
ElastiCluster Manual

Release 1.3.dev

Services and Support for Science IT, University of Zurich

Apr 01, 2018

Contents

1	Introduction	1
1.1	Features	1
2	Overview	3
3	Table of Contents	5
3.1	Installation	5
3.2	Configuration	8
3.3	Usage	24
3.4	Troubleshooting	33
3.5	Playbooks distributed with ElastiCluster	44
3.6	Elasticcluster programming API	62
4	Indices and tables	67

ElastiCluster aims to provide a user-friendly command line tool to create, manage and setup computing clusters hosted on cloud infrastructures (like [Amazon's Elastic Compute Cloud EC2](#), [Google Compute Engine](#), or a private [OpenStack](#) cloud). Its main goal is to get a private cluster up and running with just a few commands; this [video](#) demos **ElastiCluster** setting up a computational batch-queueing cluster.

Complete documentation for **ElastiCluster** is available on the [Read The Docs](#) website. General discussion over **ElastiCluster**'s usage, features, and bugs takes place on the elasticcluster@googlegroups.com mailing-list (only subscribers can post).

The **ElastiCluster** project is an effort of the [Services and Support for Science IT \(S3IT\)](#) unit at the [University of Zurich](#), licensed under the [GNU General Public License version 3](#).

1.1 Features

ElastiCluster is in active development, and offers the following features at the moment:

- INI-style configuration file to define cluster templates
- Can start and manage multiple independent clusters at the same time * Automated setup of:
 - HPC clusters using [SLURM](#) or [GridEngine](#);
 - [Spark](#) / [Hadoop](#) clusters with [HDFS](#) and [Hive/SQL](#);
 - distributed storage clusters using [GlusterFS](#), [OrangeFS](#), or [Ceph](#);
 - ...or anything that you can install with an [Ansible](#) playbook!
- Growing and shrinking a running cluster.

ElastiCluster is currently in active development: please use the [GitHub](#) issue tracker to [file enhancement requests](#) and [ideas](#), or the [mailing-list](#) for discussion.

We appreciate pull requests for new features and enhancements. Please use the *master* branch as starting point.

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

Using the command line or a simple `API`, you can start a cluster (possibly overriding some of the default values, like the number of nodes you want to fire up) and configure it:

- `ElastiCluster` connects to the cloud provider indicated in the cluster configuration file, starts virtual machines, and waits until they are accessible via ssh.
- After all the VMs are up and running, `ElastiCluster` runs `Ansible` to configure the cluster.

Upon *resize* of the cluster¹, new virtual machines will be created and again `Ansible` will run on *all* the VMs, in order to properly add the new hosts to the cluster.

`ElastiCluster` commands `export` and `import` allow moving a running cluster's definition and status data from one machine to the other, to allow controlling the same cluster from different places.

¹ Currently, only growing a cluster is fully supported; shrinking a loaded cluster may remove nodes with running jobs and cause malfunctionings. See the `remove-node` command for a safer, albeit more low-level, way of shrinking clusters.

3.1 Installation

There are three ways to install ElastiCluster:

- Use the ElastiCluster Docker image: this is the quickest and easiest option if you just want to use ElastiCluster. See section *Quickstart* for instructions.
- Run ElastiCluster from the Python source code. This is the recommended option if you want to develop or customize ElastiCluster. See section *Installing development code from GitHub* for instructions.
- Install the `elasticcluster` Python package from PyPI. See section *Installing released code from PyPI* for instructions.

Warning: The code currently available on PyPI (ElastiCluster 1.2) is quite old and is lacking a number of important fixes and updates. Until ElastiCluster 1.3 is released, we suggest that you install from [GitHub](#) instead (see section *Installing development code from GitHub* below)

3.1.1 Quickstart

To install ElastiCluster over Docker: (1) download the `elasticcluster.sh` script into a file `elasticcluster.sh`, then (2) type this at your terminal prompt:

```
chmod +x elasticcluster.sh
```

That's it! You can now check that ElastiCluster is ready by running:

```
elasticcluster.sh --help
```

The first time it is run, the `elasticcluster.sh` script will check if Docker is installed, and ask for permission to install it if Docker is not found. Follow the on-screen instructions; see section *Getting Help* if you're in trouble.

You can also rename file `elasticcluster.sh` to `elasticcluster`, if you so like, to be consistent with the rest of the documentation.

Alternatively, you can also perform both steps at the terminal prompt:

```
# use this if the `wget` command is installed
wget -O elasticcluster.sh https://raw.githubusercontent.com/gc3-uzh-ch/elasticcluster/
↳master/elasticcluster.sh
chmod +x elasticcluster.sh

# use this if the `curl` command is installed instead
curl -O https://raw.githubusercontent.com/gc3-uzh-ch/elasticcluster/master/
↳elasticcluster.sh
chmod +x elasticcluster.sh
```

Choose either one of the two methods above, depending on whether `wget` or `curl` is installed on your system (Linux systems normally have `wget`; MacOSX normally uses `curl`).

3.1.2 Prepare the environment for installation

Note: this section is only relevant if you are installing ElastiCluster from source code (see *Installing development code from GitHub*) or from the PyPI package (see *Installing released code from PyPI*). None of these instructions are needed when running ElastiCluster from the Docker image.

The following sections document preliminary steps that need to be carried out in order to install ElastiCluster on a GNU/Linux or MacOSX computer.

We strongly recommend that `elasticcluster` is installed in a `python virtualenv`, in order to create a controlled environment where ElastiCluster can run without conflicting with system files or Python libraries. Installing in a `python virtualenv` makes it also easier to uninstall or upgrade `elasticcluster`.

Install required dependencies

CentOS/RHEL

To install software prerequisites for building and running ElastiCluster, run the following commands as the `root` admin user:

```
yum install gcc gcc-c++ git libffi-devel openssl-devel python-devel python-virtualenv
```

Debian/Ubuntu

To install software prerequisites for building and running ElastiCluster, run the following commands (omit the `sudo` word if running as the `root` admin user):

```
sudo apt-get install gcc g++ git libc6-dev libffi-dev libssl-dev python-dev virtualenv
```

MacOSX

Warning: Installation and testing of ElastiCluster on MacOSX is not currently part of the development or the release cycle. So these notes could be severely out of date. Please report issues and seek for solutions on the ElastiCluster [mailing-list](#).

In order to install ElastiCluster, you need to install [Xcode](#). It's free and you can install it directly from the [AppStore](#).

Create a Python “virtualenv”

Assuming you already have `virtualenv` installed on your machine (see section [Install required dependencies](#) if not), create a virtualenv and activate one with the following commands:

```
virtualenv elasticcluster
. elasticcluster/bin/activate
```

Now upgrade the `pip` command to the latest version (to ensure that it can correctly resolve the many dependencies of the ElastiCluster code):

```
pip install --upgrade 'pip>=9.0.0'
```

3.1.3 Installing released code from PyPI

Warning: The code currently available on PyPI (ElastiCluster 1.2) is quite old and is lacking a number of important fixes and updates. Until ElastiCluster 1.3 is released, we suggest that you install from [GitHub](#) instead (see section [Installing development code from GitHub](#) below)

Please follow the instructions in section [Install required dependencies](#) before proceeding.

Please follow the instructions in section [Prepare the environment for installation](#) before proceeding. The rest of this section assumes that you have created and activated a virtualenv in directory `elasticcluster`.

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

```
pip install elasticcluster
```

If you run into any problems, please have a look at the [Troubleshooting](#) section; the [mailing-list](#) is also a good place to get help.

3.1.4 Installing development code from GitHub

The source code of ElastiCluster is on [GitHub](#), if you want to test the latest development version you can clone the [GitHub elasticcluster repository](#).

Please follow the instructions in section [Prepare the environment for installation](#) before proceeding. The rest of this section assumes that you have created and activated a virtualenv in directory `elasticcluster`.

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

```
cd elasticcluster
git clone git://github.com/gc3-uzh-ch/elasticcluster.git src
cd src
pip install -e .
```

Now the `elasticcluster` command should be available in your current environment.

If you run into any problems, please have a look at the [Troubleshooting](#) section; the [mailing-list](#) is also a good place to get help.

Building documentation files

ElastiCluster documentation is available in the `docs/` directory, in reStructuredText-formatted plain text files.

You can additionally build HTML or PDF documentation; in the directory in the ElastiCluster virtualenv, type:

```
cd docs
make html
```

To build PDF documentation, use `make latexpdf` instead.

Note that building documentation files requires that the Python module [Sphinx](#) (click on the link for install instructions) is available in the same virtual environment where ElastiCluster is installed.

3.1.5 Getting help

In case you have trouble running the installation script, please send an email to elasticcluster@googlegroups.com or post a message on the web forum <https://groups.google.com/forum/#!forum/elasticcluster>. Include the full output of the script in your email, in order to help us to identify the problem.

3.2 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: `~/.elasticcluster/config` but you can specify a different location from the command line with the `-c` option.

If directory `~/.elasticcluster/config.d` exists (or, if you run `elasticcluster` with option `-c <PATH>`, the directory `<PATH>.d`), all files named `*.conf` contained in that directory are read and parsed. In this way, you can handle multiple clusters easily by distributing the configuration over multiple files, and disable only some of them by renaming the files.

After installing ElastiCluster for the first time, we suggest you run the following command:

```
elasticcluster list-templates
```

If no configuration file is found, it will copy an [example configuration file](#) in `~/.elasticcluster/config`. The example is fully commented and references back to appropriate sections in this document.

However, the example configuration file is not complete, as it does not contain any authentication information, so you will get an error log similar to the following:

```
WARNING Deploying default configuration file to /home/rmurri/.elasticcluster/config.
ERROR In section `cluster/gridengine-on-gce`: Key 'nodes' error: ...
ERROR Dropping configuration section `cluster/gridengine-on-gce` because of the above.
↪errors
```

```
ERROR In section `cloud/openstack`: Missing keys: <type 'str'>
ERROR Dropping configuration section `cloud/openstack` because of the above errors
ERROR Configuration section `cluster/slurm-on-ubuntu14` references non-existing cloud_
→section `openstack`. Dropping cluster definition.
0 cluster templates found in configuration file.
```

You will have to edit `~/.elasticcluster/config` and update it with the correct values.

Please refer to the following sections to understand the syntax of the configuration file and to know which options you need to set in order to use `elasticcluster`.

3.2.1 Basic syntax of the configuration file

ElastiCluster's configuration files are written similar to Microsoft Windows INI files. They will be read using Python's `ConfigParser` module, which see for more information on the supported syntax.

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

```
key=value
```

Section names have the form `[type/name]` where *type* is 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>/<class> override configuration for specific class of nodes within a cluster

storage usually not needed, allow to specify a custom path for the storage directory and the default storage type.

A valid configuration file must contain at least one section for each of the `cloud`, `login`, `cluster`, and `setup` sections.

3.2.2 Processing of configuration values

Within each `key=value` assignment, the *value* part undergoes the following transformations:

- References to enviromental variables of the form `$VARIABLE` or `${VARIABLE}` are replaced by the content of the named environmental variable, wherever they appear in a *value*.

For instance, the following configuration snippet would set the OpenStack user name equal to the login name on the computer where ElastiCluster is running:

```
[cloud/openstack]
username = ${LOGNAME}
# ...
```

- The following special strings are substituted, wherever they appear in a *value*:

this string expands to:
<code>\${elasticcluster_playbooks}</code>	Path to the root directory containing the Ansible playbooks distributed with ElastiCluster
<code>\${ansible_pb_dir}</code>	Deprecated alias for <code>\${elasticcluster_playbooks}</code>

- Within values that name a file or path name, a ~ character at the beginning of the path name is substituted with the path to the user's home directory. (In fact, this is a shorthand for \$HOME/)

3.2.3 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 a `cluster` section (see below).

Currently these cloud providers are available:

- `azure`: supports Microsoft Azure cloud
- `ec2_boto`: supports Amazon EC2 and compatible clouds
- `google`: supports Google Compute Engine
- `libcloud`: support many cloud providers through Apache LibCloud
- `openstack`: supports OpenStack-based clouds

Therefore the following configuration option needs to be set in the cloud section:

```
provider  
  
    the driver to use to connect to the cloud provider: azure, ec2_boto, openstack, google or  
    libcloud.
```

Note: The LibCloud provider can also provision VMs on Azure, EC2, Google Compute Engine, and OpenStack. The native drivers may however offer functionality that is not available through the generic LibCloud driver. Feedback is welcome on the ElastiCluster [mailing-list](#).

Valid configuration keys for azure

subscription_id UUID of the Azure subscription you want to use. For instructions on how to retrieve the subscription ID from the Azure web portal, see: <https://blogs.msdn.microsoft.com/mschray/2016/03/18/getting-your-azure-subscription-guid-new-portal/>

If not set, the value of the `AZURE_SUBSCRIPTION_ID` environmental variable will be used.

tenant_id UUID of the Azure tenant where the VMs and associated resources will be created. See <https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-create-service-principal-portal#get-application-id-and-authentication-key> for instructions on how to retrieve this value from the Azure web portal.

If not set, the value of the `AZURE_TENANT_ID` environmental variable will be used.

client_id UUID identifying an authorized Azure application (must have at least the *Contributor* role). See <https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-create-service-principal-portal#get-application-id-and-authentication-key> for instructions on how to retrieve this value from the Azure web portal.

