elasticluster Documentation

Release 1.0

Grid Computing Competence Centre, University of Zurich

April 15, 2016

Contents

CHAPTER 1

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, Google Compute Engine or an 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.

In both cases, it's strongly suggested to install *elasticluster* in a python virtualenv, so that you can easily uninstall or upgrade *elasticluster*.

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
python setup.py install
```

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

4.1.3 Notes on MacOSX installation

When installing *elasticluster* on MacOSX operating systems you may get some errors while running *python setup.py install*, because pip is not always able to automatically resolve the dependencies.

In these cases, you need to find the package that is failing and install it manually using *pip*.

For instance, if during the installation you get something like:

```
Running requests-2.4.3/setup.py -q bdist_egg --dist-dir /var/folders/tZ/tZ2B3RaeGVq7+ptdJIbdj++++TI/-
Adding requests 2.4.3 to easy-install.pth file
Traceback (most recent call last):
  File "setup.py", line 109, in <module>
    'elasticluster = elasticluster.main:main',
  File "/System/Library/Frameworks/Python.framework/Versions/2.6/lib/python2.6/distutils/core.py", li
    dist.run_commands()
  File "/System/Library/Frameworks/Python.framework/Versions/2.6/lib/python2.6/distutils/dist.py", li
    self.run_command(cmd)
  File "/System/Library/Frameworks/Python.framework/Versions/2.6/lib/python2.6/distutils/dist.py", li
    cmd_obj.run()
  File "/Users/michela/elasticluster/build/setuptools/setuptools/command/install.py", line 65, in run
  File "/Users/michela/elasticluster/build/setuptools/setuptools/command/install.py", line 115, in do
  File "/Users/michela/elasticluster/build/setuptools/setuptools/command/easy_install.py", line 360,

  File "/Users/michela/elasticluster/build/setuptools/setuptools/command/easy_install.py", line 576,

  File "/Users/michela/elasticluster/build/setuptools/setuptools/command/easy_install.py", line 627,

  File "/Users/michela/elasticluster/build/setuptools/setuptools/command/easy_install.py", line 682,

  File "/Users/michela/elasticluster/build/setuptools/pkg_resources.py", line 631, in resolve
    dist = best[req.key] = env.best_match(req, ws, installer)
  File "/Users/michela/elasticluster/build/setuptools/pkg_resources.py", line 871, in best_match
    return self.obtain(req, installer)
  File "/Users/michela/elasticluster/build/setuptools/pkg_resources.py", line 883, in obtain
    return installer(requirement)
  File "/Users/michela/elasticluster/build/setuptools/setuptools/command/easy_install.py", line 595,

  File "/Users/michela/elasticluster/build/setuptools/setuptools/command/easy_install.py", line 627,

  File "/Users/michela/elasticluster/build/setuptools/setuptools/command/easy_install.py", line 659,

  File "/Users/michela/elasticluster/build/setuptools/setuptools/command/easy_install.py", line 532,

  File "/Users/michela/elasticluster/build/setuptools/setuptools/command/easy_install.py", line 734,

  File "/private/var/folders/tZ/tZ2B3RaeGVq7+ptdJIbdj++++TI/-Tmp-/easy_install-qch0dG/python-keystone
AttributeError: 'NoneType' object has no attribute 'get_script_header'
```

you probably need to install *pbr* manually using:

```
pip install pbr
```

In some MacOSX version, even if the installation *seems* to succeed, you may get the following error the first time you run *elasticluster*:

```
Traceback (most recent call last):
  File "/Users/michela/el2/bin/elasticluster", line 9, in <module>
    load_entry_point('elasticluster==1.1-dev', 'console_scripts', 'elasticluster')()
  File "/Users/michela/el2/build/setuptools/pkg_resources.py", line 356, in load_entry_point
    return get_distribution(dist).load_entry_point(group, name)
  File "/Users/michela/el2/build/setuptools/pkg_resources.py", line 2431, in load_entry_point
    return ep.load()
  File "/Users/michela/el2/build/setuptools/pkg_resources.py", line 2147, in load
    ['__name__'])
  File "build/bdist.macosx-10.6-universal/egg/elasticluster/__init__.py", line 33, in <module>
  File "build/bdist.macosx-10.6-universal/egg/elasticluster/providers/gce.py", line 37, in <module>
  File "build/bdist.macosx-10.6-universal/egg/apiclient/discovery.py", line 52, in <module>
  File "build/bdist.macosx-10.6-universal/egg/apiclient/errors.py", line 27, in <module>
ImportError: No module named anyjson
```

In this case, the issue is caused by *google-api-python-client*, and you should: 1) uninstall it using *pip uninstall* 2) reinstall it using *pip install* 3) re-run elasticluster installation

```
pip uninstall google-api-python-client
[...]
pip install google-api-python-client
[...]
python setup.py install
```

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.

Therefore, after installing *elasticluster* for the first time, we suggest to run the following command:

```
elasticluster list-templates
```

If you don't already have a configuration file, this command will create one for you. Of course, the file is not complete, as it does not contain any authentication information, and you will get an error similar to the following:

