ElastiCluster Manual

Release 1.3.dev

Services and Support for Science IT, University of Zurich

Sep 21, 2017

Contents

CHAPTER 1

Introduction

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 demoes 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 elasticluster@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.

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.

CHAPTER 2

Overview

The architecture of elasticluster is quite simple: the configuration file `~/.elasticluster/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 cluster[1], 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.

CHAPTER 3

Table of Contents

Installation

ElastiCluster is a Python program; Python version 2.6 or 2.7 is required to run it.

The easiest way to install elasticluster is by using pip, this will install the latest *stable* code release from the PyPI package repository. Section *Installing released code from PyPI* will explain you how to do it.

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

Prepare the environment for installation

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 *elasticluster* 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 *elasticluster*.

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 elasticluster
. elasticluster/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'
```

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 `elasticluster`.

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

```
pip install elasticluster
```

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

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 elasticluster 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 `elasticluster`.

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

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

Now the `elasticluster` 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.

Configuration

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

If directory *~/.elasticluster/config.d* exists (or, if you run `elasticluster` 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:

```
elasticluster list-templates
```

If no configuration file is found, it will copy an example configuration file in `~/.elasticluster/config`. The example is fully commented and self-documenting.

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

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

You will have to edit the configuration file in `~/.elasticluster/config` and update it with the correct values.

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

Basic syntax of the configuration file

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

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

```
key=value
```

Section names 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>` override configuration for specific group 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.

Processing of configuration values

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

- References to enviromental variables of the form `$VARNAME` or `${VARNAME}` 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 Linux user name on the computer where ElastiCluster is running:

  ```
  [cloud/openstack]
  username = $USER
  # ...
  ```

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

this string ...	... expands to:
`${elasticluster_playbooks}`	Path to the root directory containing the Ansible playbooks distributed with ElastiCluster
`${ansible_pb_dir}`	Deprecated alias for `${elasticluster_playbooks}`

- 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/`)

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:

- `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: `ec2_boto`, `openstack`, `google` or `libcloud`.

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

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 *elasticluster* 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

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

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

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" instanciation parameters, both in name and in type.

For example, to configure an Azure connection, go to the page https://libcloud.readthedocs.io/en/latest/compute/drivers/azure.html and check what the *Instantiating a driver* section states: you would need to configure the keys `subscription_id` and `key_file`.

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

Provider	Additional arguments	Example
Azure	key_file, subscription_id	`subscription_id=...` `key_file=/path/to/azure.pem`
CloudSigma	username, password, region, api_version	`username=user password=pass region=zrh api_version=2.0`
CloudStack	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 *elasticluster* 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/elasticluster/tree/master/examples

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 keypair does not exist it will be created by ElastiCluster, uploading the public SSH key pointed to by *user_key_public* (see below).

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.

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=elasticluster
# 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=elasticluster
# 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/elasticluster/tree/master/examples

Setup Section

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

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

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

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 `elasticluster 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 *prepended* 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/elasticluster/tree/master/examples

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 `elasticluster 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` Which class of nodes to SSH into, when running `elasticluster ssh` or `elasticluster sftp`.

Commands `elasticluster ssh` and `elasticluster 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.

`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` 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: OpenStack uses UUIDs (e.g. *2bf3baba-35c8-4e20-9cc9-b36808720c9b*), 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.

`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.

`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` Security group to use when starting the instance.

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

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.

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 of `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 a 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 requires 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/elasticluster/tree/master/examples

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 `~/.elasticluster/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/elasticluster/
```

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 fromat, use option `storage_type`:

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

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

Usage

The syntax of the elasticluster command is:

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

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

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

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

`-s PATH, --storage PATH`

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

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

`-c PATH, --config PATH`

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

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

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

stop Stop a cluster and all associated VM instances.

list List all clusters that are currently running.

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

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

setup Run Ansible to configure the cluster.

resize Resize a cluster by adding or removing nodes.

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

sftp Open an SFTP session to the cluster frontend host.

export Export a cluster as a ZIP file.

import Import a cluster from a ZIP file created with *elasticluster export*.