If not set, the value of the `AZURE_CLIENT_ID` environmental variable will be used.

secret The 44-character long “key” corresponding to the authorized application identified by `client_id` above. See <https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-create-service-principal-portal#get-application-id-and-authentication-key> for instructions on how to generate this value from the Azure web portal.

If not set, the value of the `AZURE_CLIENT_SECRET` environmental variable will be used.

location Identifier of the Azure datacenter location (e.g., `WestUS`). Case insensitive.

See <https://azure.microsoft.com/en-us/global-infrastructure/regions/> for a map, or run `az account list-locations` (if you have the Azure CLI installed).

Obtaining Azure authentication credentials

In order to use ElastiCluster with Azure, you must create an application role and authorize it to create VMs and other resources; the subscription, tenant, and application ID, together with the application key (“secret”) shown during this process have to be saved into the configuration file (see above). A step-by-step walkthrough of the application authentication procedure can be found here: <https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-create-service-principal-portal#get-application-id-and-authentication-key>

Note that:

- When authorizing an application, you have to select a role (which in turn determines what exactly the application can or cannot do on Azure). In order to work properly, ElastiCluster needs at least the *Contributor* role (the example in the instructions above uses “Reader”, which will *not* suffice).
- The value for the *key* (the *secret* configuration item) will only be shown once during the procedure – if you fail to copy the secret string, you will have to repeat the procedure again from the start.

Valid configuration keys for `ec2_boto`

ec2_url URL of the EC2 endpoint. For Amazon EC2 it is probably something like:

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

(replace `us-east-1` with the zone you want to use).

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.

vpc name or ID of the AWS Virtual Private Cloud to provision resources in.

request_floating_ip request assignment of a public IPv4 address when the instance is started. Valid values are `yes` (or `True` or `1`) and `no` (or `False` or `0`; default). Please see <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-instance-addressing.html#concepts-public-addresses> regarding Amazon EC2’s assignment of public IPv4 addresses. Setting `request_floating_ip` to `yes` will force *elasticcluster* to request a public IPv4 address if the instance doesn’t get one automatically.

price If set to a non-zero value, ElastiCluster will allocate *spot instances* with a price less than or equal to the value given here. Note that there is currently no way to specify a currency: the amount is expressed in whatever *currency* is default in the Boto API (typically, US Dollars).

Defaults to 0, i.e., use regular non-spot instances.

This is typically best used in a *compute node* configuration section (see an example in the [example configuration file](#)); you probably do not want to run login, file server or similar central services on a spot instance (which can be terminated any time, depending on spot price bid).

timeout Maximum amount of seconds to wait for a spot instance to become available; if a request for a spot instance cannot be satisfied in the given time, the instance startup process aborts. If set to 0 (default), then wait indefinitely.

Note: Ignored if `price` is zero (default).

instance_profile Name of an IAM [instance profile](#) that contains roles allowing EC2 instances to have specified privileges. For example, you can allow EC2 instances to access S3 without passing credentials in.

Valid configuration keys for google

gce_client_id The API client ID generated in the Google Developers Console

gce_client_secret The API client secret generated in the Google Developers Console

gce_project_id The project ID of your Google Compute Engine project

network The GCE network to be used. Default is default.

zone The GCE zone to be used. Default is us-central1-a.

Obtaining your gce_client_id and gce_client_secret

Find the `gce_client_id` and `gce_client_secret` values by following instructions at: http://googlegenomics.readthedocs.io/en/latest/use_cases/setup_gridengine_cluster_on_compute_engine/index.html#index-obtaining-client-id-and-client-secrets

Valid configuration keys for libcloud

`driver_name`:

Name of the driver you want to configure (provider you want to connect with); it has to be one of the strings listed in column “Provider constant” in LibCloud’s [Provider Matrix](#) (which see for all supported providers).

Other configuration keys are provider-dependent; ElastiCluster configuration items map 1-1 to LibCloud “NodeDriver” instantiation parameters, both in name and in type.

For example, to configure a Digital Ocean connection, go to the page https://libcloud.readthedocs.io/en/latest/compute/drivers/digital_ocean.html and check what the *Instantiating a driver* section states: you would need to configure the key `access_token`.

A few examples for providers supported through LibCloud are given in the table below:

Provider	Additional arguments	Example
CloudSigma	username, password, region, api_version	username=user password=pass region=zrh api_version=2.0
Cloud-Stack	apikey, secretkey, host, path	apikey=key secretkey=secret host=example.com path=/path/to/api
ExoScale	key, secret, host, path	key=key secret=secret host=example.com path= path/to/api
LibVirt	uri	uri=qemu:///system
RackSpace	username, apikey, region	username=user apikey=key region=iad
vSphere	host, username, password	host=192.168.1.100 username=user password=pass

Valid configuration keys for openstack

auth_url URL of the OpenStack Identity service (aka *Keystone*, main entry point for OpenStack clouds), same as option `--os-auth-url` of the `*openstack*` command. If the environment variable `OS_AUTH_URL` is set, this option is ignored and the value of the environment variable is used instead.

identity_api_version Force use of the OpenStack Identity (“Keystone”) API v2 or v3. (Use the values 2 or 3 respectively.) If this configuration item is not specified, ElastiCluster will try v3 and then v2. If environment variable `OS_IDENTITY_API_VERSION` is set, this option is ignored and the value of the environment variable is used instead.

username OpenStack user name, same as option `--os-username` of the `*openstack*` command. If an environment variable `OS_USERNAME` is set, this option is ignored and the value of the environment variable is used instead.

user_domain_name OpenStack user domain. This is mandatory for Identity API v3. The default value is `default`. If the environment variable `OS_USER_DOMAIN_NAME` is set, this option is ignored and the value of the environment variable is used instead.

password OpenStack password, same as option `--os-password` of the `*openstack*` command. If an environment variable `OS_PASSWORD` is set, this option is ignored and the value of the environment variable is used instead.

project_name OpenStack project to use (formerly known as “tenant”), same as option `--os-project-name` of the `*openstack*` command. If an environment variable `OS_PROJECT_NAME` or `OS_TENANT_NAME` is set, this option is ignored and the value of the environment variable is used instead.

project_domain_name OpenStack project domain. This is mandatory for Identity API v3. The default value is `default`. If the environment variable `OS_PROJECT_DOMAIN_NAME` is set, this option is ignored and the value of the environment variable is used instead.

region_name OpenStack region. This is optional; not all OpenStack clouds require it and there is no widespread default: region names are arbitrary strings set by the OpenStack cloud administrators. Ask your local OpenStack support for valid values. If environment variable `OS_REGION_NAME` is set, this option is ignored and the value of the environment variable is used instead.

request_floating_ip request assignment of a “floating IP” when the instance is started. Valid values are `yes` (or `True` or `1`) and `no` (or `False` or `0`; default). Some cloud providers do 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 `yes` will force *elastcluster* to request such a floating IP if the instance doesn’t get one automatically.

Examples

For instance, to connect to Amazon's EC2 (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
vpc=vpc-one
```

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****
```

If you would want to use libcloud to connect to openstack using password authentication you can configure the following:

```
[cloud/libcloud]
provider=libcloud
driver_name=openstack
auth_url=**** YOUR AUTH URL ****
ex_tenant_name=**** YOUR TENANT NAME ****
ex_force_auth_version=2.0_password
username=**** YOUR USERNAME ****
password=**** YOUR PASSWORD ****
```

A larger set of commented examples can be found at: <https://github.com/gc3-uzh-ch/elasticsearch/tree/master/examples>

3.2.4 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.

Configuration keys

image_user Login name used to SSH into the virtual machine. In case you're using Google Compute Engine you have to set your user name here. So if your GMail address is `karl.marx@gmail.com`, use `karl.marx` as value of `image_user`.

image_sudo Boolean value: `yes` (or `True` or `1`; default) means that on the remote machine the `image_user` can execute commands as root by running the `sudo` program.

Warning: ElastiCluster makes the assumption that this value is always true and will not work correctly otherwise. This configuration item will be removed in a future version of ElastiCluster (as there is really no option).

image_user_sudo login name of the “super user”. This is optional, and defaults to *root*. There is little reason to ever change this value from the default.

user_key_name name of the *keypair* to use on the cloud provider. If the (pre-generated) keypair does not exist on the cloud platform, it will be added by ElastiCluster, uploading the public SSH key pointed to by *user_key_public* (see below).

Note: *This option is ignored on Azure, due to its different model for handling SSH authorization.*

user_key_private file containing a valid SSH private key to be used to connect to the virtual machine. Please note that this must match the *user_key_public* file (SSH keys always come in pairs).

Note: Currently ElastiCluster only supports RSA and DSA key types. Pull requests to add support for more modern SSH key types are very welcome.

Note: *This option is ignored on Azure, due to its different model for handling SSH authorization.*

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

Examples

For a typical Ubuntu VM, on either Amazon EC2 or most OpenStack providers, these values should be fine:

```
[login/ubuntu]
image_user=ubuntu
image_user_sudo=root
image_sudo=True
user_key_name=elasticcluster
# these paths should point to the SSH key file used to log in to VMs
user_key_private=~/.ssh/id_rsa
user_key_public=~/.ssh/id_rsa.pub
```

For Google Compute Engine, something like the following should be used instead:

```
[login/google]
image_user=****REPLACE WITH YOUR GOOGLE USERID (just the userid, not email)****
image_sudo=yes
user_key_name=elasticcluster
# You can generate the keypair with the command: `gcloud compute config-ssh`
user_key_private=~/.ssh/google_compute_engine
user_key_public=~/.ssh/google_compute_engine.pub
```

In contrast to other cloud providers, GCE creates a personal account on each VM so you effectively re-use the same *[login/google]* section across different VM images.

A larger set of commented examples can be found at: <https://github.com/gc3-uzh-ch/elasticsearch/tree/master/examples>

3.2.5 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, `elasticsearch` will run a `setup` provider in order to configure it.

A `setup` section is mostly independent of any other, and can be easily re-used across multiple clouds and base OS images – that’s the whole point of ElastiCluster!

General configuration keys

provider Type of the setup provider. So far, `ansible` is the only valid value (and, obviously, the default)

Controlling what is installed on the nodes

<class>_groups Comma separated list of Ansible groups nodes of kind *class* will belong to. For each `<class>_nodes` in a `[cluster/...]` section there should be a corresponding `<class>_groups` option to include that specific class of nodes in the given Ansible groups.

For example, to set up a standard HPC cluster you probably want to define only two main kinds of nodes: `frontend_groups` (for the master/control server) and `compute_groups` (for the compute nodes). A common setup for a SLURM cluster is:

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

This will configure the `frontend001` node as SLURM master and Ganglia collector and frontend, and the `computeXXX` nodes as SLURM executors and Ganglia `gmond` sources.

Ansible group names supported by ElastiCluster can be found in the **Playbooks** section of this manual. You can combine more groups together, separating the names with a comma (,) – but of course not all combinations make sense.

Warning: Any group name that is not supported by ElastiCluster playbooks will (silently) be ignored, so watch out for typos!

<class>_var_<varname> Define an variable called `<varname>` that applies only to the given node `<class>`. See the **Playbooks** section to know which variables can be set and their meaning.

global_var_<varname> Define a variable called `<varname>` that applies to all the nodes in the cluster. See the **Playbooks** section to know which variables can be set and their meaning.

playbook_path Path to the Ansible playbook file to use when running `elasticsearch setup`. The default value is to use `playbook site.yml` in the root directory of the distributed with ElastiCluster.

Controlling Ansible invocation

ansible_command Path name of the `ansible-playbook` command; defaults to `ansible-playbook`, i.e., search for the command named `ansible-playbook` in the shell search path. Can also include arguments that will be *prepending* to other arguments that ElastiCluster adds to build the “setup” command invocation.

ansible_extra_args Arguments to *append* to the “setup” command invocation; can be used to override specific parameters or to further influence the behavior of the `ansible-playbook` command (e.g., skip certain tags).

The string is split according to POSIX shell parsing rules, so quotes can be used to protect arguments with embedded spaces.

Examples:

```
[setup/ansible]
# do not run any setup action tagged as 'users'
ansible_extra_args = --skip-tags users

[setup/ansible]
# ask for confirmation at each step
ansible_extra_args = --step
```

ansible_ssh_pipelining Enable or disable SSH pipelining when setting up the cluster. Enabled by default, as it improves connection speed. Incompatible with some base OS'es, notoriously CentOS6. Setting this to `no/false/0` disables it.

ansible_<option> Any configuration key starting with the string `ansible_` is used to set the corresponding (uppercased) environmental variable and thus override Ansible configuration.

For example, the following settings raise the number of concurrent Ansible connections to 20 and allow a maximum waiting time of 300 seconds for a single task to finish:

```
[setup/ansible]
# ...
ansible_forks=20
ansible_timeout=300
```

The full list of environment variables used by Ansible is available from the [Ansible configuration](#) section of the Ansible online documentation. Invoking `elasticluster setup` with highest verbosity (e.g., `-vvv`) will dump the entire environment that Ansible is being called with to the DEBUG-level log.

Note: Any `ANSIBLE_*` variables defined in the environment take precedence over what is defined in the `[setup/*]` section. Care must be taken when overriding some variables, particularly `ANSIBLE_ROLES_PATH`, which contain paths and references to parts of ElastiCluster: if those paths are missing from the replaced value, a number of fatal errors can happen.

ssh_pipelining **Deprecated.** Use `ansible_ssh_pipelining` instead.