```
WARNING:gc3.elasticluster:Deploying default configuration file to /home/antonio/.elasticluster/config
WARNING:gc3.elasticluster:Ignoring Cluster `ipython`: required key not provided @ data['image_user']
WARNING:gc3.elasticluster:Ignoring cluster `ipython`.
Error validating configuration file '/home/antonio/.elasticluster/config': `required key not provided
```

In this case, you have to edit the configuration file in `~/.elasticluster/config` and update it with the correct values.

Please refer to the following section to understand the syntax of the configuration file and to know which options you need to set in order to use *elasticluster*.

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. *ec2_boto*, *openstack* 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.

`request_floating_ip`

request assignment of a floating IP when the instance is started. Valid values: *True*, *False*. Some cloud providers does not automatically assign a public IP to the instances, but this is often needed if you want to connect to the VM from outside. Setting `request_floating_ip` to *True* will force *elasticluster* to request such a floating IP if the instance doesn't get one automatically.

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

Valid configuration keys for *openstack*

`auth_url`:

The URL of the keystone service (main entry point for OpenStack clouds)

`username`

OpenStack username

`password`

OpenStack password

`project_name`

OpenStack project to use (also known as *tenant*)

`region_name`

OpenStack region (optional)

`request_floating_ip`

request assignment of a floating IP when the instance is started. Valid values: *True*, *False*. Some cloud providers does not automatically assign a public IP to the instances, but this is often needed if you want to connect to the VM from outside. Setting `request_floating_ip` to *True* will force *elasticluster* to request such a floating IP if the instance doesn't get one automatically.

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
auto_ip_assignment=True
```

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 into 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.

`network_ids`

comma separated list of network IDs the nodes of the cluster will be connected to. Only supported when the cloud provider is *openstack*

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. Usually elasticluster creates new VMs in parallel, to speedup the process, but if you run it with at least *four -v* options, elasticluster will not fork and will start the VMs sequentially. Useful for debugging.

`-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 HTCondor

Tested on:

- Ubuntu 12.04

ansible groups	role
`condor_master`	Act as scheduler, submission and execution host.
`condor_workers`	Act as execution host only.

This playbook will install the HTCondor workload management system using the packages provided by the Center for High Throughput Computing at UW-Madison.

The `/home` filesystem is exported *from* the condor master to the compute nodes.

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

```
[cluster/condor]
setup_provider=ansible_condor
frontend_nodes=1
compute_nodes=2
ssh_to=frontend
...

[setup/ansible_condor]
frontend_groups=condor_master
compute_groups=condor_workers
...
```

4.4.4 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.

4.4.5 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.

4.4.6 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
...
```

4.4.7 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.

4.4.8 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.

4.5 Elasitcluster programming API

4.5.1 *elasticluster*

Overview

Elasticluster offers an API to programmatically manage compute clusters on cloud infrastructure. This page introduces the basic concepts of the API and provides sample code to illustrate the usage of the API. While this document should provide you with the basics, more details can be found in the respective module documentation

Getting Started

The following subchapters introduce the basic concepts of Elasticluster.

Cluster

This is the heart of elasticluster and handles all cluster relevant behavior. You can basically start, setup and stop a cluster. Also it provides factory methods to add nodes to the cluster. A typical workflow is as follows (see *slurm* for a code example):

1. create a new cluster
2. add nodes to fit your computing needs
3. start cluster; start all instances in the cloud
4. setup cluster; configure all nodes to fit your computing cluster
5. ssh into a node to submit computing jobs
6. eventually stop cluster; destroys all instances in the cloud

See documentation of the `Cluster` class for futher details.

Node

The node represents an instance in a cluster. It holds all information to connect to the nodes also manages the cloud instance. It provides the basic functionality to interact with the cloud instance, such as start, stop, check if the instance is up and ssh connect.

See the `Node` api docs for further details.

Cloud Provider

Manages the connection to the cloud webservice and offers all functionality used by the cluster to provision instances. Elasticluster offers two different cloud providers at the current state:

- ***BotoCloudProvider*** Cloud provider to connect to EC2 compliant web services (e.g Amazon, Openstack, etc.)
- ***GoogleCloudProvider*** **Cloud** provider to connect to the Google Compute Engine (GCE)

All listed cloud providers above can be used to manage a cluster in the cloud. If the cloud operator is not supported by the implementations above, an alternative implementation can be provided by following the *AbstractCloudProvider* contract.

Setup Provider

The setup provider configures in respect to the specified cluster and node configuration. The basic implementation *AnsibleSetupProvider* uses ansible to configure the nodes. Ansible is a push based configuration management system in which the configuration is stored locally and pushed to all the nodes in the cluster.

See the *Playbooks distributed with elasticluster* page for more details on the cluster setups possible with the ansible implementation and how the ansible playbooks can be enhanced.

If this implementation does not satisfy the clients needs, an alternative implementation can be implemented following the *AbstractSetupProvider* contract.

Cluster Repository

The cluster repository is responsible to keep track of multiple clusters over time. Therefore Elasticluster provides two implementations:

- ***MemRepository*** Stores the clusters in memory. Therefore after stopping a program using this repository, all clusters are not recoverable but possibly still running.
- ***ClusterRepository*** Stores the cluster on disk persistently. This implementation uses pickle to serialize and deserialize the cluster.

If a client wants to store the cluster in a database for example, an alternative implementation can be provided following the *AbstractClusterRepository* contract.

Sample Code

Start and setup a SLURM cluster

The following sample code shows how to start and setup a SLURM cluster on an OpenStack cloud and provides further information on each step. Other cluster types on other cloud providers can be setup accordingly.

