EC3 Documentation

Release 1.0

EC3

Jun 14, 2023

Contents

Contents:

Introduction

Elastic Cloud Computing Cluster (EC3) is a tool to create elastic virtual clusters on top of Infrastructure as a Service (IaaS) providers, either public (such as Amazon Web Services, Google Cloud or Microsoft Azure) or on-premises (such as OpenNebula and OpenStack). We offer recipes to deploy TORQUE (optionally with MAUI), SLURM, SGE, HTCondor, Mesos, Nomad and Kubernetes clusters that can be self-managed with CLUES: it starts with a single-node cluster and working nodes will be dynamically deployed and provisioned to fit increasing load (number of jobs at the LRMS). Working nodes will be undeployed when they are idle. This introduces a cost-efficient approach for Cluster-based computing.

1.1 Installation

1.1.1 Requisites

The program *ec3* requires Python 2.6+, PLY, PyYAML, Requests, jsonschema and an IM server, which is used to launch the virtual machines.

PyYAML is usually available in distribution repositories (`python-yaml` in Debian; `PyYAML` in Red Hat; and `PyYAML` in pip).

PLY is usually available in distribution repositories (`python-ply` and `ply` in pip).

Requests is usually available in distribution repositories (`python-requests` and `requests` in pip).

jsonschema is usually available in distribution repositories (`python-jsonschema` and `jsonschema` in pip).

By default *ec3* uses our public IM server in *appsgrycap.i3m.upv.es*. *Optionally* you can deploy a local IM server following the instructions of the **'IM manual'_**.

Also `sshpass` command is required to provide the user with ssh access to the cluster.

1.1.2 Installing

As Python 2 is no longer supported, we recommend to install ec3 with Python 3.

First you need to install pip tool. To install them in Debian and Ubuntu based distributions, do:

```
sudo apt update
sudo apt install -y python3-pip
```

In Red Hat based distributions (RHEL, CentOS, Amazon Linux, Oracle Linux, Fedora, etc.), do:

```
sudo yum install -y epel-release
sudo yum install -y which python3-pip
```

Then you only have to call the install command of the pip tool with the *ec3-cli* package:

```
sudo pip3 install ec3-cli
```

You can also download the last *ec3* version from this git repository:

```
git clone https://github.com/grycap/ec3
```

Then you can install it calling the pip tool with the current ec3 directory:

```
sudo pip3 install ./ec3
```

1.2 Basic example with Amazon EC2

First create a file `auth.txt` with a single line like this:

```
id = provider ; type = EC2 ; username = <<Access Key ID>> ; password = <<Secret␣
→Access Key>>
```

Replace `<<Access Key ID>>` and `<<Secret Access Key>>` with the corresponding values for the AWS account where the cluster will be deployed. It is safer to use the credentials of an IAM user created within your AWS account.

This file is the authorization file (see Authorization file), and can have more than one set of credentials.

Now we are going to deploy a cluster in Amazon EC2 with a limit number of nodes = 10. The parameter to indicate the maximum size of the cluster is called `ec3_max_instances` and it has to be indicated in the RADL file that describes the infrastructure to deploy. In our case, we are going to use the ubuntu-ec2 recipe, available in our github repo. The next command deploys a TORQUE cluster based on an Ubuntu image:

```
$ ec3 launch mycluster torque ubuntu-ec2 -a auth.txt -y
WARNING: you are not using a secure connection and this can compromise the secrecy of␣
→the passwords and private keys available in the authorization file.
Creating infrastructure
Infrastructure successfully created with ID: 60
  ¨         Front-end state: running, IP: 132.43.105.28
```

If you deployed a local IM server, use the next command instead:

```
$ ec3 launch mycluster torque ubuntu-ec2 -a auth.txt -u http://localhost:8899
```

This can take several minutes. After that, open a ssh session to the front-end:

```
$ ec3 ssh mycluster
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-24-generic x86_64)
```

(continues on next page)

```
 * Documentation:  https://help.ubuntu.com/

ubuntu@torqueserver:~$
```

Also you can show basic information about the deployed clusters by executing:

```
$ ec3 list
   name        state         IP          nodes
-------------------------------------------
 mycluster   configured  132.43.105.28    0
```

1.3 EC3 in Docker Hub

EC3 has an official Docker container image available in Docker Hub and GitHub Container Regitry that can be used instead of installing the CLI. You can download it by typing:

```
$ sudo docker pull grycap/ec3
or
$ sudo docker pull ghcr.io/grycap/ec3
```

You can exploit all the potential of EC3 as if you download the CLI and run it on your computer:

```
$ sudo docker run grycap/ec3 list
$ sudo docker run grycap/ec3 templates
```

To launch a cluster, you can use the recipes that you have locally by mounting the folder as a volume. Also it is recommendable to mantain the data of active clusters locally, by mounting a volume as follows:

```
$ sudo docker run -v /home/user/:/tmp/ -v /home/user/ec3/templates/:/etc/ec3/
→templates -v /home/user/.ec3/clusters:/root/.ec3/clusters grycap/ec3 launch␣
→mycluster torque ubuntu16 -a /tmp/auth.dat
```

Notice that you need to change the local paths to the paths where you store the auth file, the templates folder and the .ec3/clusters folder. So, once the front-end is deployed and configured you can connect to it by using:

```
$ sudo docker run -ti -v /home/user/.ec3/clusters:/root/.ec3/clusters grycap/ec3 ssh␣
→mycluster
```

Later on, when you need to destroy the cluster, you can type:

```
$ sudo docker run -ti -v /home/user/.ec3/clusters:/root/.ec3/clusters grycap/ec3␣
→destroy mycluster
```

1.4 Additional information

- EC3 Command-line Interface.

- Templates.

- Information about available templates: `ec3 templates [--search <topic>] [--full-description]`.