Examples

A `setup` section is mostly independent of any other, and can be easily re-used across multiple clouds and base OS images – that’s the whole point of ElastiCluster!

The following shows how to set up a simple SoGE cluster using the **Playbooks_** distributed with ElastiCluster:

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

This example shows how to combine multiple Ansible groups into a class of nodes; namely, install [Ganglia](#) alongside with PBS/TORQUE:

```
[setup/pbs]
provider=ansible
frontend_groups=pbs_master, ganglia_master
compute_groups=pbs_worker, ganglia_monitor
```

This final example shows how variables can be used to customize or set options in the playbooks. Specifically, the example shows how to install NIS/YP to easily manage users across the cluster:

```
[setup/slurm]
# provider=ansible is the default
frontend_groups=slurm_master
compute_groups=slurm_worker

# install NIS/YP to manage cluster users
global_var_multiuser_cluster=yes
```

A larger set of commented examples can be found at: <https://github.com/gc3-uzh-ch/elasticsearch/tree/master/examples>

3.2.6 Cluster Section

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

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

A 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.

Some configuration keys can be overridden for specific node kinds. The way to do this is to create a section named like this:

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

Any configuration specified in this section would take precedence over the values given in section `[cluster/<name>]`, but only for nodes of class `<kind>`.

For example: assume you have a standard SLURM cluster with a frontend which is used as master node and NFS server for the home directories, and a set of compute nodes. You may want to use different VM 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. If your cloud provided, e.g., a `bigdisk` flavor for VMs with a large root disk space, and a `hpc` flavor for VMs optimized for running computational jobs, you could use the former for the frontend node and the latter for the compute nodes. Your configuration will thus look like:

```
[cluster/slurm]
# ...
flavor=hpc
frontend_nodes=1
```

```

compute_nodes=10

[cluster/slurm/frontend]
flavor=bigdisk

[cluster/slurm/compute]
# the following setting is (implicitly) inherited
# from the `[cluster/slurm]` section
#flavor=hpc

```

Cluster-wide configuration keys

The following configuration keys can only be specified in a top-level `[cluster/...]` section (i.e., *not* in node-level `[cluster/.../node]` override).

cloud Name of a valid `cloud` section.

login Name of a valid `login` section. For instance `ubuntu` or `google-login`.

setup Name of a valid `setup` section.

<class>_nodes the number of nodes of type `<class>`. These configuration options will define the composition of your cluster. Each `<class>_nodes` group is configured using the corresponding `<class>_groups` configuration option in the `[setup/...]` section.

<class>_min_nodes (optional) Deprecated. Please rename to `<class>_nodes_min`.

<class>_nodes_min (optional) Minimum amount of nodes of type `<class>` that must be up & running in order to start configuring the cluster.

When running `elasticcluster start` to start a cluster, creation of some instances may fail; if at least this amount of nodes are started correctly (i.e. are not in error state), the cluster is configured anyway. Otherwise, the `start` command will fail.

ssh_to (optional; see defaults below) Which class of nodes to SSH into, when running `elasticcluster ssh` or `elasticcluster sftp`.

Commands `elasticcluster ssh` and `elasticcluster sftp` need to single out one node from the cluster, and connect to it via SSH/SFTP. This parameter can specify:

- either a node name (e.g., `master001`) which will be the target of SSH/SFTP connections, or
- a node class name (e.g., `frontend`): the first node in that class will be the target.

If `ssh_to` is not specified, ElastiCluster will try the class names `ssh`, `login`, `frontend`, and `master` (in this order). If the cluster has no node in all these classes, then the first found node is used.

ssh_probe_timeout (optional; default: 5) Maximum time (in seconds) to wait for the initial SSH connection to a node to be established.

This timeout is used during `elasticcluster start`: each of the nodes' IP addresses will be probed with an SSH connection until one responds; each attempt will time out after this number of seconds. If no attempt succeeds within `start_timeout` seconds (see below), then the node is marked as “down” and skipped in the `elasticcluster setup` phase.

You may want to increase this parameter only in case the TCP round-trip-time to the cluster is terribly slow.

start_timeout (optional; default: 300) Only used when running `elasticcluster start`: maximum time (in seconds) to wait for nodes to be up and running. A node is considered “up and running” if ElastiCluster can open an SSH connection to it.

Nodes that are not up and running after this interval has elapsed, will be ignored in the following `elasticcluster` setup phase. If not enough nodes of any class are available (see `<class>_nodes_min`), then `elasticcluster` start aborts with an error.

Sensible values for this parameter vary much depending on the cloud provider and the size of the cluster. The default value is 600 seconds (10 minutes), which is normally enough for clusters up to a few tens of nodes running on public commercial cloud providers, but may need to be increased for larger clusters.

thread_pool_max_size (optional) Maximum number of Python worker threads to create for starting VMs in parallel. Default is 10.

Overridable configuration keys

The following configuration keys can appear in a top-level `[cluster/...]` section, as well as in a node-level `[cluster/.../node]` override. Configuration keys specified in a node-level section take precedence over cluster-wide ones.

flavor The VM “size” to use. Different cloud providers call it differently: could be “instance type”, “instance size” or “flavor”. This setting can be overwritten in the Cluster Node section, e.g. to use fewer resources on the frontend nodes than on the compute nodes.

image_id Disk image ID to use as a base for all VMs in this cluster (unless later overridden for a class of nodes, see below). Actual format is cloud specific:

- Azure uses the form *publisher/offer/sku/version* (e.g., *canonical/ubuntuuserver/16.04.0-LTS/latest*) You can see commands to list available values for each of these parts at: <https://docs.microsoft.com/en-us/cli/azure/vm/image?view=azure-cli-latest>
- Amazon EC2 uses IDs like *ami-123456*.
- For Google Compute Engine you can also use a URL of a private image; run `gcloud compute images describe <your_image_name>:file:` to show the selfLink URL to use.
- OpenStack uses UUIDs (e.g. *2bf3baba-35c8-4e20-9cc9-b36808720c9b*); use command `openstack image list` or the web dashboard to list available images.

image_userdata (optional) Shell script to be executed (as root) when the machine starts. This can happen before ElastiCluster even gets a chance to connect to the VM.

Note: *This option is (currently) ignored on Azure.*

network_ids (optional) Comma separated list of network or subnet IDs the nodes of the cluster will be connected to. Only supported when the cloud provider is `ec2_boto` or `openstack`.

security_group (optional; default: default) Name of security group to use when starting the instance.

Note: *This option is ignored on Azure.*

All VMs started by ElastiCluster on MS-Azure will be put in a security group named after the cluster, which initially only allows inbound connections to the SSH port. Any other port must be added by the user through the portal or any other Azure management interface.

Note: On Amazon EC2, the “default” security group only allows network communication among hosts in the group and does *not* allow SSH connections from the outside. This will make ElastiCluster fail as it cannot connect to the cluster nodes (see, e.g., [issue #490](#)). You will need to add rules to the “default” security group (or create a new one and use that) such that: (1) SSH connections from the network where you run ElastiCluster are

allowed, and (2) all TCP and UDP connections among cluster nodes are allowed – the “default” security group only allows TCP, not UDP.

Additional optional configuration keys for Amazon EC2

Options `price` and `timeout` (see their documentation in the “ec2_boto” cloud provider section) can be specified here as well, to place nodes on spot instances.

Additional optional configuration keys for Google Cloud

accelerator_count If set to an integer number > 0, then request instances equipped with this number of accelerators (typically, GPUs) of the kind specified by `accelerator_type`.

Default is 0, i.e., do not request GPU accelerators.

accelerator_type Type of accelerator to request. Can be one of the following options:

- Full URL specifying an accelerator type valid for the zone and project VMs are being created in. For example, `https://www.googleapis.com/compute/v1/projects/[PROJECT_ID]/zones/[ZONE]/acceleratorTypes/[ACCELERATOR_TYPE]`
- An accelerator type name (any string which is not a valid URL). This is internally prefixed with the string `https://www.googleapis.com/compute/v1/projects/[PROJECT_ID]/zones/[ZONE]/acceleratorTypes/` to form a full URL.

Only used if `accelerator_count` is > 0.

allow_project_ssh_keys

When `yes` (default), SSH login is allowed to a node using any of the [project-wide SSH keys](#) (if any are defined). When `no`, only the SSH key specified by ElastiCluster config’s `[login/*]` section referenced by this cluster will be allowed to log in (instance-level key).

Note that Google Cloud API uses the *negative* setting for this option, i.e., the API allows you to *block* project-wide SSH keys – but the default outcome is unchanged.

boot_disk_type Define the type of boot disk to use. Supported values are `pd-standard` (default) and `pd-ssd`.

boot_disk_size Define the size of boot disk to use; values are specified in gigabytes. Default value is 10.

min_cpu_platform Require that VMs run on CPUs with this platform (see <https://cloud.google.com/compute/docs/instances/specify-min-cpu-platform#availablezones> for a list) or better. Setting a minimum CPU platform may be necessary to get access to instance types with special features (e.g., high number of cores)

tags Comma-separated list of instance tags.

scheduling Define the type of instance scheduling. Only supported value is `preemptible`.

Additional optional configuration keys for OpenStack clouds

boot_disk_type Define the type of boot disk to use. Supported values are types available in the OpenStack volume (“cinder”) configuration.

When using this option for OpenStack, it creates volumes to be used as the root disks for the VM’s of the specified size, when terminating and instance the volume will be deleted automatically. Always specify the `boot_disk_size` when using this with OpenStack.

boot_disk_size Define the size of boot disk to use. Values are specified in gigabytes. There is no default; this option is mandatory if `boot_disk_type` is also specified.

Examples

This basic example shows how to set up a SoGE cluster on Google Cloud. (The example assumes that sections `[setup/gridengine]`, `[cloud/google]` and `[login/google]` have been defined elsewhere in the configuration file.)

```
[cluster/gridengine-on-gce]
setup=gridengine
frontend_nodes=1
compute_nodes=2

# this is cloud specific
cloud=google
security_group=default
flavor=n1-standard-1

image_id=****REPLACE WITH OUTPUT FROM: gcloud compute images list | grep debian | cut_
↪-f 1 -d " "****

# on GCE, all images can use the same `login` section
login=google
```

The following slightly more complex example shows how to set up a TORQUE cluster on an OpenStack cloud, using different VM flavors for the front-end node (less CPU and larger disk) and compute nodes (more CPU and memory).

The rationale behind this configuration is as follows: for the front-end node more space is needed (since it's the NFS server for the whole cluster) and you don't need many cores or much memory, while the compute nodes may require more memory and more cores but are not eager about disk space. If your cloud provided, e.g., a "big disk" flavor for VMs with a large root disk space, and a "hpc" flavor for VMs optimized for running computational jobs, you could use the former for the frontend node and the latter for the compute nodes. Your configuration will thus look like the following:

```
[cluster/torque]
setup=pbs
frontend_nodes=1
compute_nodes=8

# this is cloud-specific info (using OpenStack for the example)
cloud=openstack
network_ids=eaf06405-6dc2-43d1-9d5a-18bb266e36a8
security_group=default

# CentOS 7.4
image_id=bab386b3-2c21-4a67-a146-a658668ac096

# `login` info is -in theory- image-specific
login=centos

[cluster/torque/frontend]
# front-end has less CPU and RAM but more disk
flavor=2cpu-4ram-largedisk

[cluster/torque/compute]
```

```
# compute nodes have much CPU power and RAM
flavor=8cpu-64ram-hpc
```

The following example shows how to set up a SLURM cluster on AWS which uses EC2 “spot instances” for the compute nodes, by specifying bidding price and maximum wait timeout. The spot instance configuration applies only to the cluster nodes of class `compute` – the front-end node runs on a regular instance so it is not terminated abruptly (which would lead to total job and data loss). Since compute nodes are started on “spot instances”, which may be not available at the bid price within the given timeout, you also want to set a *minimum* number of compute nodes (configuration item `compute_nodes_min`) that must be available in order to proceed to cluster setup:

```
[cluster/slurm-on-aws]
setup=slurm
frontend_nodes=1
compute_nodes=8
compute_nodes_min=2

# this is cloud-specific info
cloud=amazon-us-east-1
image_id=ami-90a21cf9
security_group=default
flavor=m3.large

# login info is image-specific
login=ubuntu

[cluster/slurm-on-aws/compute]
# use spot instances for compute
price=0.08
timeout=600
```

A larger set of commented examples can be found at: <https://github.com/gc3-uzh-ch/elasticsearch/tree/master/examples>

3.2.7 Storage section

This section is used to customize the way ElastiCluster saves the state of your clusters on disk.

By default, all persisted data is saved in `~/.elasticsearch/storage`. This includes two main files for each cluster:

- `<cluster>.yaml`: a file containing information about your cluster
- `<cluster>.known_hosts`: a file containing the ssh host keys of the nodes of your cluster.

These files are very important: if they are broken or missing, ElastiCluster will **not** be able to recover any information about the cluster.

In addition to these two files, the setup provider and the cloud provider might create other files in the storage directory, but these are not critical, as they are re-generated if needed.