```
import elasticluster

# Initialise an EC2 compatible cloud provider, in this case an OpenStack
# cloud operator is chosen. To initialise the cloud provider the
# following parameters are passed:
#   url:        url to connecto to the cloud operator web service
#   region:     region to start the nodes on
#   access_key: access key of the current user to connect
#   secret_key: secret key of the current user to connect
cloud_provider = elasticluster.BotoCloudProvider(
                                        'http://uzh.ch/services/Cloud',
                                        'nova', 'access_key', 'secret_key')

# Initialising the setup provider needs a little more preparation:
```

```
# the groups dictionary specifies the kind of nodes used for this cluster.
# In this case we want a frontend and a compute kind. The frontend node
# (s) will be setup as slurm_master, the compute node(s) as slurm_clients.
# This corresponds to the documentation of the ansible playbooks
# provided with elasticluster. The kind of the node is a name specified
# by the user. This name will be used to set a new hostname on the
# instance, therefore it should meet the requirements of RFC 953
# groups['kind'] = ['andible_group1', 'ansible_group2']
groups = dict()
groups['frontend'] = ['slurm_master']
groups['compute'] = ['slurm_clients']

setup_provider = elasticluster.AnsibleSetupProvider(groups)

# cluster initialisation (note: ssh keys are same for all nodes)
# After the steps above initialising an empty cluster is a peace of cake.
# The cluster takes the following arguments:
# name:           name to identify the cluster
#   cloud_provider: cloud provider to connect to cloud
#   setup_provider: setup provider to configure the cluster
#   ssh_key_name:   name of the ssh key stored (or to be stored) on the
#                   cloud
#   ssh_key_pub:    path to public ssh key file
#   ssh_key_priv:   path to private ssh key file
#
# The ssh key files are used for all instances in this cluster.
cluster = elasticluster.Cluster('my-cluster', cloud_provider,
                                setup_provider, 'ssh_key_name',
                                '~/ssh/keys/my_ssh_key.pub',
                                '~/ssh/keys/my_ssh_key')

# To add nodes to the cluster we can use the add_node. This
# only initialises a new node, but does not start it yet.
# The add node function is basically a factory method to make it easy to
# add nodes to a cluster. It takes the following arguments:
#   kind:   kind of the node in this cluster. This corresponds to the
#           groups defined in the cloud_provider.
cluster.add_node('frontend', 'ami-00000048', 'gc3-user',
                 'm1.tiny', 'all_tcp_ports')

# We can also add multiple nodes with the add_nodes method.
# The following command will add 2 nodes of the kind `compute` to the
# cluster
cluster.add_nodes('compute', 2, 'ami-00000048', 'gc3-user', 'm1.tiny',
                  'all_tcp_ports')

# Since we initialised all the nodes for this computing cluster,
# we can finally start the cluster.
# The start method is blocking and does the following tasks:
#   * call the cloud provider to start an instance for each node in a
#     seperate thread.
#   * to make sure elasticluster is not stopped during creation of an
#     instance, it will overwrite the sigint handler
#   * waits until all nodes are alive (meaning ssh connection
#     works)
#   * If the startup timeout is reached and not all nodes are alive,
#     the cluster will stop and destroy all instances
cluster.start()
```

```
# Now, all the nodes are started and we can call the setup method to
# configure slurm on the nodes.
cluster.setup()
```

Asynchronous node start

The *start()* method of the *Cluster()* class is blocking and therefore waits until all nodes are alive. If a client wants to use this time for other tasks, the nodes can as well be started asynchronous:

```
# retrieve all nodes from the cluster
nodes = cluster.get_all_nodes()

# start each node
# The start method on the node is non blocking and will return as soon
# as the cloud provider is contacted to start a new instance
for node in nodes:
    node.start()

# wait until all nodes are alive
starting_nodes = nodes[:]
while starting_nodes:
    starting_nodes = [n for n in starting_nodes if not n.is_alive()]
```

Storing a cluster on disk

By default elasticluster will store the cluster in memory only. Therefore after a programm shutdown the cluster will not be available anymore in elasticluster, but might still be running on the cloud. The following example shows how to store clusters on disk to retrieve after a programm restart:

```
# The cluster repository uses pickle to store clusters each in a
# seperate file in the provided storage directory.
repository = elasticluster.ClusterRepository('/path/to/storage/dir')

# On cluster initialisation we can pass the repository as optional
# argument.
cluster = elasticluster.Cluster('my-cluster', cloud_provider,
                                setup_provider, 'ssh_key_name',
                                '~/ssh/keys/my_ssh_key.pub',
                                '~/ssh/keys/my_ssh_key',
                                repository=repository)