Architecture

2.1 Overview

EC3 proposes the combination of Green computing, Cloud computing and HPC techniques to create a tool that deploys elastic virtual clusters on top of IaaS Clouds. EC3 creates elastic cluster-like infrastructures that automatically scale out to a larger number of nodes on demand up to a maximum size specified by the user. Whenever idle resources are detected, the cluster dynamically and automatically scales in, according to some predefined policies, in order to cut down the costs in the case of using a public Cloud provider. This creates the illusion of a real cluster without requiring an investment beyond the actual usage. Therefore, this approach aims at delivering cost-effective elastic Cluster as a Service on top of an IaaS Cloud.

2.2 General Architecture

Fig. 1 summarizes the main architecture of EC3. The deployment of the virtual elastic cluster consists of two phases. The first one involves starting a VM in the Cloud to act as the cluster front-end while the second one involves the automatic management of the cluster size, depending on the workload and the specified policies. For the first step, a launcher (EC3 Launcher) has been developed that deploys the front-end on the Cloud using the infrastructure deployment services described in Section 3.1. The sysadmin will run this tool, providing it with the following information:

- Maximum cluster size. This serves to establish a cost limit in case of a workload peak. The maximum cluster size can be modified at any time once the virtual cluster is operating. Thus, the sysadmins can adapt the maximum cluster size to the dynamic requirements of their users. In this case the LRMS must be reconfigured to add the new set of virtual nodes and in some cases it may imply a LRMS service restart.

- *RADL* document specifying the desired features of the cluster front-end, regarding both hardware and software (OS, LRMS, additional libraries, etc.). These requirements are taken by the launcher and extended to include additional ones (such as installing CLUES and its requirements together with the libraries employed to interact with the IaaS Cloud provider, etc.) in order to manage elasticity.

The launcher starts an IM that becomes responsible of deploying the cluster front-end. This is done by means of the following steps:
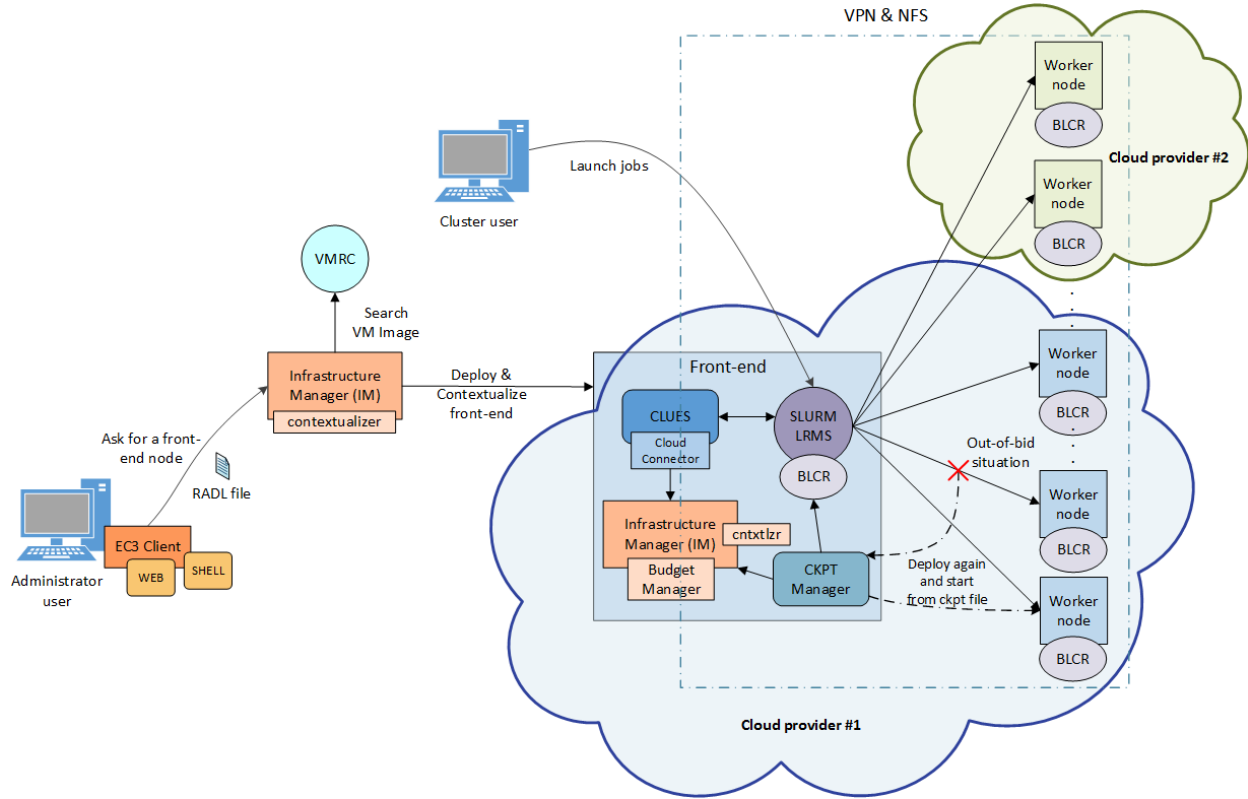
Fig. 1: Fig 1. EC3 Architecture.

1. Selecting the VMI for the front-end. The IM can take a particular user-specified VMI, or it can contact the VMRC to choose the most appropriate VMI available, considering the requirements specified in the RADL.

2. Choosing the Cloud deployment according to the specification of the user (if there are different providers).

3. Submitting an instance of the corresponding VMI and, once it is available, installing and configuring all the required software that is not already preinstalled in the VM

One of the main LRMS configuration steps is to set up the names of the cluster nodes. This is done using a sysadmin-specified name pattern (e.g. vnode-*) so that the LRMS considers a set of nodes such as vnode-1, vnode-2, ... , vnode-n, where n is the maximum cluster size. This procedure results in a fully operational elastic cluster. *Fig. 2* represents the sequence diagram and the interaction of the main components and actors during the deployment of the frontend of the cluster using EC3.