To change the default path to the storage directory you can create a new `storage` section and set the `storage_path` value:

```
[storage]
storage_path = $HOME/src/elasticsearch/
```

By default the status of the cluster is saved in **YAML** format, but also **Pickle** and **Json** formats are available. To save the cluster in a different format, use option `storage_type`:

```
[storage]
storage_path = $HOME/src/elasticcluster/
storage_type = json
```

Please note that only newly-created files will honour the `storage_type` option! Existing files will keep their format.

3.3 Usage

The syntax of the `elasticcluster` command is:

```
elasticcluster [-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 `elasticcluster` creates new VMs in parallel, to speedup the process, but if you run it with at least *four* `-v` options, `elasticcluster` 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 `~/.elasticcluster/storage`

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

-c PATH, --config PATH

Path to the configuration file. By default this is `~/.elasticcluster/config`. If a directory named `<PATH>.d` (or, by default, `~/.elasticcluster/config.d`) exists, all files contained in that directory and ending in `.conf` are read too.

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

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

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.

export Export a cluster as a ZIP file.

import Import a cluster from a ZIP file created with `elasticcluster export`.

An help message explaining the available options and subcommand of `elasticcluster` is available by running:

```
elasticcluster -h
```

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

```
elasticcluster <cmd> -h
```

3.3.1 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: elasticcluster 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:

```
elasticcluster 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 `elasticcluster` with option:

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

--no-setup By default `elasticcluster` 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, `elasticcluster` will:

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

    elasticsearch ssh slurm

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

    elasticsearch 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.

3.3.2 The stop command

The **stop** command terminates all the running VM instances and deletes all information related to the cluster saved on the local disk.

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

Basic usage of the command is:

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

Like for 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 elasticsearch), this command will ignore these errors and will force termination of all the other instances.

`--yes`

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

3.3.3 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:

```
$ elasticcluster 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
```

3.3.4 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: elasticcluster 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 `elasticcluster list-nodes` will not contact the EC2 provider to get up-to-date information, unless `-u` option is given.

Example:

```
$ elasticcluster 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:

```
elasticcluster ssh centossge
```

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

```
elasticcluster 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
```

3.3.5 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: elasticcluster 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:

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

name:      aws-slurm
cloud:     aws
compute nodes: 2
frontend nodes: 1
```

```

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

name:      slurm_x1
cloud:     hobbles
compute nodes: 2
frontend nodes: 1

name:      slurm13.04
cloud:     hobbles
compute nodes: 2
frontend nodes: 1

```

3.3.6 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: elasticsearch setup [-h] [-v] cluster [-- extra ...]
```

First argument `cluster` is the name of a cluster; it must have been *started* previously.

Following arguments (if any) are appended verbatim to the *ansible-playbook* command-line invocation that is used to actually carry out the configuration task. This allows overriding some defaults set by ElastiCluster or, more interestingly, add other playbook files or variables or change the behavior of *ansible-playbook* altogether, as the following examples show:

- Only execute setup actions marked with a specific tag:

```
elasticsearch setup mycluster -- --tags hdfs
```

- Read and define additional variables from file `:file:vars.yml`:

```
elasticsearch setup mycluster -- -e @vars.yml
```

- Execute an additional playbook after ElastiCluster's main one:

```
elasticsearch setup mycluster -- /path/to/play.yml
```

- Do not run setup at all, just show what nodes are to run what play:

```
elasticsearch setup mycluster -- --list-hosts
```

The following options are additionally available:

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

-v, --verbose Adding one or more `-v` will increase the verbosity accordingly. The verbosity setting is propagated to the *ansible-playbook* command.

3.3.7 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, `elasticcluster` 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 `elasticcluster` 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 `elasticcluster` package.

Basic usage of the command is:

```
usage: elasticcluster 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:

```
elasticcluster 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: `elasticcluster` 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 `elasticcluster` 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 disruptive action and is not supported by all the distributed playbooks, `elasticcluster` will always ask for confirmation before doing any modification, unless this option is given.

3.3.8 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 `elasticcluster list-nodes <cluster>` will show which host will be used as frontend node.

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

```
elasticcluster 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:

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

Note that `elasticcluster` will save in `~/.elasticcluster/storage/<clustername>.known_hosts` the ssh host keys of the VM instances after the first connection, and re-use them to protect you from a Man-In-The-Middle attack. Therefore, the following options are passed to `ssh` command line:

- `-o UserKnownHostsFile=~/.elasticcluster/storage/<clustername>.known_hosts` Use the generated known hosts file to protect against MIIT attacks.
- `-o StrictHostKeyChecking=yes` Enable check of the host key of the remote machine.

3.3.9 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 `elasticcluster list-nodes <cluster>` will show which host will be used as frontend node.

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

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

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

Note that elasticcluster will save in `~/.elasticcluster/storage/<clustername>.known_hosts` the ssh host keys of the VM instances after the first connection, and re-use them to protect you from a Man-In-The-Middle attack. Therefore, the following options are passed to `sftp` command line:

- `-o UserKnownHostsFile=~/.elasticcluster/storage/<clustername>.known_hosts` Use the generated known hosts file to protect against MIIT attacks.
- `-o StrictHostKeyChecking=yes` Enable check of the host key of the remote machine.

3.3.10 The export command

The `export` command is useful when you need to copy a cluster you already created on a different computer.

The usage of `export` command is as follow:

```
elasticcluster export [-h] [--overwrite] [--save-keys] [-o FILE] cluster
```

The following options are available:

`--overwrite`

Overwrite ZIP file if it exists.

`--save-keys`

Also store public and *private* ssh keys. **WARNING:** this will copy sensible data. Use with caution!

`-o FILE, --output-file FILE`

Output file to be used. By default the cluster is exported into a `<cluster>.zip` file where `<cluster>` is the cluster name.

When exporting a cluster, a zip file will be created, containing all the necessary information for elasticcluster. By default elasticcluster will not export also the ssh keys; if you want to export them as well, run with option `--save-keys`

One word of advice though: ssh keys are *sensible data*: they allow to connect to a remote host without knowing the remote password. Owning a private ssh key means having access to any machine where the corresponding public key was deployed.

If you use a different set of keys for each cluster, and you don't use the same key also for other hosts, you are safe using `--save-keys`. Otherwise, be sure to share the ZIP file *only* with people that are supposed to have access to **all** hosts that give access to those ssh keys.

3.3.11 The import command

The `import` command is used to import a cluster exported with the `export` command. This will copy the relevant data in the storage directory and if needed will also update the cluster (for instance, the path to the `known_hosts` file, or the cluster name, if changed)

Basic usage of the command is:

```
elasticcluster import [-h] [-v] [--rename NAME] file
```

where `file` is the zip file produced by `elasticcluster export`.

The following option is available:

`--rename NAME`

Rename the cluster during import.

ElastiCluster will refuse to import a cluster if in the current storage directory is already present a cluster with the same name. You can however import it using a different name, by passing an argument to the `--rename` option.

The `import` command can also import any ssh key included in the ZIP file, if the export was performed with `--save-keys`. In this case, elasticsearch will also update the corresponding attributes of both cluster and nodes.

Please note that if the cluster was *not* exported with `--save-keys`, elasticsearch cannot know where the correct ssh key files are, therefore you will probably need to *manually* update these values in the storage file.

3.4 Troubleshooting

This section lists known problems together with solutions and workarounds (if available). Please use the [mailing-list](#) for further help and for any problem not reported here!

Contents

- *Troubleshooting*
 - *Setup of a cluster fails and stops at task `nfs-client: add to /etc/fstab`*
 - *Downloading fails with error “Failed to validate the SSL certificate”*
 - *Running any `elasticsearch` command fails with a version conflict about the `requests` package*
 - *Running any `elasticsearch` command fails with a version conflict about the `pbr` package*
 - *Installation fails with `ValueError: ('Expected version spec in' [...])`*
 - *Installation fails complaining about version of `setuptools`*
 - *Installation fails with `VersionConflict: ... Requirement.parse('setuptools>=17.1')`*
 - *Upgrading `setuptools` fails with `ImportError: No module named extern`*
 - *Installation fails with: “fatal error: `ffi.h`: No such file or directory”*
 - *Installation fails with: “fatal error: `openssl/openssl.h`: No such file or directory”*
 - *Installation fails with: “fatal error: `Python.h`: No such file or directory”*
 - *Installation fails with: “unable to execute `gcc`: No such file or directory”*
 - *Installation fails with `Too many levels of symbolic links`*
 - *Setup of RHEL/CentOS 7 clusters fails immediately*
 - *Setup of Ubuntu 16.04 (“xenial”) clusters fails immediately*
 - *Issues when installing from source on MacOSX*
 - *Error “`ImportError: No module named anyjson`” on MacOSX*

3.4.1 Setup of a cluster fails and stops at task `nfs-client: add to /etc/fstab`

You get this error when starting a new cluster: virtual machines are started correctly and cluster configuration begins, however at some point the progress stalls and then after a few minutes’ time out, the Ansible playbook stops running. This error will be the last task mentioned before the “PLAY RECAP”:

```
TASK [nfs-client : add to /etc/fstab]
↳*****
fatal: [compute001]: FAILED! => {"changed": false, "failed": true, "msg": "Error
↳mounting /home: mount.nfs: Connection timed out\n"}
```

This is due to the security group (also named “direwall rules”) not allowing all traffic across cluster VMs: for NFS to work correctly, each VM in the cluster must be able to open connections to other VMs on any TCP *and* UDP port. (TCP and UDP are typically named “protocols” in security groups / firewall rules context.)

- For AWS EC2, you apply the instructions given in page “Security Groups for Your VPC”, section “Adding, Removing, and Updating Rules” at step 6. – be sure to apply them to the security group you’re using with ElastiCluster;
- For Google Cloud, you need to add the rule `default-allow-internal` – see <https://cloud.google.com/vpc/docs/firewalls> for details.
- For OpenStack, you can find instructions on how to manipulate security groups at <https://help.dreamhost.com/hc/en-us/articles/360000717692-Managing-Security-Groups-using-the-OpenStack-CLI>. Note that you can also manipulate security groups through the Horizon web interface but it’s hard to find web documents on its usage since every commercial provider seems to have their own different reimplementations of the OpenStack web frontend.

3.4.2 Downloading fails with error “Failed to validate the SSL certificate”

This error may happen at different stages and with different web sites in the setup phase but the commonality is that the Ansible error message starts with the words *Failed to validate the SSL certificate*. For example:

```
TASK [lmod : Download sources]
↳*****
fatal: [frontend001]: FAILED! => {"changed": false, "failed": true, "msg": "Failed to
↳validate the SSL certificate for github.com:443. Make sure your managed systems
↳have a valid CA certificate installed. You can use validate_certs=False if you do
↳not need to confirm the servers identity but this is unsafe and not recommended.
↳Paths checked for this platform: /etc/ssl/certs, /etc/pki/ca-trust/extracted/pem, /
↳etc/pki/tls/certs, /usr/share/ca-certificates/cacert.org, /etc/ansible. The
↳exception msg was: (\\"bad handshake: Error([('SSL routines', 'ssl3_read_bytes',
↳'tlsv1 alert protocol version')]),)\",)."}"
```

This error is typically caused by the “trusted CA repository” on the host being set up being out of sync with the site (GitHub in the above example). The site might have gotten a new SSL/TLS certificate more recently than the base OS updated its “trusted CA” store, so Ansible cannot verify that the connection is valid and you’re not downloading software from a rogue site.

The “correct fix” (from a security point of view) would be to ensure that the base OS from your cluster has an up-to-date “trusted CA repository” that can validate popular sites like `github.com`. However, providing instructions for how to do this is very much dependent on the base OS and certainly outside the scope of these short notes.

The *quick workaround* instead is to allow ElastiCluster to skip the verification process and download software insecurely. This can be accomplished by adding this line to your cluster’s `[setup/...]` configuration section:

```
global_var_insecure_https_downloads=yes
```

3.4.3 Running any `elasticluster` command fails with a version conflict about the `requests` package

You can get this error when ElastiCluster installed fine, but attempting to run *any* command fails with a Python traceback like the following one:

```
Traceback (most recent call last):
  File "/home/ec2-user/elasticluster/bin/elasticluster", line 6, in <module>
    from pkg_resources import load_entry_point
  File "/home/ec2-user/elasticluster/lib/python2.7/site-packages/pkg_resources/__init__
↪_.py", line 3036, in <module>
    @_call_aside
  File "/home/ec2-user/elasticluster/lib/python2.7/site-packages/pkg_resources/__init__
↪_.py", line 3020, in _call_aside
    f(*args, **kwargs)
  File "/home/ec2-user/elasticluster/lib/python2.7/site-packages/pkg_resources/__init__
↪_.py", line 3049, in _initialize_master_working_set
    working_set = WorkingSet._build_master()
  File "/home/ec2-user/elasticluster/lib/python2.7/site-packages/pkg_resources/__init__
↪_.py", line 656, in _build_master
    return cls._build_from_requirements(__requires__)
  File "/home/ec2-user/elasticluster/lib/python2.7/site-packages/pkg_resources/__init__
↪_.py", line 669, in _build_from_requirements
    dists = ws.resolve(reqs, Environment())
  File "/home/ec2-user/elasticluster/lib/python2.7/site-packages/pkg_resources/__init__
↪_.py", line 859, in resolve
    raise VersionConflict(dist, req).with_context(dependent_req)
pkg_resources.ContextualVersionConflict: (requests 2.13.0 (/home/ec2-user/
↪elasticluster/lib/python2.7/site-packages), Requirement.parse('requests!=2.12.2,!=2.
↪13.0,>=2.10.0'), set(['keystoneauth1']))
```

There is a workaround for this bug in ElastiCluster from [commit 7bf55b8](#) onwards, (to be) included in ElastiCluster 1.3. So, upgrading to the latest ElastiCluster code should solve the issue. Alternatively, you can solve the problem by manually resolving the conflict:

```
pip install requests==2.12.3
```

Unfortunately, the root cause of this problem does not lie in ElastiCluster; instead it stems from the dependency resolution mechanism of the package installer *pip*. See [ElastiCluster issue #414](#) for more technical details.