# When starting the cluster, it will safe its state using the repository.
cluster.start()
```

After a program shutdown we can therefore fetch the cluster from the repository again and work with it as expected:

```
repository = elasticluster.ClusterRepository('/path/to/storage/dir')

# retrieve the cluster from the repository
cluster = repository.get('my-cluster')

# or retrieve all clusters that are stored in the repository
clusters = repository.get_all()
```

Logging

Elasticluster uses the python *logging* module to log events. A client can overwrite the settings as illustrated below:

```
import logging

import elasticluster

log = elasticluster.log
level = logging.getLevelName('INFO')
log.setLevel(level)
```

The current example only shows how to increase the log level, but any settings can be applied compliant with the logging module of python.

4.5.2 *elasticluster.cluster*

class `elasticluster.cluster.`**`Cluster`**(*name*, *cloud_provider*, *setup_provider*, *user_key_name*, *user_key_public*, *user_key_private*, *repository=None*, ***extra*)

This is the heart of elasticluster and handles all cluster relevant behavior. You can basically start, setup and stop a cluster. Also it provides factory methods to add nodes to the cluster. A typical workflow is as follows:

- create a new cluster
- add nodes to fit your computing needs
- start cluster; start all instances in the cloud
- setup cluster; configure all nodes to fit your computing cluster
- eventually stop cluster; destroys all instances in the cloud

Parameters

- **`name`** (*str*) – unique identifier of the cluster
- **`cloud_provider`** (*elasticluster.providers.AbstractCloudProvider*) – access to the cloud to manage nodes
- **`setup_provider`** (*elasticluster.providers.AbstractSetupProvider*) – provider to setup cluster
- **`user_key_name`** (*str*) – name of the ssh key to connect to cloud
- **`user_key_public`** (*str*) – path to ssh public key file
- **`user_key_private`** (*str*) – path to ssh private key file
- **`repository`** (*elasticluster.repository.AbstractClusterRepository*) – by default the *elasticluster.repository.MemRepository* is used to store the cluster in memory. Provide another repository to store the cluster in a persistent state.
- **`extra`** – tbd.

Variables **`nodes`** – dict [node_type] = [*Node*] that represents all nodes in this cluster

`add_node`(*kind*, *image_id*, *image_user*, *flavor*, *security_group*, *image_userdata=''*, *name=None*, ***extra*)

Adds a new node to the cluster. This factory method provides an easy way to add a new node to the cluster

by specifying all relevant parameters. The node does not get started nor setup automatically, this has to be done manually afterwards.

Parameters

- **kind** (*str*) – kind of node to start. this refers to the groups defined in the ansible setup provider `elasticluster.providers.AnsibleSetupProvider` Please note that this must match the *[a-zA-Z0-9-]* regexp, as it is used to build a valid hostname
- **image_id** (*str*) – image id to use for the cloud instance (e.g. ami on amazon)
- **image_user** (*str*) – user to login on given image
- **flavor** (*str*) – machine type to use for cloud instance
- **security_group** (*str*) – security group that defines firewall rules to the instance
- **image_userdata** (*str*) – commands to execute after instance starts
- **name** (*str*) – name of this node, automatically generated if None

Raises ValueError: *kind* argument is an invalid string.

Returns created *Node*

add_nodes(*kind*, *num*, *image_id*, *image_user*, *flavor*, *security_group*, *image_userdata=''*, ***extra*)
Helper method to add multiple nodes of the same kind to a cluster.

Parameters

- **kind** (*str*) – kind of node to start. this refers to the groups defined in the ansible setup provider `elasticluster.providers.AnsibleSetupProvider`
- **num** (*int*) – number of nodes to add of this kind
- **image_id** (*str*) – image id to use for the cloud instance (e.g. ami on amazon)
- **image_user** (*str*) – user to login on given image
- **flavor** (*str*) – machine type to use for cloud instance
- **security_group** (*str*) – security group that defines firewall rules to the instance
- **image_userdata** (*str*) – commands to execute after instance starts

get_all_nodes()
Returns a list of all nodes in this cluster as a mixed list of different node kinds.

Returns list of *Node*

get_frontend_node()
Returns the first node of the class specified in the configuration file as *ssh_to*, or the first node of the first class in alphabetic order.

Returns *Node*

Raise `elasticluster.exceptions.NodeNotFound` if no valid frontend node is found

remove_node(*node*)
Removes a node from the cluster, but does not stop it. Use this method with caution.

Parameters **node** (*Node*) – node to remove

setup()
Configure the cluster nodes with the specified This is delegated to the provided *elasticluster.providers.AbstractSetupProvider*

Returns bool - True on success, False otherwise

start (*min_nodes=None*)