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

```
elasticluster -h
```

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

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

The `start` command

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

Basic usage of the command is:

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

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

```
elasticluster start slurm
```

The following options are available:

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

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

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

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

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

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

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

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

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

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

When you start a new cluster, elasticluster will:

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

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

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

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

To login on the frontend node, run the command:

    elasticluster ssh slurm

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

    elasticluster sftp slurm
```

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

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

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

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

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

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

Basic usage of the command is:

```
usage: elasticluster 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 elasticluster), this command will ignore these errors and will force termination of all the other instances.

`--yes`

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

The `list` command

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

```
$ elasticluster list

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

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

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

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

The `list-nodes` command

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

Basic usage of the command is:

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

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

The following options are available:

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

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

`-u, --update`

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

Example:

```
$ elasticluster list-nodes centossge

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

To login on the frontend node, run the command:

    elasticluster ssh centossge

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

    elasticluster sftp centossge

frontend nodes:

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

compute nodes:

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

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

The `list-templates` command

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

Basic usage of the command is:

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

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

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

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

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

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

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

The `setup` command

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

Basic usage of the command is:

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

  ```
  elasticluster setup mycluster -- --tags hdfs
  ```

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

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

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

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

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

```
elasticluster 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.

The `resize` command

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

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

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

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

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

Basic usage of the command is:

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

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

The following options are available:

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

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

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

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

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

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

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

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

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

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

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

`--no-setup`

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

`--yes`

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

The `ssh` command

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

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

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

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

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

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

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

Note that elasticluster will save in *~/.elasticluster/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=~/.elasticluster/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.

The `sftp` command

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

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

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

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

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

Note that elasticluster will save in *~/.elasticluster/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=~/.elasticluster/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.

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:

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

The following options are available:

`--overwrite`

Overwritep 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 elasticluster. By default elasticluster 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 rempote 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.

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:

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

where `file` is the zip file produced by *elasticluster 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, elasticluster will also update the corresponding attributes of both cluster and nodes.

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

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

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.

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.

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 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 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'
```

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:///.../elasticluster/src
  Running setup.py egg_info for package from file:///.../elasticluster/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 (/.../elasticluster/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 (/.../elasticluster/lib/python2.7/site-packages))

----------------------------------------
Cleaning up...
Command python setup.py egg_info failed with error code 2 in /.../elasticluster/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 ElastiCluster 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.

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

When trying to install ElastiCluster 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 ElastiCluster 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 ElastiCluster (e.g. CentOS' default 0.9.8);
2. The `setuptools` Python package was updated by running `easy_install -U setuptools`.

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.

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.)

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.)

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.)

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.)

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/
↪elasticluster/elasticluster/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'")]
```

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

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

Setup of RHEL/CentOS 7 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: [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.

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 tha base image;
- make a snapshot;
- use that snmapshot as the base for ElastiCluster.

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

Issues when installing from source 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.

When installing ElastiCluster 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>
    'elasticluster = elasticluster.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/elasticluster/build/setuptools/setuptools/command/install.py",␣
↪line 65, in run
  File "/Users/michela/elasticluster/build/setuptools/setuptools/command/install.py",␣
↪line 115, in do_egg_install
  File "/Users/michela/elasticluster/build/setuptools/setuptools/command/easy_install.
↪py", line 360, in run

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

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

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

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

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

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

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

  File "/Users/michela/elasticluster/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
```

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 *elasticluster*:

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

In this case, the issue is caused by *google-api-python-client*, and you should:

1. uninstall it using *pip uninstall*

2. reinstall it using *pip install*
3. re-run elasticluster installation:

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

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 bundled[1] with ElastiCluster and how to use them.

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>
```

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
# ...
```

[1] The playbooks can be found in the `elasticluster/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.

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
`slurm_master`	SLURM controller/scheduler node; also runs the accounting storage daemon *slurmdbd* and its MySQL/MariaDB backend.
`slurm_worker`	SLURM execution node: runs the *slurmd* daemon.
`slurm_submit`	SLURM client: has all the submission and query commands installed, but runs no daemon.