3.4.4 Running any `elasticluster` command fails with a version conflict about the `pbr` package

You can get this error when ElastiCluster installed fine, but attempting to run *any* command fails with a Python traceback that ends with a line like the following one:

```
pkg_resources.ContextualVersionConflict: (pbr 1.10.0 (...), Requirement.parse('pbr>=2.
↪0.0'), set(['oslo.i18n', 'oslo.serialization', 'oslo.utils', 'debtcollector']))
```

This means that you have a mixture of older and newer OpenStack libraries in your ElastiCluster installation: to solve the issue, make a new ElastiCluster virtual environment and install again from scratch.

The root cause of the issue lies in the interplay between the way *pip* handles dependencies of dependent packages. There is unfortunately little ElastiCluster can do about it.

3.4.5 Installation fails with ValueError: ('Expected version spec in' [...])

When trying to install ElastiCluster with `pip install`, you get a long error report that ends with this Python traceback:

```
Traceback (most recent call last):
  File "/opt/python/2.7.9/lib/python2.7/site-packages/pip/basecommand.py", line 232, in
  ↪ in main
    status = self.run(options, args)
  File "/opt/python/2.7.9/lib/python2.7/site-packages/pip/commands/install.py", line
  ↪ 339, in run
    requirement_set.prepare_files(finder)
  File "/opt/python/2.7.9/lib/python2.7/site-packages/pip/req/req_set.py", line 436, in
  ↪ in prepare_files
    req_to_install.extras):
  File "/opt/python/2.7.9/lib/python2.7/site-packages/pip/_vendor/pkg_resources/__
  ↪ init__.py", line 2496, in requires
    dm = self._dep_map
  File "/opt/python/2.7.9/lib/python2.7/site-packages/pip/_vendor/pkg_resources/__
  ↪ init__.py", line 2491, in _dep_map
    dm.setdefault(extra, []).extend(parse_requirements(reqs))
  File "/opt/python/2.7.9/lib/python2.7/site-packages/pip/_vendor/pkg_resources/__
  ↪ init__.py", line 2820, in parse_requirements
    "version spec")
  File "/opt/python/2.7.9/lib/python2.7/site-packages/pip/_vendor/pkg_resources/__
  ↪ init__.py", line 2785, in scan_list
    raise ValueError(msg, line, "at", line[p:])
ValueError: ('Expected version spec in', 'python-novaclient;python_version>="2.7"',
  ↪ 'at', ';python_version>="2.7"')
```

This means that the `pip` command is too old to properly parse Python environment markers `<https://www.python.org/dev/peps/pep-0508/>`; `pip` version 8.1.2 is the first one known to work well but version 9.0.0 improves support.

To fix the issue, please upgrade `pip` to (at least) version 9.0.0:

```
pip install --upgrade 'pip>=9.0.0'
```

3.4.6 Installation fails complaining about version of setuptools

While trying to install ElastiCluster on CentOS/RHEL machines with `pip install`, you get a long error report that goes along these lines:

```
Obtaining file:///.../elasticsearch/src
Running setup.py egg_info for package from file:///.../elasticsearch/src

The required version of setuptools (>=20.6.8) is not available,
and can't be installed while this script is running. Please
install a more recent version first, using
'easy_install -U setuptools'.

(Currently using setuptools 0.9.8 (/.../elasticsearch/lib/python2.7/site-
  ↪ packages))
Complete output from command python setup.py egg_info:
```

```
The required version of setuptools (>=20.6.8) is not available,
and can't be installed while this script is running. Please
install a more recent version first, using
'easy_install -U setuptools'.

(Currently using setuptools 0.9.8 (/.../elasticcluster/lib/python2.7/site-packages))

-----
Cleaning up...
Command python setup.py egg_info failed with error code 2 in /.../elasticcluster/src
Storing complete log in /home/hydra/rmurri/.pip/pip.log
```

To fix the issue, please run these commands instead:

```
pip install six packaging appdirs; pip install --upgrade setuptools
```

Then resume the installation procedure of ElasticCluster from where you left off and run the pip step again.

Warning: Do not heed the advice given in the error message and run the command `easy_install -U setuptools`: it might get you in trouble later on, see next section.

3.4.7 Installation fails with VersionConflict: ... Requirement.parse('setuptools>=17.1')

When trying to install ElasticCluster with `pip install`, amid the installation of dependency packages, you get a long error report that ends with a Python traceback similar to this one (some parts omitted for clarity):

```
Complete output from command .../bin/python -c "import setuptools;__file__='.../build/
↳funcsigs/setup.py';exec(compile(open(__file__).read().replace('\r\n', '\n'), __file_
↳_, 'exec'))" install --record /tmp/pip-P4Xwfz-record/install-record.txt --single-
↳version-externally-managed --install-headers .../include/site/python2.7:
Traceback (most recent call last):

File "<string>", line 1, in <module>
...
File ".../lib/python2.7/site-packages/pkg_resources.py", line 630, in resolve

    raise VersionConflict(dist, req) # XXX put more info here

pkg_resources.VersionConflict: (setuptools 0.9.8 (.../lib/python2.7/site-packages),
↳Requirement.parse('setuptools>=17.1'))
```

To fix the issue, run this command instead:

```
pip install six packaging appdirs; pip install --upgrade setuptools
```

Then resume the installation procedure of ElasticCluster from where you left off and run the pip step again.

This problem has so far only been reported on CentOS 7.x and apparently only happens when both the following conditions are met:

1. The version of `setuptools` initially installed in the virtual environment was less than the one required by ElasticCluster (e.g. CentOS' default 0.9.8);
2. The `setuptools` Python package was updated by running `easy_install -U setuptools`.

3.4.8 Upgrading `setuptools` fails with `ImportError: No module named extern`

Updating `setuptools` by means of the `easy_install` command fails with a traceback like the one below:

```
$ easy_install -U setuptools
Traceback (most recent call last):
  File "/tmp/e/bin/easy_install", line 9, in <module>
    load_entry_point('setuptools==27.3.0', 'console_scripts', 'easy_install')()
  File "/tmp/e/lib/python2.7/site-packages/pkg_resources.py", line 378, in load_entry_
↪point
    return get_distribution(dist).load_entry_point(group, name)
  File "/tmp/e/lib/python2.7/site-packages/pkg_resources.py", line 2566, in load_
↪entry_point
    return ep.load()
  File "/tmp/e/lib/python2.7/site-packages/pkg_resources.py", line 2260, in load
    entry = __import__(self.module_name, globals(),globals(), ['__name__'])
  File "build/bdist.linux-x86_64/egg/setuptools/__init__.py", line 10, in <module>
  File "build/bdist.linux-x86_64/egg/setuptools/extern/__init__.py", line 1, in
↪<module>
ImportError: No module named extern
```

To fix the issue, run this command instead:

```
pip install six packaging appdirs; pip install --upgrade setuptools
```

Then resume the installation procedure of `ElastiCluster` from where you left off and run the `pip` step again.

This problem has so far only been reported on CentOS 7.x platforms.

3.4.9 Installation fails with: “fatal error: ffi.h: No such file or directory”

While trying to install `ElastiCluster` with `pip install`, you get a long error report that ends with these lines:

```
No package 'libffi' found
c/_cffi_backend.c:15:17: fatal error: ffi.h: No such file or directory
  #include <ffi.h>
                ^
compilation terminated.
error: Setup script exited with error: command 'gcc' failed with exit status 1
```

To fix the issue on Debian/Ubuntu computers, please install package `libffi-dev` prior to attempting to install `ElastiCluster`:

```
sudo apt-get install libffi-dev
```

To fix the issue on RHEL/CentOS computers, please install package `libffi-devel`:

```
yum install libffi-devel # run this as root
```

After installing the FFI devel packages, repeat the installation steps for `ElastiCluster`.

(Note: this error comes from missing or badly installed dependency software for `ElastiCluster`; you might want to repeat the steps in section *Install required dependencies* again and be sure they run through successful completion.)

3.4.10 Installation fails with: “fatal error: openssl/opensslv.h: No such file or directory”

While trying to install ElastiCluster with `pip install`, you get a long error report that ends with lines like these:

```
building '_openssl' extension
x86_64-linux-gnu-gcc -pthread -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -fno-
↳strict-aliasing -Wdate-time -D_FORTIFY_SOURCE=2 -g -fstack-protector-strong -
↳Wformat -Werror=format-security -fPIC -I/usr/include/python2.7 -c build/temp.linux-
↳x86_64-2.7/_openssl.c -o build/temp.linux-x86_64-2.7/build/temp.linux-x86_64-2.7/_
↳openssl.o
build/temp.linux-x86_64-2.7/_openssl.c:423:30: fatal error: openssl/opensslv.h: No
↳such file or directory
compilation terminated.
error: command 'x86_64-linux-gnu-gcc' failed with exit status 1
```

To fix the issue on Debian/Ubuntu computers, please install package `libssl-dev` prior to attempting to install ElastiCluster:

```
sudo apt-get install libssl-dev
```

To fix the issue on RHEL/CentOS computers, please install package `libffi-devel`:

```
yum install openssl-devel # run this as root
```

After installing the OpenSSL devel packages, repeat the installation steps for ElastiCluster.

(*Note:* this error comes from missing or badly installed dependency software for ElastiCluster; you might want to repeat the steps in section *Install required dependencies* again and be sure they run through successful completion.)

3.4.11 Installation fails with: “fatal error: Python.h: No such file or directory”

While trying to install ElastiCluster with `pip install`, you get a long error report that ends with lines like these:

```
fatal error: Python.h: No such file or directory
  #include <Python.h>
           ^
compilation terminated.
error: command 'gcc' failed with exit status 1
```

To fix the issue on Debian/Ubuntu computers, please install packages `libc6-dev` and `python-dev` prior to attempting to install ElastiCluster:

```
sudo apt-get install libc6-dev python-dev
```

To fix the issue on RHEL/CentOS computers, please install packages `glibc-devel` and `python-devel`:

```
yum install glibc-devel python-devel # run this as root
```

After installing the packages, repeat the installation steps for ElastiCluster.

(*Note:* this error comes from missing or badly installed dependency software for ElastiCluster; you might want to repeat the steps in section *Install required dependencies* again and be sure they run through successful completion.)

3.4.12 Installation fails with: “unable to execute gcc: No such file or directory”

While trying to install ElastiCluster with `pip install`, you get a long error report that ends with lines like these:

```
Complete output from command python setup.py egg_info:
unable to execute gcc: No such file or directory
unable to execute gcc: No such file or directory

No working compiler found, or bogus compiler options
passed to the compiler from Python's distutils module.
See the error messages above.
(If they are about -mno-fused-madd and you are on OS/X 10.8,
see http://stackoverflow.com/questions/22313407/ .)
```

To fix the issue on Debian/Ubuntu computers, please install package `gcc` prior to attempting to install ElastiCluster:

```
sudo apt-get install gcc libc6-dev
```

To fix the issue on RHEL/CentOS computers, please install package `gcc`:

```
yum install gcc glibc-devel # run this as root
```

After installing the GCC packages, repeat the installation steps for ElastiCluster.

(*Note:* this error comes from missing or badly installed dependency software for ElastiCluster; you might want to repeat the steps in section *Install required dependencies* again and be sure they run through successful completion.)

3.4.13 Installation fails with Too many levels of symbolic links

Running `pip install` to install ElastiCluster fails with a Python error like the one below (some parts omitted for brevity):

```
Cleaning up...
Exception:
Traceback (most recent call last):
...
  File ".../lib/python2.7/site-packages/pip/download.py", line 420, in unpack_file_url
    shutil.copytree(source, location)
  File "/usr/lib64/python2.7/shutil.py", line 208, in copytree
    raise Error, errors
Error: [..., "[Errno 40] Too many levels of symbolic links: '/home/centos/
↪elasticcluster/elasticcluster/share/playbooks/roles/roles/roles/roles/roles/roles/
↪roles/roles/roles/roles/roles/roles/roles/roles/roles/roles/roles/roles/roles/roles/
↪roles/roles/roles/roles/roles/roles/roles/roles/roles/roles/roles/roles/roles/roles/
↪roles/roles/roles/roles/roles/roles/roles/roles/roles'") ]
```

This error only happens because the `pip` program is too old. Upgrade `pip` by running the command:

```
pip install --upgrade "pip>=7.1.0"
```

3.4.14 Setup of RHEL/CentOS 7 clusters fails immediately

While running `elasticcluster setup` (or in the final part of `elasticcluster start`) an Ansible playbook is run, but it stops as early as the first task. A long error message follows, resembling this one:

```
PLAY [Common setup for all hosts] *****

TASK [setup] *****
fatal: [worker001]: FAILED! => {"changed": false, "failed": true, "invocation": {
  ↪ "module_name": "setup"}, "module_stderr": "sudo: sorry, you must have a tty to run_",
  ↪ sudo\n", "module_stdout": "", "msg": "MODULE FAILURE", "parsed": false}
```

The key error message here is `sudo: sorry, you must have a tty to run sudo`. Apparently RHEL and CentOS ship with a default configuration that requires an interactive terminal to run `sudo`; this is not there when `sudo` is run remotely from Ansible.