Starts up all the instances in the cloud. To speed things up all instances are started in a seperate thread. To make sure elasticluster is not stopped during creation of an instance, it will overwrite the sigint handler. As soon as the last started instance is returned and saved to the repository, sigint is executed as usual. An instance is up and running as soon as a ssh connection can be established. If the startup timeout is reached before all instances are started, the cluster will stop and destroy all instances.

This method is blocking and might take some time depending on the amount of instances to start.

Parameters **min_nodes** (*dict [node_kind] = number*) – minimum number of nodes to start in case the quota is reached before all instances are up

startup_timeout = 600
timeout in seconds to start all nodes

stop (*force=False*)
Destroys all instances of this cluster and calls delete on the repository.

Parameters **force** (*bool*) – force termination of instances in any case

update ()
Update all connection information of the nodes of this cluster. It occurs for example public ip's are not available imediatly, therefore calling this method might help.

class elasticluster.cluster.**Node** (*name*, *cluster_name*, *kind*, *cloud_provider*, *user_key_public*, *user_key_private*, *user_key_name*, *image_user*, *security_group*, *image*, *flavor*, *image_userdata=None*, ***extra*)
The node represents an instance in a cluster. It holds all information to connect to the nodes also manages the cloud instance. It provides the basic functionality to interact with the cloud instance, such as start, stop, check if the instance is up and ssh connect.

Parameters

- **name** (*str*) – identifier of the node
- **kind** (*str*) – kind of node in regard to cluster. this usually refers to a specified group in the *elasticluster.providers.AbstractSetupProvider*
- **cloud_provider** (*elasticluster.providers.AbstractCloudProvider*) – cloud provider to manage the instance
- **user_key_public** (*str*) – path to the ssh public key
- **user_key_private** (*str*) – path to the ssh private key
- **user_key_name** (*str*) – name of the ssh key
- **image_user** (*str*) – user to connect to the instance via ssh
- **security_group** (*str*) – security group to setup firewall rules
- **image** (*str*) – image id to launch instance with
- **flavor** (*str*) – machine type to launch instance
- **image_userdata** (*str*) – commands to execute after instance start

Variables

- **instance_id** – id of the node instance on the cloud
- **preferred_ip** – IP address used to connect to the node.
- **ips** – list of all the IPs defined for this node.

connect()
Connect to the node via ssh using the paramiko library.

Returns `paramiko.SSHClient` - ssh connection or None on failure

connection_ip()
Returns the IP to be used to connect to this node.

If the instance has a public IP address, then this is returned, otherwise, its private IP is returned.

connection_timeout = 5
timeout in seconds to connect to host via ssh

is_alive()
Checks if the current node is up and running in the cloud. It only checks the status provided by the cloud interface. Therefore a node might be running, but not yet ready to ssh into it.

pprint()
Pretty print information about the node.

Returns str - representaion of a node in pretty print

start()
Starts the node on the cloud using the given instance properties. This method is non-blocking, as soon as the node id is returned from the cloud provider, it will return. Therefore the *is_alive* and *update_ips* methods can be used to further gather details about the state of the node.

stop()
Destroys the instance launched on the cloud for this specific node.

update_ips()
Retrieves the public and private ip of the instance by using the cloud provider. In some cases the public ip assignment takes some time, but this method is non blocking. To check for a public ip, consider calling this method multiple times during a certain timeout.

4.5.3 *elasticluster.conf*

class `elasticluster.conf.`**ConfigReader**(*configfile*)
Reads the configuration properties from a ini file.

Parameters **configfile** (*str*) – path to configfile

read_config()
Reads the configuration properties from the ini file and links the section to comply with the cluster config dictionary format.

Returns dictionary containing all configuration properties from the ini file in compliance to the cluster config format

Raises `voluptuous.MultipleInvalid` if not all sections present or broken links between secitons

class `elasticluster.conf.`**ConfigValidator**(*config*)
Validator for the cluster configuration dictionary.

Parameters **config** – dictionary containing cluster configuration properties

validate()
Validates the given configuration `self.config` to comply with elasticluster. As well all types are converted to the expected format if possible.

Raisegs `voluptuous.MultipleInvalid` if multiple properties are not compliant

Raises `voluptuous.Invalid` if one property is invalid

class `elasticluster.conf.`**Configurator**(*cluster_conf*, *storage_path=None*)

The Configurator is responsible for (I) keeping track of the configuration and (II) offer factory methods to create all kind of objects that need information from the configuration.

The cluster configuration dictionary is structured in the following way: (see an example @

https://github.com/gc3-uzh-ch/elasticluster/wiki/Configuration-Module)