The following example configuration sets up a SLURM batch-queuing 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 the Ganglia one; in this case the `setup` stanza will look like:

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

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

Variable name	Default	Description
`slurm_selecttype`	`select/cons_res`	Value of *SelectType* in *slurm.conf*
`slurm_selecttypeparameters`	`CR_Core_Memory`	Value of *SelectTypeParameters* in *slurm.conf*
`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)
`multiuser_cluster`	yes	Install NIS/YP

Note that the `slurm_*` extra variables need to be set *globally* (e.g., `global_var_slurm_selectype`) 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
`gridengine_master`	Scheduler, admin, and submission host
`gridengine_worker`	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).

HTCondor

Tested on:

- Ubuntu 12.04

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

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

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

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

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

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

Ganglia

Tested on:

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

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

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

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

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

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

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.

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
`hadoop_master`	Install the Hadoop cluster master node: run YARN "ResourceManager" and Hive "MetaStore" server. In addition, install a PostgreSQL server to host Hive metastore tables.
`hadoop_worker`	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).

CephFS

Supported on:

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

Ansible group	Action
`ceph_mon`	Install Ceph server software and configure this host to run the MON (monitor) service. There *must* be at least one MON node in any Ceph cluster.
`ceph_osd`	Install Ceph server software and configure this host to run the OSD (object storage device) service. There *must* be at least three OSD nodes in a Ceph cluster.
`ceph_mds`	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.
`ceph_client`	Install required packages to make usage of Ceph Storage Cluster and CephFS possible. Any mount point listed in `CEPH_MOUNTS` 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 3.1: Ceph/CephFS variables in ElastiCluster

Variable	Default value	Description
`ceph_release`	`luminous`	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.
`ceph_osd_pool_size`	2	Default number of object replicas in a pool.
`ceph_osd_pg_num`	computed according to: http://docs.ceph.com/docs/master/rados/operations/placement-groups/#a-preselection-of-pg-num	Default number of PGs in a pool.
`ceph_metadata_pg_num`	1/8 of `ceph_osd_pool_size`	Number of PGs for the CephFS metadata pool.
`ceph_data_pg_num`	7/8 of `ceph_osd_pool_size`	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 tu 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
- RHEL/CentOS 6.x, 7.x

ansible groups	action
`glusterfs_server`	Run a GlusterFS server with a single *brick*
`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 volume is neither 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

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

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

```
[cluster/slurm+orangefs]
frontend_nodes=1
compute_nodes=10
orangefs_nodes=10
ssh_to=frontend
setup_provider=ansible_slurm+orangefs
...

[setup/ansible_slurm+orangefs]
frontend_groups=slurm_master,pvfs2_client
compute_groups=slurm_worker,pvfs2_client
orangefs_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.

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.

Elasticluster programming API

elasticluster

Overview

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

Getting Started

The following subchapters introduce the basic concepts of Elasticluster.

Cluster

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

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

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

Node

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

See the `Node` api docs for further details.

Cloud Provider

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

- **`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 elasticluster

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

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

setup_provider = elasticluster.AnsibleSetupProvider(groups)

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

# To add nodes to the cluster we can use the add_node. This
# only initialises a new node, but does not start it yet.
# The add node function is basically a factory method to make it easy to
# add nodes to a cluster. It takes the following arguments:
#   kind:   kind of the node in this cluster. This corresponds to the
#           groups defined in the cloud_provider.
```

```
cluster.add_node('frontend', 'ami-00000048', 'gc3-user',
                 'm1.tiny', 'all_tcp_ports')

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

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

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

Asynchronous node start

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

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

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

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

Storing a cluster on disk

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

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

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

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

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

```
repository = elasticluster.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

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

```
import logging

import elasticluster

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

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

elasticluster.cluster

elasticluster.conf

elasticluster.exceptions

elasticluster.gc3pie_config

elasticluster.__main__

elasticluster.providers

elasticluster.providers.ansible_provider

elasticluster.providers.ec2_boto

elasticluster.providers.gce

elasticluster.providers.openstack

elasticluster.repository

elasticluster.subcommands

elasticluster.utils

CHAPTER 4

Indices and tables

- genindex
- modindex
- search