A solution is to use the ElastiCluster configuration key `image_userdata` to alter `sudo` behavior to allow TTY-less operation. For example:

```
[cluster/sge]
image_userdata=#!/bin/bash
  echo 'Defaults:centos !requiretty' > /etc/sudoers.d/999-requiretty && chmod 440 /
  ↪ etc/sudoers.d/999-requiretty
```

Another solution is to turn **SSH pipelining** off. There are two ways of doing this:

1. Add the line `ansible_ssh_pipelining=no` in the cluster `[setup/*]` section. For instance:

```
[setup/slurm]
provider=ansible
ansible_ssh_pipelining=no
# ...rest of sections is unchanged...
```

2. Or set the `ANSIBLE_SSH_PIPELINING` environment variable to the value `no`. For example:

```
env ANSIBLE_SSH_PIPELINING=no elasticluster setup mycluster
```

You can read a more complete explanation in the book [Ansible: Up and Running](#) by Lorin Hochstein.

3.4.15 Setup of Ubuntu 16.04 (“xenial”) clusters fails immediately

While running `elasticluster setup` (or in the final part of `elasticluster start`) an Ansible playbook is run, but it stops as early as the first task. A long error message follows, resembling this one:

```
PLAY [Common setup for all hosts] *****

TASK [setup] *****
fatal: [master001]: FAILED! => {"changed": false, "failed": true, "module_stderr": "",
  ↪ "module_stdout": "/bin/sh: 1: /usr/bin/python: not found\r\n", "msg": "MODULE_",
  ↪ FAILURE", "parsed": false}
...

```

The key part of the error message is: `/usr/bin/python: not found`; **Ubuntu 16.04 does not install Python 2.x** by default.

To fix the issue install package `python` on the Ubuntu VMs:

- run `sudo apt install python` in a VM started with the base image;
- make a snapshot;
- use that snapshot as the base for ElastiCluster.

Additional support will be required in ElastiCluster to automate these steps, see [issue #304](#)

3.4.16 Issues when installing from source on MacOSX

Warning: Installation and testing of ElasticCluster on MacOSX is not currently part of the development or the release cycle. So these notes could be severely out of date. Please report issues and seek for solutions on the ElasticCluster mailing-list.

When installing ElasticCluster on MacOSX 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/-Tmp-/easy_install-CrTFFL/requests-2.4.3/egg-dist-tmp-
↳JZ2MOD
Adding requests 2.4.3 to easy-install.pth file
Traceback (most recent call last):
  File "setup.py", line 109, in <module>
    'elasticsearch = elasticsearch.main:main',
  File "/System/Library/Frameworks/Python.framework/Versions/2.6/lib/python2.6/
↳distutils/core.py", line 152, in setup
    dist.run_commands()
  File "/System/Library/Frameworks/Python.framework/Versions/2.6/lib/python2.6/
↳distutils/dist.py", line 975, in run_commands
    self.run_command(cmd)
  File "/System/Library/Frameworks/Python.framework/Versions/2.6/lib/python2.6/
↳distutils/dist.py", line 995, in run_command
    cmd_obj.run()
  File "/Users/michela/elasticsearch/build/setuptools/setuptools/command/install.py",
↳line 65, in run
    File "/Users/michela/elasticsearch/build/setuptools/setuptools/command/install.py",
↳line 115, in do_egg_install
    File "/Users/michela/elasticsearch/build/setuptools/setuptools/command/easy_install.
↳py", line 360, in run

    File "/Users/michela/elasticsearch/build/setuptools/setuptools/command/easy_install.
↳py", line 576, in easy_install

    File "/Users/michela/elasticsearch/build/setuptools/setuptools/command/easy_install.
↳py", line 627, in install_item

    File "/Users/michela/elasticsearch/build/setuptools/setuptools/command/easy_install.
↳py", line 682, in process_distribution

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

```

File "/Users/michela/elasticsearch/build/setuptools/setuptools/command/easy_install.
↪py", line 627, in install_item

File "/Users/michela/elasticsearch/build/setuptools/setuptools/command/easy_install.
↪py", line 659, in process_distribution

File "/Users/michela/elasticsearch/build/setuptools/setuptools/command/easy_install.
↪py", line 532, in install_egg_scripts

File "/Users/michela/elasticsearch/build/setuptools/setuptools/command/easy_install.
↪py", line 734, in install_wrapper_scripts

File "/private/var/folders/tZ/tZ2B3RaeGVq7+ptdJIbdj++++TI/-Tmp-/easy_install-qch0dG/
↪python-keystoneclient-0.11.1/pbr-0.10.0-py2.6.egg/pbr/packaging.py", line 512, in_
↪override_get_script_args
AttributeError: 'NoneType' object has no attribute 'get_script_header'

```

you probably need to install *pbr* manually using:

```
pip install pbr
```

3.4.17 Error “ImportError: No module named anyjson” on MacOSX

Warning: Installation and testing of ElastiCluster on MacOSX is not currently part of the development or the release cycle. So these notes could be severely out of date. Please report issues and seek for solutions on the ElastiCluster [mailing-list](#).

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

```

Traceback (most recent call last):
File "/Users/michela/el2/bin/elasticsearch", line 9, in <module>
load_entry_point('elasticsearch==1.1-dev', 'console_scripts', 'elasticsearch')()
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/elasticsearch/__init__.py", line 33, in
↪<module>
File "build/bdist.macosx-10.6-universal/egg/elasticsearch/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 elasticsearch installation:

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

3.5 Playbooks distributed with ElastiCluster

ElastiCluster uses [Ansible](#) to configure the VM cluster based on the options read from the configuration file. This chapter describes the Ansible playbooks bundled¹ with ElastiCluster and how to use them.

Contents

- *Playbooks distributed with ElastiCluster*
 - *Setup variables*
 - * *General setup variables*
 - *Compute clusters*
 - * *SLURM*
 - * *GridEngine*
 - * *Hadoop + Spark*
 - * *HTCondor*
 - * *Kubernetes*
 - *Filesystems and storage*
 - * *CephFS*
 - * *GlusterFS*
 - * *OrangeFS/PVFS2*
 - *Add-on software*
 - * *Anaconda*
 - * *Ansible*
 - * *CUDA*
 - * *EasyBuild*
 - * *Ganglia*
 - * *IPython cluster*
 - * *JupyterHub*
 - * *R language*

¹ ElastiCluster playbooks can be found in the `elasticsearch/share/playbooks` directory of the source code. You are free to copy, customize and redistribute them under the terms of the GNU General Public License version 3 or (at your option) any later version.

* *SAMBA*

3.5.1 Setup variables

In some cases, extra variables can be set to playbooks to modify its default behavior. In these cases, you can either define a variable global to the cluster using:

```
global_var_<varname>=<value>
```

or, if the variable must be defined only for a specific group of hosts:

```
<groupname>_var_<varname>=<value>
```

For example:

```
slurm_worker_allow_reboot=yes
```

General setup variables

The following customization variables apply to all ElastiCluster playbooks.

Variable name	De-fault	Description
allow_reboot	no	Allow rebooting nodes if needed. Be careful if this is set when resizing clusters, as you may lose running jobs.
insecure_http	no (default)	If no (default), require that web sites, from where software is downloaded, present a valid SSL/TLS certificate. However, it may happen that the base OS trusted certificates repository is not fully up-to-date and this verification fails. (See, for instance, issue #539). In these cases, setting this option to yes allows the playbooks to continue (at the expense of some security).
multiuse_cluster	no	Install NIS/YP. The first node in the master class will be assigned the role of the YP master server; additional nodes in the “master” class (if any) will be YP slave servers.
ssh_password	yes	Allow users to log in via SSH by providing a password. Note: the default in ElastiCluster is the opposite of what all major GNU/Linux distributions do.
upgrade_packages	yes	Upgrade all installed to the latest available version. Setting this to no allows speedier setup when starting from cluster snapshots. <i>Note:</i> even when set to no, some packages may still be upgraded to satisfy dependencies of other packages that need to be installed.

3.5.2 Compute clusters

The following playbooks are available for installing compute clusters (batch-queuing or otherwise) with ElastiCluster.

SLURM

Supported on:

- Ubuntu 12.04 and later
- Debian 7 (“wheezy”) and 8 (“jessie”)
- RHEL/CentOS 6.x and 7.x

This playbook installs the [SLURM](#) batch-queueing system.

You are supposed to only define one `slurm_master` and multiple `slurm_worker`. The first will act as login node, NFS server for the `/home` filesystem, and runs the SLURM scheduler and accounting database; the workers will only execute the jobs. A `slurm_submit` role allows you to optionally install “SLURM client” nodes, i.e., hosts whose only role in the cluster is to submit jobs and query the queue status.

Ansible group	Action
<code>slurm_master</code>	SLURM controller/scheduler node; also runs the accounting storage daemon <i>slurmdbd</i> and its MySQL/MariaDB backend.
<code>slurm_worker</code>	SLURM execution node: runs the <i>slurmd</i> daemon.
<code>slurm_submit</code>	SLURM client: has all the submission and query commands installed, but runs no daemon.

The following example configuration sets up a SLURM batch-queueing cluster using 1 front-end and 4 execution nodes:

```
[cluster/slurm]
master_nodes=1
worker_nodes=4
ssh_to=master
setup_provider=slurm
# ...

[setup/slurm]
master_groups=slurm_master
worker_groups=slurm_worker
# ...
```

You can combine the SLURM playbook with other add-on software. For instance, you can install the Ganglia monitoring system alongside with SLURM; in this case the `setup` stanza will look like:

```
[setup/slurm+ganglia]
frontend_groups=slurm_master, ganglia_master
compute_groups=slurm_worker, ganglia_monitor
...
```

When combined with the CUDA add-on, and if any actual NVIDIA GPU devices are found on the compute nodes, SLURM will be configured with the GPU devices as [GRES](#) resources so that you can request the use of GPUs in your jobs by passing the `--gres=gpu:...` option to `sbatch`:

```
# request 2 GPU devices
sbatch --gres=gpu:2 my_gpgpu_job.sh
```

Note that compute nodes need to be given the `cuda` add-on playbook in order for CUDA GPU detection and configuration to work. For example, the following configuration will detect and configure NVIDIA GPUs on all compute nodes (but not on the front-end):

```
[setup/slurm+gpu]
frontend_groups=slurm_master
compute_groups=slurm_worker, cuda
...
```

Extra variables can be set by editing the `setup/` section:

Table 3.1: cgroup-related SLURM settings

Variable	Default value	Description
slurm_allowedramspace 1	100	Max percentage of RAM that can be allocated to a job. If <code>slurm_constrainramspace</code> (see below) is <code>yes</code> , then this limit is applied to a job's <i>real memory</i> usage; otherwise, this limit is summed with <code>slurm_allowedswapspace</code> to cap the <i>virtual memory</i> usage (see SLURM's <code>VSizeFactor</code> configuration parameter).
slurm_allowedswapspace	1	Max percentage of virtual memory (in addition to the real memory) that can be allocated to a job. This value is summed with <code>slurm_allowedramspace</code> to cap a job's total <i>virtual memory</i> usage. You might want to set this limit to a much higher value when using GPUs, as GPU memory might be accounted in the job's virtual memory.
slurm_constrainramspace	yes	Only used if “<code>slurm_taskplugin</code>“ is set to “<code>task/cgroup</code>“. If set to <code>yes</code> then SLURM constrains the job's RAM usage by setting the memory soft limit to the allocated memory and the hard limit to the allocated memory * <code>slurm_allowedramspace</code> (see below). This can add stability to a system when there are multiple misbehaving jobs that allocate large amounts of memory, but can be problematic with jobs using GPUs (since the memory used on the GPU seems to be accounted against the job's own CPU RAM consumption).
slurm_constrainswapspace	yes	Only used if “<code>slurm_taskplugin</code>“ is set to “<code>task/cgroup</code>“. If set to <code>yes</code> then SLURM kills jobs whose virtual memory usage exceeds allocated memory * <code>slurm_allowedswapspace</code> (see below).
slurm_fastschedule	1	Value of <code>FastSchedule</code> in <code>slurm.conf</code>
slurm_jobacctgatherfrequency	60	Value of <code>JobAcctGatherFrequency</code> in <code>slurm.conf</code>
slurm_jobacctgatheretype	jobacct_gather/ linux	Value of <code>JobAcctGratherType</code> in <code>slurm.conf</code>
slurm_maxarraysize	1000	Maximum size of an array job
slurm_maxjobcount	10000	Maximum nr. of jobs actively managed by the SLURM controller (i.e., pending and running)
slurm_proctracktype	proctrack/ linuxproc	Value of <code>ProcTrackType</code> in <code>slurm.conf</code>
slurm_returntoservice	2	Value of <code>ReturnToService</code> in <code>slurm.conf</code>
slurm_selecttype	select/ cons_res	Value of <code>SelectType</code> in <code>slurm.conf</code>
slurm_selecttypeparameters	CR_Core_Memory	Value of <code>SelectTypeParameters</code> in <code>slurm.conf</code>
slurm_taskplugin	task/none	Value of <code>TaskPlugin</code> in <code>slurm.conf</code>

Note that the `slurm_*` extra variables need to be set *globally* (e.g., `global_var_slurm_selecttype`) because the SLURM configuration file must be identical across the whole cluster.

Global variable `multiuser_cluster` controls whether the NIS/YP software is installed on the cluster (NIS master

on the cluster master node, compute nodes are NIS slaves) to make it easier to add users to the cluster (just run the `adduser` command on the master).

The “SLURM” playbook depends on the following Ansible roles being available:

- `slurm-common`
- `slurm-client`
- `slurm-master`
- `slurm-worker`

In order for the NFS exported home directory to be mountable from the cluster’s compute nodes, security groups on OpenStack need to permit all UDP traffic between all cluster nodes.

GridEngine

Tested on:

- CentOS 6.x and 7.x
- Ubuntu 14.04 (“trusty”) and 16.04 (“xenial”)
- Debian 8 (“jessie”)

ansible groups	role
<code>gridengine_master</code>	Scheduler, admin, and submission host
<code>gridengine_worker</code>	Compute (exec) node and submission host

This playbook installs `GridEngine` using the packages distributed with Ubuntu, Debian, or CentOS, and creates a basic working configuration.

You are supposed to only define one `gridengine_master` and multiple `gridengine_worker`. The first acts as login node, fileserver, and runs the master scheduler (SGE `qmaster`), whereas the others will only execute jobs (SGE `execd`).

The `/home` filesystem is exported *from* the gridengine “master” to the worker nodes. The cell directory `$SGE_ROOT/$SGE_CELL/common` directory is shared from the gridengine server to the compute nodes (via NFS).

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_worker
...
```

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

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

Global variable `multiuser_cluster` controls whether the NIS/YP software is installed on the cluster (NIS master on the cluster master node, compute nodes are NIS slaves) to make it easier to add users to the cluster (just run the `adduser` command on the master).