```
{ "<cluster_template>" : {
    "setup" : { properties of the setup section },
    "cloud" : { properties of the cloud section },
    "login" : { properties of the login section },
    "cluster" : { properties of the cluster section },
    "nodes": {  "<node_kind>" : { properties of the node},
                "<node_kind>" : { properties of the node},
            },
    },
 "<cluster_template>" : {
    (see above)
    }
 }
```

Parameters

- **cluster_conf** (*dict*) – see description above
- **storage_path** (*str*) – path to store data

Raises **MultipleInvalid** – configuration validation

create_cloud_provider(*cluster_template*)

Creates a cloud provider by inspecting the configuration properties of the given cluster template.

Parameters **cluster_template** (*str*) – template to use (if not already specified on init)

Returns cloud provider that fulfills the contract of *elasticluster.providers.AbstractSetupProvider*

create_cluster(*template*, *name=None*)

Creates a cluster by inspecting the configuration properties of the given cluster template.

Parameters

- **template** (*str*) – name of the cluster template
- **name** (*str*) – name of the cluster. If not defined, the cluster

will be named after the template.

Returns `elasticluster.cluster.cluster` instance

Raises **ConfigurationError** – cluster template not found in config

create_setup_provider(*cluster_template*, *name=None*)

Creates the setup provider for the given cluster template.

Parameters

- **cluster_template** (*str*) – template of the cluster
- **name** (*str*) – name of the cluster to read configuration properties

classmethod **fromConfig**(*configfile*, *storage_path=None*)

Helper method to initialize Configurator from an ini file.

Parameters **configfile** (*str*) – path to the ini file

Returns *Configurator*

load_cluster(*cluster_name*)
Loads a cluster from the cluster repository.

Parameters **cluster_name** (*str*) – name of the cluster

Returns elasticluster.cluster.cluster instance

4.5.4 elasticluster.exceptions

4.5.5 elasticluster.gc3_config

4.5.6 elasticluster.helpers

class elasticluster.helpers.**Singleton**(*decorated*)
see: http://stackoverflow.com/a/7346105 A non-thread-safe helper class to ease implementing singletons. This should be used as a decorator – not a metaclass – to the class that should be a singleton.

The decorated class can define one *__init__* function that takes only the *self* argument. Other than that, there are no restrictions that apply to the decorated class.

To get the singleton instance, use the *Instance* method. Trying to use *__call__* will result in a *TypeError* being raised.

Limitations: The decorated class cannot be inherited from.

Instance()
Returns the singleton instance. Upon its first call, it creates a new instance of the decorated class and calls its *__init__* method. On all subsequent calls, the already created instance is returned.

4.5.7 elasticluster.main

4.5.8 elasticluster.module

4.5.9 elasticluster.providers

class elasticluster.providers.**AbstractCloudProvider**(***config*)
Defines the contract for a cloud provider to proper function with elasticluster.

get_ips(*instance_id*)
Retrieves the private and public ip addresses for a given instance.

Returns list (IPs)

is_instance_running(*instance_id*)
Checks if the instance is up and running.

Parameters **instance_id** (*str*) – instance identifier

Returns bool - True if running, False otherwise

start_instance(*key_name*, *public_key_path*, *private_key_path*, *security_group*, *flavor*, *image_id*, *image_userdata*, *username=None*, *node_name=None*)
Starts a new instance on the cloud using the given properties. Multiple instances might be started in different threads at the same time. The implementation should handle any problems regarding this itself.

Parameters

- **key_name** (*str*) – name of the ssh key to connect
- **public_key_path** (*str*) – path to ssh public key
- **private_key_path** (*str*) – path to ssh private key
- **security_group** (*str*) – firewall rule definition to apply on the instance
- **flavor** (*str*) – machine type to use for the instance
- **image_name** (*str*) – image type (os) to use for the instance
- **image_userdata** (*str*) – command to execute after startup
- **username** (*str*) – username for the given ssh key, default None

Returns str - instance id of the started instance

stop_instance(*instance_id*)
Stops the instance gracefully.

Parameters **instance_id** (*str*) – instance identifier

Returns None

class elasticluster.providers.**AbstractSetupProvider**
TODO: define...

cleanup()
Cleanup any temporary file or directory created during setup. This method is called every time a cluster is stopped.

Returns None

setup_cluster(*cluster*)
Configures all nodes of a cluster to function in respect to the given configuration.

This method *must* be idempotent, i.e. it should always be safe calling it multiple times..

Parameters **cluster** (*elasticluster.cluster.Cluster*) – cluster to configure

Returns *True* if the cluster is correctly configured, even if the method didn't actually do anything. *False* if the cluster is not configured.

4.5.10 elasticluster.providers.ansible_provider

class elasticluster.providers.ansible_provider.**AnsibleSetupProvider**(*groups*, *playbook_path=None*, *environment_vars={}*, *storage_path=None*, *sudo=True*, *sudo_user='root'*, *ansible_module_dir=None*, ***extra_conf*)
This implementation uses ansible to configure and manage the cluster setup. See https://github.com/ansible/ansible for details.

Parameters

- **groups** (*dict*) – dictionary of node kinds with corresponding ansible groups to install on the node kind. e.g [node_kind] = ['ansible_group1', 'ansible_group2'] The group defined here can be references in each node. Therefore groups can make it easier to define multiple groups for one node.