Once the front-end and the elasticity manager (CLUES) have been deployed, the virtual cluster becomes totally autonomous and every user will be able to submit jobs to the LRMS, either from the cluster front-end or from an external node that provides job submission capabilities. The user will have the perception of a cluster with the number of nodes specified as maximum size. CLUES will monitor the working nodes and intercept the job submissions before they arrive to the LRMS, enabling the system to dynamically manage the cluster size transparently to the LRMS and the user, scaling in and out on demand.

Just like in the deployment of the front-end, CLUES internally uses an IM to submit the VMs that will be used as working nodes for the cluster. For that, it uses a RADL document defined by the sysadmin, where the features of the working nodes are specified. Once these nodes are available, they are automatically integrated in the cluster as new available nodes for the LRMS. Thus, the process to deploy the working nodes is similar to the one employed to deploy the front-end.

Fig. 3 represents the sequence diagram and the interaction when a new job arrives to the LRMS and no nodes are
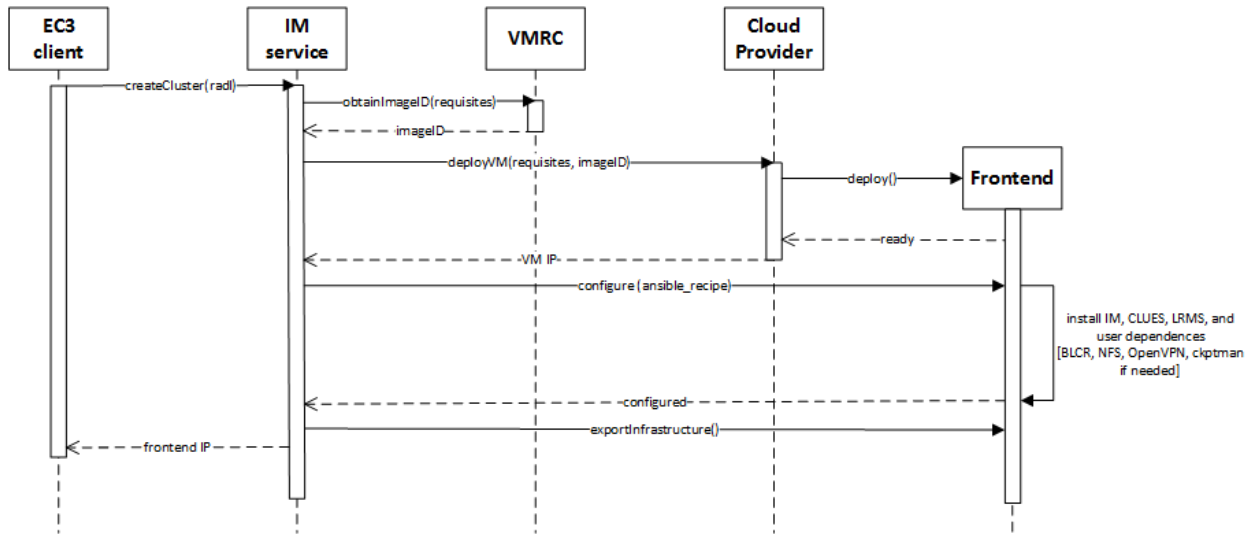
Fig. 2: Fig 2. Sequence diagram for the deployment of the frontend.

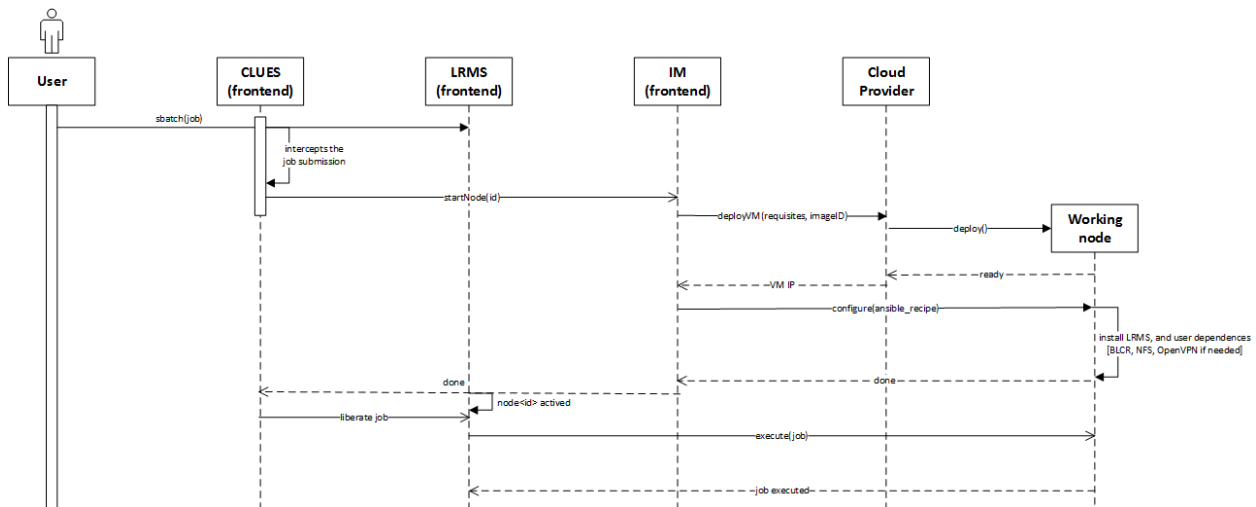available for the execution of the job.

Fig. 3: Fig 3. Sequence diagram that represents when a new job arrives to the cluster.

Note that the EC3-L tool can be executed on any machine that has a connection with the Cloud system and it is only employed to bootstrap the cluster. Once deployed, the cluster becomes autonomous and self-managed, and the machine from which the EC3-L tool was used (the dashed rectangle in Fig. 1) is no longer required. The expansion of the cluster while it is operating is carried out by the front-end node, by means of CLUES, as explained above.