Hadoop + Spark

Supported on:

- Ubuntu 16.04, 14.04
- Debian 8 (“jessie”)

This playbook installs a [Hadoop 2.x](#) cluster with [Spark](#) and [Hive](#), using the packages provided by the Apache [Bigtop](#) project. The cluster comprises the HDFS and YARN services: each worker node acts both as a HDFS “DataNode” and as a YARN execution node; there is a single master node, running YARN’s “ResourceManager” and “JobHistory”, and Hive’s “MetaStore” services.

Ansible group	Action
<code>hadoop_master</code>	Install the Hadoop cluster master node: run YARN “ResourceManager” and Hive “MetaStore” server. In addition, install a PostgreSQL server to host Hive metastore tables.
<code>hadoop_workers</code>	Install a YARN+HDFS node: run YARN’s “NodeManager” and HDFS’ “DataNode” services. This is the group of nodes that actually provide the storage and execution capacity for the Hadoop cluster.

HDFS is only formatted upon creation; if you want to reformat/zero out the HDFS filesystem you need to run the `hdfs namenode -format` command yourself. No rebalancing is done when adding or removing data nodes from the cluster.

Nota bene:

1. Currently ElastiCluster turns off HDFS permission checking: therefore Hadoop/HDFS clusters installed with ElastiCluster are only suitable for shared usage by *mutually trusting* users.
2. Currently ElastiCluster has no provision to vacate an HDFS data node before removing it. Be careful when shrinking a cluster, as this may lead to data loss!

The following example configuration sets up a Hadoop cluster using 4 storage+execution nodes:

```
[cluster/hadoop+spark]
master_nodes=1
worker_nodes=4
ssh_to=master

setup_provider=hadoop+spark
# ...

[setup/hadoop+spark]
provider=ansible
master_groups=hadoop_master
worker_groups=hadoop_worker
```

Global variable `multiuser_cluster` controls whether the NIS/YP software is installed on the cluster (NIS master on the cluster master node, compute nodes are NIS slaves) to make it easier to add users to the cluster (just run the `adduser` command on the master).

The following variables can be used to control the defaults for running Spark applications. (Note that they set a *default*, hence can be overridden by applications when creating a Spark context; on the other hand, these defaults are *exactly* what is used when running `pyspark` or a Jupyter notebook with Spark support.)

Table 3.2: Spark settings

Variable	Default value	Description
spark_driver_memory_mb 1	(Free memory on master node / nr. of CPUs of master node)	Used to set spark.driver.memory: maximum amount of memory (counted in MBs) that a Spark “driver” process is allowed to use.
spark_driver_maxresults 1	80% of “spark_driver_memory_mb”	Used to set spark.driver.maxResultSize: Limit of total size (amount in MBs) of serialized results of all partitions for each Spark action (e.g. collect)
spark_executor_memory_mb 1	Max free memory on worker node / max nr. of CPUs on a node)	Used to set spark.executor.memory: Maximum amount of memory (counted in MBs) that a Spark “executor” process is allowed to use.
spark_python_worker_memory_mb 1	50% of “spark_executor_memory_mb”	Used to set spark.python.worker.memory: Maximum amount of memory (counted in MBs) to use per Python worker process during aggregation. If the memory used during aggregation goes above this amount, Spark starts spilling the data into disks.

HTCondor

Tested on:

- Debian 8.x
- Ubuntu 14.04

ansible groups	role
condor_master	Act as scheduler, submission and execution host.
condor_worker	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=htcondor
frontend_nodes=1
compute_nodes=2
ssh_to=frontend
# ...

[setup/htcondor]
frontend_groups=condor_master
compute_groups=condor_worker
# ...
```

Kubernetes

Supported on:

- Ubuntu 16.04
- RHEL/CentOS 7.x

This playbook installs the *Kubernetes* container management system on each host. It is configured using kubeadm. Currently only 1 master node is supported.

To force the playbook to run, add the Ansible group `kubernetes`. The following example configuration sets up a kubernetes cluster using 1 master and 2 worker nodes, and additionally installs flannel for the networking (canal is also available):

```
[cluster/kubernetes]
master_nodes=1
worker_nodes=2
ssh_to=master
setup_provider=kubernetes
# ...

[setup/kubernetes]
master_groups=kubernetes_master
worker_groups=kubernetes_worker
# ...
```

SSH into the cluster and execute ‘`sudo kubectl –kubeconfig /etc/kubernetes/admin.conf get nodes`’ to view the cluster.

3.5.3 Filesystems and storage

The following playbooks are available for installing storage clusters and distributed filesystems with ElastiCluster. These can be used standalone, or mixed with compute clusters to provide additional storage space or more performant cluster filesystems.

CephFS

Supported on:

- Ubuntu 16.04, 14.04
- Debian 8 (“jessie”), 9 (“stretch”)
- CentOS 6.x and 7.x

Ansible group	Action
<code>ceph_mon</code>	Install Ceph server software and configure this host to run the MON (monitor) service. There <i>must</i> be at least one MON node in any Ceph cluster.
<code>ceph_osd</code>	Install Ceph server software and configure this host to run the OSD (object storage device) service. There <i>must</i> be at least three OSD nodes in a Ceph cluster.
<code>ceph_mds</code>	Install Ceph server software and configure this host to run the MDS (meta-data server) service. This node is optional but CephFS is only available if at least one MDS is available in the cluster.
<code>ceph_client</code>	Install required packages to make usage of Ceph Storage Cluster and CephFS possible. Any mount point listed in <code>CEPH_MOUNTS</code> will be created and the corresponding filesystem mounted.

This will install a Ceph Storage Cluster with CephFS. Actual data storage is done in the OSD nodes, on each node under directory `/var/lib/ceph/osd/ceph-NNN`. All hosts in group `ceph_client` can then mount this filesystem, using either the [CephFS kernel driver](#) or the [CephFS FUSE driver](#).

Management of the cluster is possible from any node (incl. `ceph_client` nodes) using the `client.admin` key (deployed in file `/etc/ceph/ceph.client.admin.keyring`, by default only readable to the `root` user).

The Ceph and CephFS behavior can be changed by defining the following variables in the `setup/` section:

Table 3.3: Ceph/CephFS variables in ElastiCluster

Variable	Default value	Description
<code>ceph_release</code>	<code>luminous</code>	Name of Ceph release to install, e.g. “luminous” or “jewel”. Note that not all releases are available on all platforms; for instance, selecting the “hammer” release on Ubuntu 16.04 will install “jewel” instead.
<code>ceph_osd_pool_size</code>	<code>2</code>	Default number of object replicas in a pool.
<code>ceph_osd_pool_size</code>	<code>computed according to: http://docs.ceph.com/docs/master/rados/operations/placement-groups/#a-preselection-of-pg-num</code>	Default number of PGs in a pool.
<code>ceph_metadata_pg_num</code>	<code>1/8 * ceph_osd_pool_size</code>	Number of PGs for the CephFS metadata pool.
<code>ceph_data_pg_num</code>	<code>7/8 * ceph_osd_pool_size</code>	Number of PGs for the CephFS data pool.

More detailed information can be found in the [ceph role README](#).

Note:

- In contrast with similar ElastiCluster playbooks, the CephFS playbook does *not* automatically mount CephFS on client nodes.
- This playbook’s defaults differ from Ceph’s upstream defaults in the following ways:
 - default replication factor for objects in a pool is 2 (Ceph’s upstream is 3)
 - the minimum number of copies of an object that a pool should have to continue being operational is 1 (Ceph’s upstream is 2).

In both cases, the different default is motivated by the assumption that cloud-based storage is “safer” than normal disk-based storage due to redundancy and fault-tolerance mechanisms at the cloud IaaS level.

- The [CephFS kernel driver](#) for the “Luminous” release requires features that are only present in the Linux kernel from version 4.5 on. At the time of this writing, a >4.5 kernel is only installed by default on Debian 9 “stretch”. To mount a “Luminous” CephFS on any other Linux distribution, you will have to either use the [CephFS FUSE driver](#) or tell Ceph not to use tunables v5:

```
sudo ceph osd crush tunables hammer
```

The following example configuration sets up a CephFS cluster using 1 MON+MDS node, 5 OSD nodes and providing 3 replicas for each object:

```
[setup/ceph1]
mon_groups=ceph_mon,ceph_mds
osd_groups=ceph_osd
client_groups=ceph_client
```

```

global_var_ceph_release=luminous
global_var_ceph_osd_pool_size=3

[cluster/ceph1]
setup=ceph1

mon_nodes=1
osd_nodes=5
client_nodes=1
ssh_to=client

# .. cloud-specific params ...

```

This example configuration sets up a CephFS cluster using 3 MON+OSD nodes, 1 MDS nodes and sets explicitly the number of PGs to use for CephFS metadata and data:

```

[setup/ceph2]
mon_groups=ceph_mon,ceph_osd
mds_groups=ceph_mds
client_groups=ceph_client

global_var_ceph_release=luminous
global_var_ceph_metadata_pg_num=1024
global_var_ceph_data_pg_num=8192

[cluster/ceph2]
setup=ceph2

mon_nodes=3
mds_nodes=1
client_nodes=1
ssh_to=client

# .. cloud-specific params ...

```

GlusterFS

Supported on:

- Ubuntu 14.04 and later
- Debian 8 and later
- RHEL/CentOS 6.x, 7.x

ansible groups	action
glusterfs_server	Run a GlusterFS server with a single <i>brick</i>
glusterfs_client	Install gluster client and (optionally) mount a GlusterFS filesystem.

This will install a GlusterFS using all the `glusterfs_server` nodes as servers with a single *brick* located in directory `/srv/glusterfs`, and any `glusterfs_client` to mount this filesystem over directory `/glusterfs`.

To manage the GlusterFS filesystem you need to connect to a `gluster_server` node.

By default the GlusterFS volume is “pure distributed”: i.e., there is no redundancy in the server setup (if a server goes offline, so does the data that resides there), and neither is the data replicated nor striped, i.e., replica and stripe number

is set to 1. This can be changed by defining the following variables in the *setup/* section:

variable name	default	description
gluster_stripes	no stripe	set the stripe value for default volume
gluster_replicas	no replica	set replica value for default volume
gluster_redundancy	0	nr. of servers that can fail or be offline without affecting data availability

Note that setting `gluster_redundancy` to a non-zero value will force the volume to be “dispersed”, which is incompatible with striping and replication. In other words, the `gluster_redundancy` option is incompatible with `gluster_stripes` and/or `gluster_replicas`. You can read more about the GlusterFS volume types and permitted combinations at <http://docs.gluster.org/en/latest/Administrator%20Guide/Setting%20Up%20Volumes/>.

The following example configuration sets up a GlusterFS cluster using 8 data nodes and providing 2 replicas for each file:

```
[cluster/gluster]
  client_nodes=1
  server_nodes=8
  ssh_to=client

  setup_provider=gluster
  # ... rest of cluster params as usual ...

[setup/gluster]
  provider=ansible

  client_groups=glusterfs_client
  server_groups=glusterfs_server,glusterfs_client

  # set replica and stripe parameters
  server_var_gluster_replicas=2
  server_var_gluster_stripes=1
```

The following example configuration sets up a dispersed GlusterFS volume using 6 data nodes with redundancy 2, i.e., two servers can be offlined without impacting data availability:

```
[cluster/gluster]
  client_nodes=1
  server_nodes=6
  ssh_to=client

  setup_provider=gluster
  # ... rest of cluster params as usual ...