- **playbook_path** (*str*) – path to playbook; if empty this will use the shared playbook of elasticluster
- **environment_vars** (*dict*) – dictonary to define variables per node kind, e.g. [node_kind][var] = value
- **storage_path** (*str*) – path to store the inventory file. By default the inventory file is saved temporarily in a temporary directory and deleted when the cluster in stopped.
- **sudo** (*bool*) – indication whether use sudo to gain root permission
- **sudo_user** (*str*) – user with root permission
- **ansible_module_dir** (*str*) – path to addition ansible modules
- **extra_conf** – tbd.

Variables

- **groups** – node kind and ansible group mapping dictionary
- **environment** – additional environment variables

cleanup(*cluster*)

Deletes the inventory file used last recently used.

Parameters **cluster** (*elasticluster.cluster.Cluster*) – cluster to clear up inventory file for

setup_cluster(*cluster*)

Configures the cluster according to the node_kind to ansible group matching. This method is idempotent and therefore can be called multiple times without corrupting the cluster configuration.

Parameters **cluster** (*elasticluster.cluster.Cluster*) – cluster to configure

Returns True on success, False otherwise. Please note, if nothing has to be configures True is returned

Raises *AnsibleError* if the playbook can not be found or playbook is corrupt

4.5.11 *elasticluster.providers.ec2_boto*

class elasticluster.providers.ec2_boto.**BotoCloudProvider**(*ec2_url*, *ec2_region*, *ec2_access_key*, *ec2_secret_key*, *storage_path=None*, *request_floating_ip=False*)

This implementation of *elasticluster.providers.AbstractCloudProvider* uses the boto ec2 interface to connect to ec2 compliant clouds and manage instances.

Please check https://github.com/boto/boto for further information about the supported cloud platforms.

Parameters

- **ec2_url** (*str*) – url to connect to cloud web service
- **ec2_region** (*str*) – region identifier

- **ec2_access_key** (*str*) – access key of the user account
- **ec2_secret_key** (*str*) – secret key of the user account
- **storage_path** (*str*) – path to store temporary data
- **request_floating_ip** (*bool*) – Whether ip are assigned automatically *True* or floating ips have to be assigned manually *False*

get_ips(*instance_id*)
Retrieves the private and public ip addresses for a given instance.

Returns list (ips)

is_instance_running(*instance_id*)
Checks if the instance is up and running.

Parameters **instance_id** (*str*) – instance identifier

Returns bool - True if running, False otherwise

start_instance(*key_name*, *public_key_path*, *private_key_path*, *security_group*, *flavor*, *image_id*, *image_userdata*, *username=None*, *node_name=None*, ***kwargs*)
Starts a new instance on the cloud using the given properties. The following tasks are done to start an instance:

•establish a connection to the cloud web service

•check ssh keypair and upload it if it does not yet exist. This is a locked process, since this function might be called in multiple threads and we only want the key to be stored once.

•check if the security group exists

•run the instance with the given properties

Parameters

- **key_name** (*str*) – name of the ssh key to connect
- **public_key_path** (*str*) – path to ssh public key
- **private_key_path** (*str*) – path to ssh private key
- **security_group** (*str*) – firewall rule definition to apply on the instance
- **flavor** (*str*) – machine type to use for the instance
- **image_id** (*str*) – image type (os) to use for the instance
- **image_userdata** (*str*) – command to execute after startup
- **username** (*str*) – username for the given ssh key, default None

Returns str - instance id of the started instance

stop_instance(*instance_id*)
Stops the instance gracefully.

Parameters **instance_id** (*str*) – instance identifier

4.5.12 *elasticluster.providers.gce*

Cloud provider for the Google Compute Engine.

See <https://code.google.com/p/google-cloud-platform-samples/source/browse/python-client-library-example/gce.py?repo=compute> for reference.

elasticluster.providers.gce.**GCE_SCOPE = 'https://www.googleapis.com/auth/compute'**
the OAuth scope for the GCE web API

class elasticluster.providers.gce.**GoogleCloudProvider**(*gce_client_id*, *gce_client_secret*, *gce_project_id*, *zone='us-central1-a'*, *network='default'*, *email='default'*, *storage_path=None*)
Cloud provider for the Google Compute Engine.

Parameters

- **gce_client_id** (*str*) – Client ID to use in OAuth authentication.
- **gce_client_secret** (*str*) – Client secret (password) to use in OAuth authentication.
- **gce_project_id** (*str*) – Project name to log in to GCE.
- **zone** (*str*) – gce zone, default is *us-central1-a*
- **network** (*str*) – network to use, default is *default*
- **email** (*str*) – service email to use, default is *default*
- **storage_path** (*str*) – path to store authentication data (oauth.dat file). If no path is specified, the login data has to be entered after every request.

get_ips(*instance_id*)
Retrieves the ip addresses (private and public) from the cloud provider by the given instance id.

Parameters **instance_id** (*str*) – id of the instance

Returns list (ips)

Raises InstanceError if the ip could not be retrieved.

is_instance_running(*instance_id*)
Check whether the instance is up and running.

Parameters **instance_id** (*str*) – instance identifier

Reutrn True if instance is running, False otherwise

list_instances(*filter=None*)
List instances on GCE, optionally filtering the results.

Parameters **filter** (*str*) – Filter specification; see https://developers.google.com/compute/docs/reference/latest/instances/list for details.