2.3 Infrastructure Manager

The Infrastructure Manager (IM) is a tool that eases the access and the usability of IaaS clouds by automating the VMI selection, deployment, configuration, software installation, monitoring and update of Virtual Appliances. It supports APIs from a large number of virtual platforms, making user applications cloud-agnostic. In addition it integrates a contextualization system to enable the installation and configuration of all the user required applications providing the user with a fully functional infrastructure.

2.4 RADL

The main purpose of the Resource and Application description Language (RADL) is to specify the requirements of the resources where the scientific applications will be executed. It must address not only hardware (CPU number, CPU architecture, RAM size, etc.) but also software requirements (applications, libraries, data base systems, etc.). It should include all the configuration details needed to get a fully functional and configured VM (a Virtual Appliance or VA). It merges the definitions of specifications, such as OVF, but using a declarative scheme, with contextualization languages such as Ansible. It also allows describing the underlying network capabilities required.

2.5 CLUES

CLUES is an energy management system for High Performance Computing (HPC) Clusters and Cloud infrastructures. The main function of the system is to power off internal cluster nodes when they are not being used, and conversely to power them on when they are needed. CLUES system integrates with the cluster management middleware, such as a batch-queuing system or a cloud infrastructure management system, by means of different connectors.

Deployment Models

EC3 supports a wide variety of deployment models (i.e. cluster behaviour). In this section, we provide information about all of them and a example of configuration for each deployment model. For more details, you can follow reading ec3_variables, which provides more information regarding EC3 special variables that support the specification of the deployment model in the templates.

3.1 Basic structure (homogeneous cluster)

An homogeneous cluster is composed by working nodes that have the same characteristics (hardware and software). This is the basic deployment model of EC3, where we only have one type of `system` for the working nodes.

In EC3, a template specifying this model would be, for instance:

```
system wn (
   ec3_max_instances = 6 and
   ec3_node_type = 'wn' and
   cpu.count = 4 and
   memory.size >= 2048M and
   disk.0.os.name = 'linux' and
   net_interface.0.connection = 'net'
)
```

This RADL defines a *system* with the feature `cpu.count` equal to four, the feature `memory.size` greater or equal than `2048M`, a operative system based on `linux` and with the feature `net_interface.0.connection` bounded to `'net'`. It also fixes the maximum number of working nodes to 6 with the EC3 special variable `ec3_max_instances`, and indicates that this *system* is of type `wn` though `ec3_node_type`.

3.2 Heterogeneous cluster

This model allows that the working nodes comprising the cluster can be of different characteristics (hardware and software). This is of special interest when you need nodes with different configuration or hardware specifications but
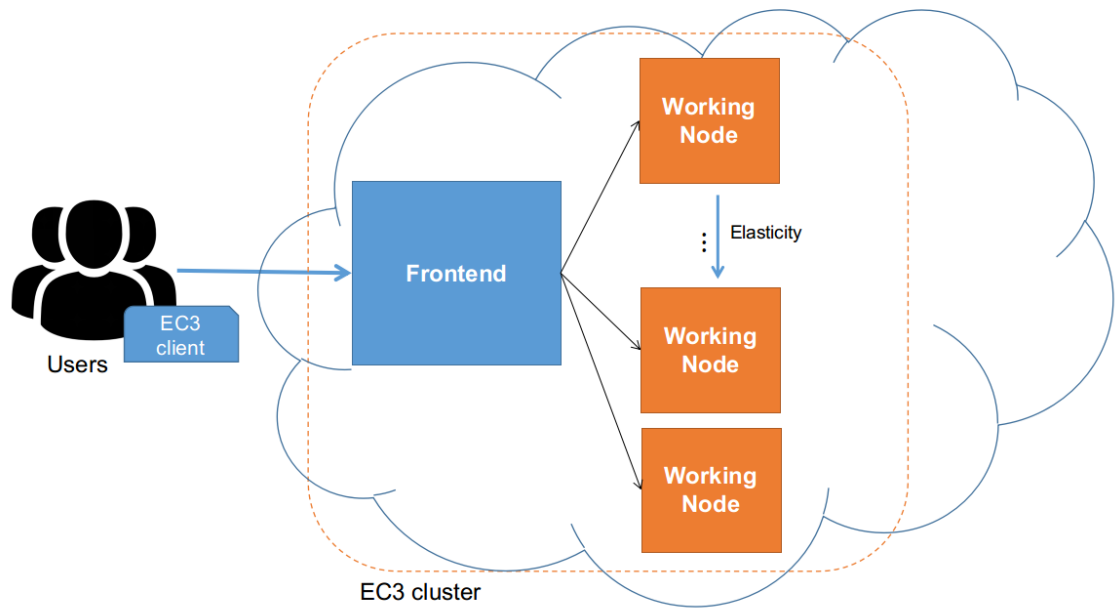
Fig. 1: Fig 1. EC3 Deployment Model for an homogeneous cluster.

all working together in the same cluster. It also allows you to configure several queues and specify from which queue the working node belongs to.