[setup/gluster]
  provider=ansible

  client_groups=glusterfs_client
  server_groups=glusterfs_server,glusterfs_client

  # set redundancy and force "dispersed" volume
  server_var_gluster_redundancy=2
```

The “GlusterFS” playbook depends on the following Ansible roles being available:

- `glusterfs-common`
- `glusterfs-client`

- glusterfs-server

OrangeFS/PVFS2

Tested on:

- Ubuntu 14.04

ansible groups	role
pvfs2_meta	Run the pvfs2 metadata service
pvfs2_data	Run the pvfs2 data service
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 addition, 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+orange]
frontend_nodes=1
compute_nodes=10
orange_nodes=10
ssh_to=frontend
setup_provider=ansible_slurm+orange
...

[setup/ansible_slurm+orange]
frontend_groups=slurm_master,pvfs2_client
compute_groups=slurm_worker,pvfs2_client
orange_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.

3.5.4 Add-on software

The following playbooks add functionality or software on top of a cluster (e.g., monitoring with web-based dashboard). They can also be used stand-alone (e.g., JupyterHub).

Anaconda

Supported on:

- Ubuntu 14.04 and later
- RHEL/CentOS 6.x and 7.x

Ansible group	Action
anaconda	Install the Anaconda Python distribution

This playbook installs the [Anaconda](#) Python distribution. Using customization variables you can choose whether to install the Python 2.7 or Python 3 version, and whether to make the Anaconda Python interpreter the default Python interpreter for logged-in users.

The following variables may be set to alter the role behavior:

Table 3.4: Anaconda role variables in ElastiCluster

Variable	Default value	Description
<code>anaconda_version</code>	5	Version of the Anaconda Python distribution to install
<code>anaconda_python_version</code>	2	Anaconda comes with either a Python2 or a Python3 interpreter – choose which one you want here.
<code>anaconda_in_path</code>	yes	whether the Python interpreter from Anaconda should be made the first match in users' shell <code>\$PATH</code>

For instance, the following configuration snippet requests that ElastiCluster and Ansible install Anaconda Python3 on a SLURM cluster, and make it the default interpreter on the frontend node only:

```
[setup/slurm+anaconda]
# ... same as usual SLURM setup, but:
master_groups=slurm_master,anaconda
worker_groups=slurm_worker,anaconda

# use Anaconda Python3 flavor
global_var_anaconda_python_version=3

# make it default for logged-in users on master node only
master_var_anaconda_in_path=yes
worker_var_anaconda_in_path=no
```

The code from this role is a minor modification of the `ansible-anaconda` playbook written by Andrew Rothstein, and as such maintains the original distribution license. See the accompanying `LICENSE` file for details.

Ansible

Supported on:

- Ubuntu 12.04 and later
- RHEL/CentOS 6.x and 7.x

This playbook installs the [Ansible](#) orchestration and configuration management system on each host. There is not much clustering happening here; this playbook is provided in case you want to be able to run Ansible playbooks from inside the cluster (as opposed to always running them from the ElastiCluster host).

To force the playbook to run, add the Ansible group `ansible` to any node. The following example configuration sets up a SLURM batch-queuing cluster using 1 front-end and 4 execution nodes, and additionally installs [Ansible](#) on the front-end:

```
[cluster/slurm]
master_nodes=1
worker_nodes=4
ssh_to=master
setup_provider=slurm+ansible
# ...

[setup/slurm+ansible]
master_groups=slurm_master,ansible
```

```
worker_groups=slurm_worker
# ...
```

CUDA

Tested on:

- Ubuntu 14.04, 16.04
- CentOS 6.x, 7.x

ansible groups	role
cuda	Install the NVIDIA device drivers and the CUDA toolkit and runtime.

This playbook will detect NVIDIA GPU devices on every host where it is run, and install the CUDA toolkit and runtime software, thus enabling the use of GPU accelerators.

Note that in some cases a reboot is necessary in order to correctly install the NVIDIA GPU device drivers; this is only allowed if the global variable `allow_reboot` is `true` – by default, reboots are not allowed so the playbook will fail if a reboot is needed.

A variable can be used to control the version of the CUDA toolkit that will be installed on the nodes:

Variable name	Default value	Description
<code>cuda_version</code>	8.0	What version of the CUDA toolkit to install

The default is to install CUDA toolkit and runtime version 8.0

EasyBuild

Supported on:

- Ubuntu 16.04, 14.04
- Debian 8 (“jessie”), 9 (“stretch”)
- CentOS 6.x and 7.x

Ansible group	Action
easybuild	Install the EasyBuild HPC package manager.

This playbook installs [EasyBuild](#) and its dependencies to provide a working build environment for HPC clusters.

EasyBuild is configured with the following options:

- use [Lmod](#) as the “environment modules” tool, and generate module files with [Lua](#) syntax.
- use [minimal toolchains](#) (including the “dummy” / system compiler toolchain)
- use [generic optimization flags](#) for maximum compatibility

The following variables may be set to alter the role behavior:

Table 3.5: EasyBuild role variables in ElastiCluster

Variable	Default value	Description
EASYBUILD_VERSION	2.8.2	The version of EasyBuild to install. Interpolated into the (default) source archive name.
EASYBUILD_PREFIX	easybuild	Root directory of all the EasyBuild-related paths: source archive, <code>.eb`</code> files repository, installed software, etc.
EASYBUILD_SOFTWARE	Build no software during cluster setup.	List of <code>.eb</code> recipes to build. This is a <i>YAML list</i> , i.e., a comma-separated list of recipe names enclosed in square brackets; for example: <pre>global_var_EASYBUILD_INSTALL=[RELION,OpenMPI]</pre> <i>Beware:</i> the initial EasyBuild invocation will have to build the entire toolchain, so it can take a couple of hours even to install a small and relatively simple package. For this reason, the default value of this variable is the empty list (i.e., do not install any software through EasyBuild).
EASYBUILD_OPTARCH	GENERIC	Optimization flags for building software, see: http://easybuild.readthedocs.io/en/latest/Controlling_compiler_optimization_flags.html#controlling-target-architecture-specific-optimizations-via-optarch By default the “GENERIC” value is used which should produce code compatible with any x86-64 processor.

It is advised to install EasyBuild on the master/frontend node only, and export the software directories from there to compute nodes via NFS, to cut down on build times and to avoid coherency issues.

Ganglia

Tested on:

- Ubuntu 12.04, 14.04, 16.04
- CentOS 6.x, 7.x

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.

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.

JupyterHub

Supported on:

- Ubuntu 16.04, 14.04
- Debian 8 (“jessie”), 9 (“stretch”)
- CentOS 6.x and 7.x

Ansible group	Action
jupyterhub	Install Jupyter and JupyterHub to work with interactive computational notebooks.

Install [JupyterHub](#) to grant multiple users access to [Jupyter](#) notebooks through a web interface. Kernels are installed to run code written in Python 2.7 and 3.x (with [Anaconda Python](#)), BASH (using the OS-provided shell), PySpark (in conjunction with the Hadoop+Spark playbook), R language (if the R add-on is installed, see below), and MATLAB (if installed).

Note: JupyterHub is configured to authenticate users with the GNU/Linux `/etc/passwd` database. So, in order to log in you need to create users first (or set passwords to existing users).

In order to create a new user, run the following commands at the node’s shell prompt:

```
# replace `user_name` with an actual name (e.g. `jsmith`)
sudo adduser user_name
```

In order to set the password for an existing user, run the following commands at the node’s shell prompt:

```
# replace `user_name` with an actual name (e.g. `jsmith`)  
sudo passwd user_name
```

To use the JupyterHub server:

1. Note down the IP address of the server VM created by ElastiCluster
2. In your browser, open <https://server.ip/>
3. Accept the self-signed SSL certificate in the browser
4. Log in using username and password

Note: You must edit the VM security group to allow connections to port 443! (ElastiCluster will not do this automatically.)

The JupyterHub role can be combined with other playbooks (it is advised to add it to the frontend/master node), or can be used to install a stand-alone server. A full example of how to install a JupyterHub stand-alone server can be found at: <https://github.com/gc3-uzh-ch/elasticsearch/blob/master/examples/jupyterhub-on-google.conf>

R language

Supported on:

- Ubuntu 16.04, 14.04
- Debian 8 (“jessie”), 9 (“stretch”)
- CentOS 6.x and 7.x

Ansible group	Action
r	Install the interpreter and a basic libraries for the GNU R language and statistical system.

This playbook installs the [R language](#) interpreter and a few additional libraries. R binaries installed by ElastiCluster come from 3rd-party repositories which (normally) provide more up-to-date releases compared to the OS packages.

The following extra variables can be set to control installation of additional R libraries:

Variable name	Default	Description
r_libraries	[devtools]	List of R packages to install
r_cluster_support	yes if installing R on more than 1 node, “no” otherwise	Whether to install Rmpi and other packages for distributing work across a computing cluster

By default, the `devtools` library is installed so that R packages can be installed directly from their GitHub location. Additionally, R support for MPI and distribution of work across a cluster is available if R support is being deployed to more than 1 host.

Note: Note that the `r_libraries` variable is a *YAML list*. In order to customize its value, you must provide a comma-separated list of library names, enclosed in square brackets.

For instance, the following configuration snippet requests that ElastiCluster and Ansible install R libraries `devtools` and `doSnow` on the compute nodes of a SLURM cluster:

```
[setup/slurm+r]
# ...
compute_groups=slurm_worker,r
compute_r_libraries=[devtools,doSnow]
```

SAMBA

Supported on:

- Ubuntu 16.04, 14.04
- Debian 8 (“jessie”), 9 (“stretch”)
- CentOS 6.x and 7.x

Ansible group	Action
samba	Install and configure the SAMBA suite for serving local files over the network with the SMB/CIFS protocol

This playbook installs the [SAMBA](#) server suite, which implements a server for the SMB/CIFS network filesystem, plus other utilities for integrating Linux/UNIX systems in a Windows environment. Note that ElastiCluster only configures the `smbd` daemon for serving files as a “standalone SMB server” – no other integration with Windows services is included here.

The following extra variables can be set to control the way the SMB server is set up:

Variable name	Default	Description
<code>smb_workgroup</code>	<code>elasticsearch</code>	NetBIOS workgroup name
<code>smb_shares</code>	<i>(no additional shares)</i>	Define additional SMB shares (see example below)

By default, ElastiCluster only configures sharing users’ home directories over SMB/CIFS. Additional shares can be defined by adding a `smb_shares` variable in the `setup/` section. The value of this variable should be a list (comma-separated, enclosed in `[` and `]`) of share definitions; a share definition is enclosed in `{` and `}` and is comprised of comma-separated *key:value* pair; the following *key:value* pair will be acted upon:

name The SMB share name; the string that clients must use to connect to this share

path Local path to the root directory being served

readonly Whether writes are allowed to the share. If `no` (default), then no writes are allowed by any client.

public If `yes`, any user that can connect to the server can read (and also write, depending on the `readonly` setting above) files in the share. If `no` (default), only authenticated users can access the share.

For instance, the following ElastiCluster configuration snippet configures two *additional* shares, one named `public` which serves files off local path `/data/public` to any user who can connect, and one named `secret` which serves files off local path `/data/secret` to authenticated users:

```
[setup/samba]
server_groups=samba
server_smb_shares=[
  { name: 'public', path: '/data/public', readonly: yes, public: yes },
  { name: 'secret', path: '/data/secret', readonly: yes, public: no },
]
```

3.6 ElastiCluster programming API

3.6.1 *elasticcluster*

Overview

Elasticcluster 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 elasticcluster 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 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 further 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:

- **OpenStackCloudProvider** Cloud provider to connect to an OpenStack cloud.
- **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.
- **PickleRepository** 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 elasticcluster

# 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 = elasticcluster.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_worker.
# This corresponds to the documentation of the ansible playbooks
```

```

# provided with elasticcluster. 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_worker']

setup_provider = elasticcluster.AnsibleSetupProvider(groups)

# cluster initialisation (note: ssh keys are same for all nodes)
# After the steps above initialising an empty cluster is a piece 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 = elasticcluster.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
#   separate thread.
# * to make sure elasticcluster 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 `elasticcluster` will store the cluster in memory only. Therefore after a programm shutdown the cluster will not be available anymore in `elasticcluster`, 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
# separate file in the provided storage directory.
repository = elasticcluster.PickleRepository('/path/to/storage/dir')

# On cluster initialisation we can pass the repository as optional
# argument.
cluster = elasticcluster.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 save 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 = elasticcluster.PickleRepository('/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

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

```
import logging

import elasticcluster

log = elasticcluster.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.

3.6.2 elasticcluster.cluster

3.6.3 elasticcluster.conf

3.6.4 elasticcluster.exceptions

3.6.5 elasticcluster.gc3pie_config

3.6.6 elasticcluster.__main__

3.6.7 elasticcluster.providers

3.6.8 elasticcluster.providers.ansible_provider

3.6.9 elasticcluster.providers.ec2_boto

3.6.10 elasticcluster.providers.gce

3.6.11 elasticcluster.providers.openstack

3.6.12 elasticcluster.repository

3.6.13 elasticcluster.subcommands

3.6.14 elasticcluster.utils

CHAPTER 4

Indices and tables

- `genindex`
- `modindex`
- `search`