Returns list of instances

start_instance(*key_name*, *public_key_path*, *private_key_path*, *security_group*, *flavor*, *image_id*, *image_userdata*, *username=None*, *instance_name=None*, ***kwargs*)
Starts a new instance with the given properties and returns the instance id.

Parameters

- **key_name** (*str*) – name of the ssh key to connect
- **public_key_path** (*str*) – path to ssh public key
- **private_key_path** (*str*) – path to ssh private key
- **security_group** (*str*) – firewall rule definition to apply on the instance

- **flavor** (*str*) – machine type to use for the instance
- **image_id** (*str*) – image type (os) to use for the instance
- **image_userdata** (*str*) – command to execute after startup
- **username** (*str*) – username for the given ssh key, default None
- **instance_name** (*str*) – name of the instance

Returns str - instance id of the started instance

stop_instance(*instance_id*)
Stops the instance gracefully.

Parameters **instance_id** (*str*) – instance identifier

Raises *InstanceError* if instance can not be stopped

4.5.13 *elasticluster.repository*

class elasticluster.repository.**AbstractClusterRepository**
Defines the contract for a cluster repository to store clusters in a persistent state.

delete(*cluster*)
Deletes the cluster from persistent state.

Parameters **cluster** (*elasticluster.cluster.Cluster*) – cluster to delete from persistent state

get(*name*)
Retrieves the cluster by the given name.

Parameters **name** (*str*) – name of the cluster (identifier)

Returns instance of *elasticluster.cluster.Cluster* that matches the given name

get_all()
Retrieves all stored clusters from the persistent state.

Returns list of *elasticluster.cluster.Cluster*

save_or_update(*cluster*)
Save or update the cluster in a persistent state. Elasticluster will call this method multiple times, so the implementation should handle save and update seamlessly

Parameters **cluster** (*elasticluster.cluster.Cluster*) – cluster object to store

class elasticluster.repository.**ClusterRepository**(*storage_path*)
This implementation of *AbstractClusterRepository* stores the cluster on the local disc using pickle. Therefore the cluster object and all its dependencies will be saved in a pickle (binary) file.

Parameters **storage_path** (*str*) – path to the folder to store the cluster information

delete(*cluster*)
Deletes the cluster from persistent state.

Parameters **cluster** (*elasticluster.cluster.Cluster*) – cluster to delete from persistent state

get(*name*)
Retrieves the cluster with the given name.

Parameters **name** (*str*) – name of the cluster (identifier)

Returns *elasticluster.cluster.Cluster*

get_all()
Retrieves all clusters from the persistent state.

Returns list of *elasticluster.cluster.Cluster*

save_or_update(*cluster*)
Save or update the cluster to persistent state.

Parameters **cluster** (*elasticluster.cluster.Cluster*) – cluster to save or update

class elasticluster.repository.**MemRepository**
This implementation of *AbstractClusterRepository* stores the clusters in memory, without actually saving the data on disk.

delete(*cluster*)
Deletes the cluster from memory.

Parameters **cluster** (*elasticluster.cluster.Cluster*) – cluster to delete

get(*name*)
Retrieves the cluster by the given name.

Parameters **name** (*str*) – name of the cluster (identifier)

Returns instance of *elasticluster.cluster.Cluster* that matches the given name

get_all()
Retrieves all stored clusters from the memory.

Returns list of *elasticluster.cluster.Cluster*

save_or_update(*cluster*)
Save or update the cluster in a memory.

Parameters **cluster** (*elasticluster.cluster.Cluster*) – cluster object to store

4.5.14 *elasticluster.subcommands*

class elasticluster.subcommands.**AbstractCommand**(*params*)
Defines the general contract every command has to fulfill in order to be recognized by the arguments list and executed afterwards.

execute()
This method is executed after a command was recognized and may vary in its behavior.

pre_run()
Overrides this method to execute any pre-run code, especially to check any command line options.

setup(*subparsers*)
This method handles the setup of the subcommand. In order to do so, every command has to add a parser to the subparsers reference given as parameter. The following example is the minimum implementation of such a setup procedure: parser = subparsers.add_parser("start") parser.set_defaults(func=self.execute)

class elasticluster.subcommands.**ListClusters**(*params*)
Print a list of all clusters that have been started.

class elasticluster.subcommands.**ListNodes**(*params*)
Show some information on all the nodes belonging to a given cluster.

execute()
Lists all nodes within the specified cluster with certain information like id and ip.

class elasticluster.subcommands.**ListTemplates**(*params*)
List the available templates defined in the configuration file.

class elasticluster.subcommands.**ResizeCluster**(*params*)
Resize the cluster by adding or removing compute nodes.

class elasticluster.subcommands.**SetupCluster**(*params*)
Setup the given cluster by calling the setup provider defined for this cluster.

class elasticluster.subcommands.**SftpFrontend**(*params*)
Open an SFTP session to the cluster frontend host.

class elasticluster.subcommands.**SshFrontend**(*params*)
Connect to the frontend of the cluster using *ssh*.

class elasticluster.subcommands.**Start**(*params*)
Create a new cluster using the given cluster template.

execute()
Starts a new cluster.

class elasticluster.subcommands.**Stop**(*params*)
Stop a cluster and terminate all associated virtual machines.

execute()
Stops the cluster if it's running.

setup(*subparsers*)
@see abstract_command contract

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

e

V