In EC3, a template specifying this model would be, for instance:

```
system wn (
   ec3_max_instances = 6 and
   ec3_node_type = 'wn' and
   ec3_node_queues_list = 'smalljobs' and
   ec3_node_pattern = 'wn[1,2,3]' and
   cpu.count = 4 and
   memory.size >= 2048M and
   disk.0.os.name = 'linux' and
   net_interface.0.connection = 'net'
)

system largewn (
   ec3_inherit_from = system wn and
   ec3_node_queues_list = 'largejobs' and
   ec3_node_pattern = 'wn[4,5,6]' and
   cpu.count = 8 and
   memory.size >= 4096M
)
```

This RADL defines two different *system*. The first one defines the `wn` with the feature `cpu.count` equal to four, the feature `memory.size` greater or equal than `2048M`, and with the feature `net_interface.0.connection` bounded to `'net'`. Again, it also fixes the maximum number of working nodes to `6` with the EC3 special variable `ec3_max_instances`, and indicates that this *system* is of type `wn` though `ec3_node_type`. More systems can be defined, it is not limited to two types of working nodes, it's only an example. The second defined *system*, called `largewn`, inherits the already defined characteristics of `system wn`, by using the EC3 special feature
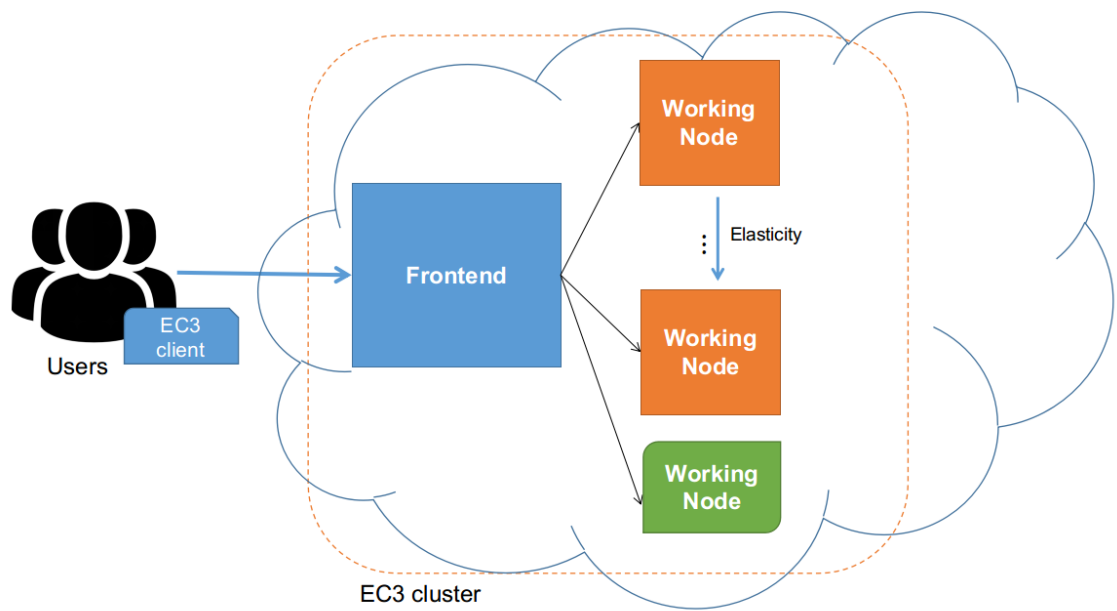
Fig. 2: Fig 2. EC3 Deployment Model for an heterogeneous cluster.

`ec3_inherit_from`, but it changes the values for `cpu.count` and `memory.size`. Regarding queue management, the RADL defines two queues by using `ec3_node_queues_list`, and determines whose nodes belong to them. It is also defined the pattern to construct the name of the nodes by the `ec3_node_pattern` variable.

3.3 Cloud Bursting (Hybrid clusters)

The third model supported by EC3 is Cloud Bursting. It consists on launching nodes in two or more different Cloud providers. This is done to manage user quotas or saturated resources. When a limit is reached and no more nodes can be deployed inside the first Cloud Provider, EC3 will launch new nodes in the second defined Cloud provider. This is also called a hybrid cluster. The nodes deployed in different Cloud providers can be different also, so heterogeneous clusters with cloud bursting capabilities can be deployed and automatically managed with EC3. The nodes would be automatically interconnected by using VPN or SSH tunneling techniques.

In EC3, a template specifying this model would be, for instance:

```
system wn (
    disk.0.os.name = 'linux' and
    disk.0.image.url = 'one://mymachine.es/1' and
    disk.0.os.credentials.username = 'ubuntu' and
    ec3_max_instances = 6 and              # maximum instances of this kind
    cpu.count = 4 and
    memory.size >= 2048M and
    ec3_if_fail = 'wn_aws'
)


system wn_aws (
    ec3_inherit_from = system wn and       # Copy features from system 'wn'
    disk.0.image.url = 'aws://us-east-1/ami-30519058' and # Ubuntu 14.04
```
(continues on next page)
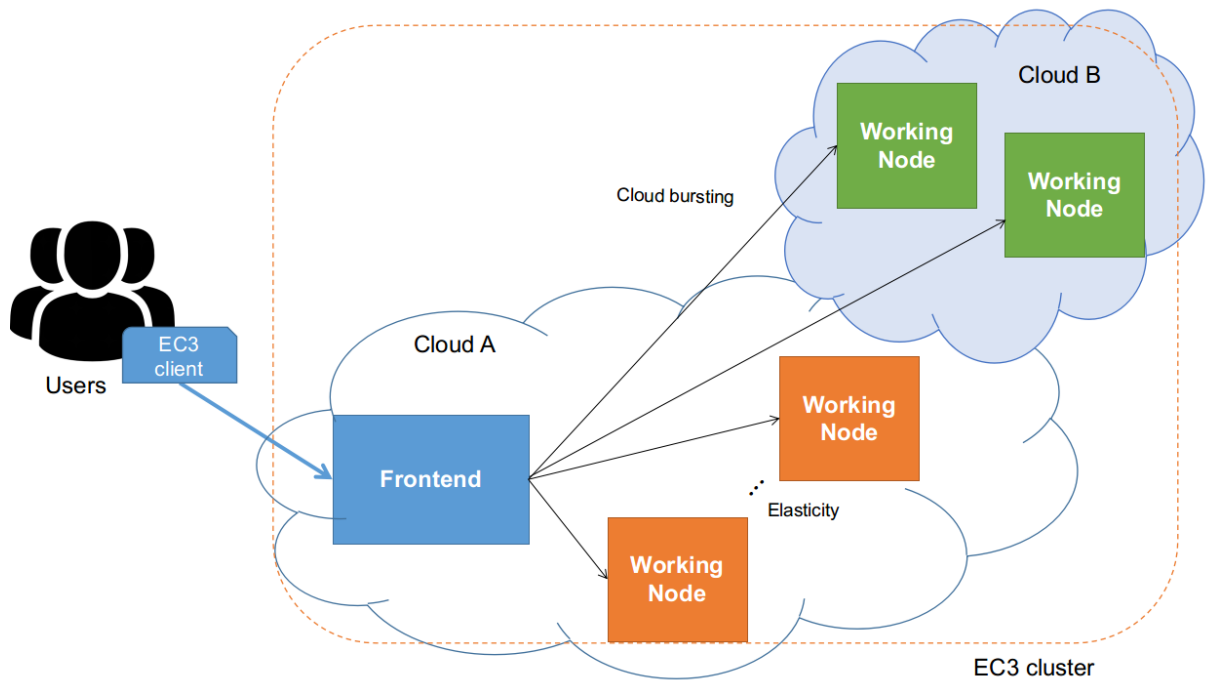
Fig. 3: Fig 3. EC3 Deployment Model for an hybrid cluster.

```
    disk.0.os.credentials.username = 'ubuntu' and
    ec3_max_instances = 8 and                   # maximum instances of this kind
    ec3_if_fail = ''
)
```

