It also run the monitor daemon.
`ganglia_monitor`	Run ganglia monitor daemon.

This playbook will install Ganglia monitoring tool using the packages distributed with Ubuntu or CentOS and will configure frontend and monitors.

You should run only one `ganglia_master`. This will install the `gmetad` daemon to collect all the metrics from the monitored nodes and will also run apache.

If the machine in which you installed `ganglia_master` has IP `10.2.3.4`, the ganglia web interface will be available at the address http://10.2.3.4/ganglia/

This playbook is supposed to be compatible with all the other available playbooks.

4.4.4 IPython cluster

Tested on:

- Ubuntu 12.04
- CentOS 6.3
- Debian 7.1 (GCE)

- CentOS 6.2 (GCE)

ansible groups	role
`ipython_controller`	Run an IPython cluster controller
`ipython_engine`	Run a number of ipython engine for each core

This playbook will install an IPython cluster to run python code in parallel on multiple machines.

One of the nodes should act as *controller* of the cluster (`ipython_controller`), running the both the *hub* and the *scheduler*. Other nodes will act as *engine*, and will run one "ipython engine" per core. You can use the *controller* node for computation too by assigning the `ipython_engine` class to it as well.

A *snippet* of typical configuration for an Hadoop cluster is:

```
[cluster/ipython]
setup_provider=ansible_ipython
controller_nodes=1
worker_nodes=4
ssh_to=controller
...

[setup/ansible_ipython]
controller_groups=ipython_controller,ipython_engine
worker_groups=ipython_engine
...
```

In order to use the IPython cluster, using the default configuration, you are supposed to connect to the controller node via ssh and run your code from there.

4.4.5 Hadoop

Tested on:

- Ubuntu 12.04
- CentOS 6.3
- Debian 7.1 (GCE)

ansible groups	role
`hadoop_namenode`	Run the Hadoop NameNode service
`hadoop_jobtracker`	Run the Hadoop JobTracker service
`hadoop_datanode`	Act as datanode for HDFS
`hadoop_tasktracker`	Act as tasktracker node accepting jobs from the JobTracker

Hadoop playbook will install a basic hadoop cluster using the packages available on the Hadoop website. The only supported version so far is **1.1.2 x86_64** and it works both on CentOS and Ubuntu.

You must define only one `hadoop_namenode` and one `hadoop_jobtracker`. Configuration in which both roles belong to the same machines are not tested. You can mix `hadoop_datanode` and `hadoop_tasktracker` without problems though.

A *snippet* of a typical configuration for an Hadoop cluster is:

```
[cluster/hadoop]
hadoop-name_nodes=1
hadoop-jobtracker_nodes=1
hadoop-task-data_nodes=10
setup_provider=ansible_hadoop
ssh_to=hadoop-name
...
```

```
[setup/ansible_hadoop]
hadoop-name_groups=hadoop_namenode
hadoop-jobtracker_groups=hadoop_jobtracker
hadoop-task-data_groups=hadoop_tasktracker,hadoop_datanode
...
```

4.4.6 GlusterFS

Tested on:

- Ubuntu 12.04
- CentOS 6.3

ansible groups	role
`gluster_data`	Run a gluster *brick*
`gluster_client`	Install gluster client and install a gluster filesystem on `/glusterfs`

This will install a GlusterFS using all the `gluster_data` nodes as *bricks*, and any `gluster_client` to mount this filesystem in `/glusterfs`.

Setup is very basic, and by default no replicas is set.

To manage the gluster filesystem you need to connect to a `gluster_data` node.

4.4.7 OrangeFS/PVFS2

Tested on:

- Ubuntu 12.04

ansible groups	role
`pvfs2_meta`	Run the pvfs2 metadata service
`pvfs2_data`	Run the pvfs2 data node
`pvfs2_client`	configure as pvfs2 client and mount the filesystem

The OrangeFS/PVFS2 playbook will configure a pvfs2 cluster. It downloads the software from the OrangeFS website, compile and install it on all the machine, and run the various server and client daemons.

In addiction, it will mount the filesystem in `/pvfs2` on all the clients.

You can combine, for instance, a SLURM cluster with a PVFS2 cluster:

```
[cluster/slurm+pvfs2]
frontend_nodes=1
compute_nodes=10
pvfs2-nodes=10
ssh_to=frontend
setup_provider=ansible_slurm+pvfs2
...

[setup/ansible_slurm+pvfs2]
frontend_groups=slurm_master,pvfs2_client
compute_groups=slurm_clients,pvfs2_client
pvfs-nodes_groups=pvfs2_meta,pvfs2_data
...
```

This configuration will create a SLURM cluster with 10 compute nodes, 10 data nodes and a frontend, and will mount the `/pvfs2` directory from the data nodes to both the compute nodes and the frontend.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search