This RADL is similar to the upper ones. It also defines two different *system*, but the important detail here is the EC3 variable `ec3_if_fail`. It defines the next *system* type to be used when no more instances of *system wn* can be launched.

Command-line Interface

The program is called like this:

```
$ ec3 [-l <file>] [-ll <level>] [-q]␣
→launch|list|show|templates|ssh|reconfigure|destroy|clone|migrate|stop|restart|tranfer|update␣
→[args...]
```

-l <file>, **--log-file** <file>
> Path to file where logs are written. Default value is standard output error.

-ll <level>, **--log-level** <level>
> Only write in the log file messages with level more severe than the indicated: 1 for *debug*, 2 for *info*, 3 for *warning* and 4 for *error*.

-q, --quiet
> Don't show any message in console except the front-end IP.

4.1 Command `launch`

To deploy a cluster issue this command:

```
ec3 launch <clustername> <template_0> [<template_1> ...] [-a <file>] [-u <url>] [-y]
```

clustername
> Name of the new cluster.

template_0 ...
> Template names that will be used to deploy the cluster. ec3 tries to find files with these names and extension `.radl` in `~/.ec3/templates` and `/etc/ec3/templates`. Templates are RADL descriptions of the virtual machines (e.g., instance type, disk images, networks, etc.) and contextualization scripts. See *Command templates* to list all available templates.

--add
> Add a piece of RADL. This option is useful to set some features. The following example deploys a cluster with the Torque LRMS with up to four working nodes:
>
> > ./ec3 launch mycluster torque ubuntu-ec2 –add "system wn (ec3_max_instances = 4)"

-u <url>, **--restapi-url** <url>
> URL to the IM REST API service.

-a <file>, **--auth-file** <file>
> Path to the authorization file, see *Authorization file*. This option is compulsory.

--dry-run
> Validate options but do not launch the cluster.

-n, --not-store
> The new cluster will not be stored in the local database.

-p, --print
> Print final RADL description if the cluster after cluster being successfully configured.

--json
> If option -p indicated, print RADL in JSON format instead.

--on-error-destroy
> If the cluster deployment fails, try to destroy the infrastructure (and relinquish the resources).

-y, --yes
> Do not ask for confirmation when the connection to IM is not secure. Proceed anyway.

-g, --golden-images
> Generate a VMI from the first deployed node, to accelerate the contextualization process of next node deployments.

4.2 Command `reconfigure`

The command reconfigures a previously deployed clusters. It can be called after a failed deployment (resources provisioned will be maintained and a new attempt to configure them will take place). It can also be used to apply a new configuration to a running cluster:

```
ec3 reconfigure <clustername>
```

-a <file>, **--auth-file** <file>
> Append authorization entries in the provided file. See *Authorization file*.

--add
> Add a piece of RADL. This option is useful to include additional features to a running cluster. The following example updates the maximum number of working nodes to four:

```
./ec3 reconfigure mycluster --add "system wn ( ec3_max_instances = 4 )"
```

-r, --reload
> Reload templates used to launch the cluster and reconfigure it with them (useful if some templates were modified).

--template, -t
> Add a new template/recipe. This option is useful to add new templates to a running cluster. The following example adds the docker recipe to the configuration of the cluster (i.e. installs Docker):

```
./ec3 reconfigure mycluster -r -t docker
```

4.3 Command `ssh`

The command opens a SSH session to the infrastructure front-end:

```
ec3 ssh <clustername>
```

--show-only
> Print the command line to invoke SSH and exit.

4.4 Command `destroy`

The command undeploys the cluster and removes the associated information in the local database.:

```
ec3 destroy <clustername> [--force]
```

--force
> Removes local information of the cluster even when the cluster could not be undeployed successfully.

4.5 Command `show`

The command prints the RADL description of the cluster stored in the local database:

```
ec3 show <clustername> [-r] [--json]
```

-r, --refresh
> Get the current state of the cluster before printing the information.

--json
> Print RADL description in JSON format.

4.6 Command `list`

The command prints a table with information about the clusters that have been launched:

```
ec3 list [-r] [--json]
```

-r, --refresh
> Get the current state of the cluster before printing the information.

--json
> Print the information in JSON format.

4.7 Command `templates`

The command displays basic information about the available templates like *name*, *kind* and a *summary* description:

```
ec3 templates [-s/--search <pattern>] [-f/--full-description] [--json]
```

-s, --search
    Show only templates in which the `<pattern>` appears in the description.

-n, --name
    Show only the template with that name.

-f, --full-description
    Instead of the table, it shows all the information about the templates.

--json
    Print the information in JSON format.

If you want to see more information about templates and its kinds in EC3, visit Templates.

4.8 Command `clone`

The command clones an infrastructure front-end previously deployed from one provider to another:

```
ec3 clone <clustername> [-a/--auth-file <file>] [-u <url>] [-d/--destination
↪<provider>] [-e]
```

-a `<file>`, **--auth-file** `<file>`
    New authorization file to use to deploy the cloned cluster. See *Authorization file*.

-d `<provider>`, **--destination** `<provider>`
    Provider ID, it must match with the id provided in the auth file. See *Authorization file*.

-u `<url>`, **--restapi-url** `<url>`
    URL to the IM REST API service. If not indicated, EC3 uses the default value.

-e, --eliminate
    Indicate to destroy the original cluster at the end of the clone process. If not indicated, EC3 leaves running the original cluster.

4.9 Command `migrate`

The command migrates a previously deployed cluster and its running tasks from one provider to another. It is mandatory that the original cluster to migrate has been deployed with SLURM and BLCR, if not, the migration process can't be performed. Also, this operation only works with clusters which images are selected by the VMRC, it does not work if the URL of the VMI/AMI is explicitly written in the system RADL:

```
ec3 migrate <clustername> [-b/--bucket <bucket_name>] [-a/--auth-file <file>] [-u
↪<url>] [-d/--destination <provider>] [-e]
```

-b `<bucket_name>`, **--bucket** `<bucket_name>`
    Bucket name of an already created bucket in the S3 account displayed in the auth file.

-a `<file>`, **--auth-file** `<file>`
> New authorization file to use to deploy the cloned cluster. It is mandatory to have valid AWS credentials in this file to perform the migration operation, since it uses Amazon S3 to store checkpoint files from jobs running in the cluster. See *Authorization file*.

-d `<provider>`, **--destination** `<provider>`
> Provider ID, it must match with the id provided in the auth file. See *Authorization file*.

-u `<url>`, **--restapi-url** `<url>`
> URL to the IM REST API service. If not indicated, EC3 uses the default value.

-e, --eliminate
> Indicate to destroy the original cluster at the end of the migration process. If not indicated, EC3 leaves running the original cluster.

4.10 Command `stop`

To stop a cluster to later continue using it, issue this command:

```
ec3 stop <clustername> [-a <file>] [-u <url>] [-y]
```

clustername
> Name of the new cluster to stop.

-a `<file>`, **--auth-file** `<file>`
> Path to the authorization file, see *Authorization file*.

-u `<url>`, **--restapi-url** `<url>`
> URL to the IM REST API external service.

-y, --yes
> Do not ask for confirmation to stop the cluster. Proceed anyway.

4.11 Command `restart`

To restart an already stopped cluster, use this command:

```
ec3 restart <clustername> [-a <file>] [-u <url>]
```

clustername
> Name of the new cluster to restart.

-a `<file>`, **--auth-file** `<file>`
> Path to the authorization file, see *Authorization file*.

-u `<url>`, **--restapi-url** `<url>`
> URL to the IM REST API external service.

4.12 Command `transfer`

To transfers an already launched cluster that has not been transfered to the internal IM, use this command:

```
ec3 transfer <clustername> [-a <file>] [-u <url>]
```

clustername
> Name of the new cluster to transfer.

-a `<file>`, **--auth-file** `<file>`
> Path to the authorization file, see *Authorization file*.

-u `<url>`, **--restapi-url** `<url>`
> URL to the IM REST API external service.

4.13 Command `update`

The command updates a previously deployed clusters. It can be called to update the RADL of the WNs enabling to change some of their features (url or the image, cpu, memory . . .) that will be used in next "power on" operations on the cluster:

```
ec3 update <clustername>
```

-a `<file>`, **--auth-file** `<file>`
> Append authorization entries in the provided file. See *Authorization file*.

--add
> Add a piece of RADL. This option enables to include additional features to a running cluster. The following example updates the number of cpus of the WNs:

```
./ec3 update mycluster --add "system wn ( cpu.count = 2 )"
```

4.14 Configuration file

Default configuration values are read from `~/.ec3/config.yml`. If this file doesn't exist, it is generated with all the available options and their default values.

The file is formated in YAML. The options that are related to files admit the next values:

- an scalar: it will be treated as the content of the file, e.g.:

```
auth_file: |
    type = OpenNebula; host = myone.com:9999; username = user; password = 1234
    type = EC2; username = AKIAAAAAAAAAAAAAAAAA; password =␣
→aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

- a mapping with the key `filename`: it will be treated as the file path, e.g.:

```
auth_file:
    filename: /home/user/auth.txt
```

- a mapping with the key `stream`: it will select either standard output (`stdout`) or standard error (`stderr`), e.g.:

```
log_file:
    stream: stdout
```

4.15 Authorization file

The authorization file stores in plain text the credentials to access the cloud providers, the IM service and the VMRC service. Each line of the file is composed by pairs of key and value separated by semicolon, and refers to a single credential. The key and value should be separated by " = ", that is **an equals sign preceded and followed by one white space at least**, like this:

```
id = id_value ; type = value_of_type ; username = value_of_username ; password = 
↪value_of_password
```

Values can contain "=", and "\n" is replaced by carriage return. The available keys are:

- `type` indicates the service that refers the credential. The services supported are `InfrastructureManager`, `VMRC`, `OpenNebula`, `EC2`, `OpenStack`, `OCCI`, `LibCloud`, `Docker`, `GCE`, `Azure`, and `LibVirt`.

- `username` indicates the user name associated to the credential. In EC2 it refers to the *Access Key ID*. In Azure it refers to the user Subscription ID. In GCE it refers to *Service Account's Email Address*.

- `password` indicates the password associated to the credential. In EC2 it refers to the *Secret Access Key*. In GCE it refers to *Service Private Key*. See how to get it and how to extract the private key file from here info). In OpenStack sites using 3.x_oidc_access_token authentication it indicates the OIDC access token.

- `tenant` indicates the tenant associated to the credential. This field is only used in the OpenStack plugin.

- `host` indicates the address of the access point to the cloud provider. This field is not used in IM and EC2 credentials.

- `proxy` indicates the content of the proxy file associated to the credential. To refer to a file you must use the function "file(/tmp/proxyfile.pem)" as shown in the example. This field is only used in the OCCI plugin.

- `project` indicates the project name associated to the credential. This field is only used in the GCE plugin.

- `public_key` indicates the content of the public key file associated to the credential. To refer to a file you must use the function "file(cert.pem)" as shown in the example. This field is only used in the Azure plugin. See how to get it here

- `private_key` indicates the content of the private key file associated to the credential. To refer to a file you must use the function "file(key.pem)" as shown in the example. This field is only used in the Azure plugin. See how to get it here

- `id` associates an identifier to the credential. The identifier should be used as the label in the *deploy* section in the RADL.

- `token` indicates the OpenID token associated to the credential. This field is used in the OCCI and also to authenticate with the InfrastructureManager. To refer to the output of a command you must use the function "command(command)" as shown in the examples.

An example of the auth file:

```
id = one; type = OpenNebula; host = oneserver:2633; username = user; password = pass
id = ost; type = OpenStack; host = ostserver:5000; username = user; password = pass;
↪tenant = tenant
type = InfrastructureManager; username = user; password = pass
type = VMRC; host = http://server:8080/vmrc; username = user; password = pass
id = ec2; type = EC2; username = ACCESS_KEY; password = SECRET_KEY
id = gce; type = GCE; username = username.apps.googleusercontent.com; password = pass;
↪ project = projectname
id = docker; type = Docker; host = http://host:2375
id = occi; type = OCCI; proxy = file(/tmp/proxy.pem); host = https://fc-one.i3m.upv.
↪es:11443
```

(continues on next page)

```
id = azure; type = Azure; username = subscription-id; public_key = file(cert.pem);␣
↪private_key = file(key.pem)
id = kub; type = Kubernetes; host = http://server:8080; username = user; password =␣
↪pass
type = InfrastructureManager; token = command(oidc-token OIDC_ACCOUNT)
```

Notice that the user credentials that you specify are *only* employed to provision the resources (Virtual Machines, security groups, keypairs, etc.) on your behalf. No other resources will be accessed/deleted. However, if you are concerned about specifying your credentials to EC3, note that you can (and should) create an additional set of credentials, perhaps with limited privileges, so that EC3 can access the Cloud on your behalf. In particular, if you are using Amazon Web Services, we suggest you use the Identity and Access Management (IAM) service to create a user with a new set of credentials. This way, you can rest assured that these credentials can be cancelled at anytime.

4.16 Usage of Golden Images

Golden images are a mechanism to accelerate the contextualization process of working nodes in the cluster. They are created when the first node of the cluster is deployed and configured. It provides a preconfigured AMI specially created for the cluster, with no interaction with the user required. Each golden image has a unique id that relates it with the infrastructure. Golden images are also deleted when the cluster is destroyed.

There are two ways to indicate to EC3 the usage of this strategy:

- Command option in the CLI interface: as explained before, the `launch` command offers the option `-g`, `--golden-images` to indicate to EC3 the usage of golden images, e.g.:

```
./ec3 launch mycluster slurm  ubuntu -a auth.dat --golden-images
```

- In the RADL: as an advanced mode, the user can also specify the usage of golden images in the RADL file that describes the `system` architecture of the working nodes, e.g.:

```
system wn (
  cpu.arch = 'x86_64' and
  cpu.count >= 1 and
  memory.size >= 1024m and
  disk.0.os.name = 'linux' and
  disk.0.os.credentials.username = 'ubuntu' and
  disk.0.os.credentials.password = 'dsatrv' and
  ec3_golden_images = 'true'
)
```

Currently this feature is only available in the command-line interface for OpenNebula and Amazon Web Services providers. The list of supported providers will be uploaded soon.

Web Interface

5.1 Overview

EC3 as a Service (EC3aaS), is a web service offered to the community to facilitate the usage of EC3 to non-experienced users. Anyone can access the website and try the tool by using the user-friendly wizard to easily configure and deploy Virtual Elastic Clusters on multiple Clouds. The service does not require any account to use it. The user only needs to choose the Cloud provider and provide its credentials to allow EC3 to provision VMs from the underlying Clouds on behalf of the user.

5.2 Configuration and Deployment of a Cluster

In order to configure and deploy a Virtual Elastic Cluster using EC3aaS, a user accesses the homepage and selects "Deploy your cluster!" (*Fig. 1*). With this action, the web page will show different Cloud providers supported by the web interface version of EC3. Notice that not all the Cloud providers supported by EC3 appear in the website, only the most important providers in the field of research are currently supported by the web version. Users that want to use another supported Cloud provider, such as Microsoft Azure or Google Cloud Engine, are encouraged to use the CLI interface.

The first step, then, is to choose the Cloud provider where the cluster will be deployed (*Fig. 2*).

When the user chooses one of the offered providers (Amazon EC2, OpenNebula, OpenStack or EGI FedCloud), a wizard pops up (*Fig. 3*). This wizard will guide the user during the configuration process of the cluster, allowing to configure details like the operating system, the characteristics of the nodes, the maximum number of nodes of the cluster or the pre-installed software packages.

Specifically, the general wizard steps are:

1. **Provider account**: Valid user credentials are required to access to the resources of the Cloud provider chosen. Additionally, in the OpenNebula, OpenStack and EGI FedCloud wizards, the endpoint of the server is also required. The use of temporary credentials is recommended.

2. **Operating System**: the user can choose the OS of the cluster, by using a select box where the most common OS are available or by indicating a valid AMI/VMI identifier for the Cloud selected.
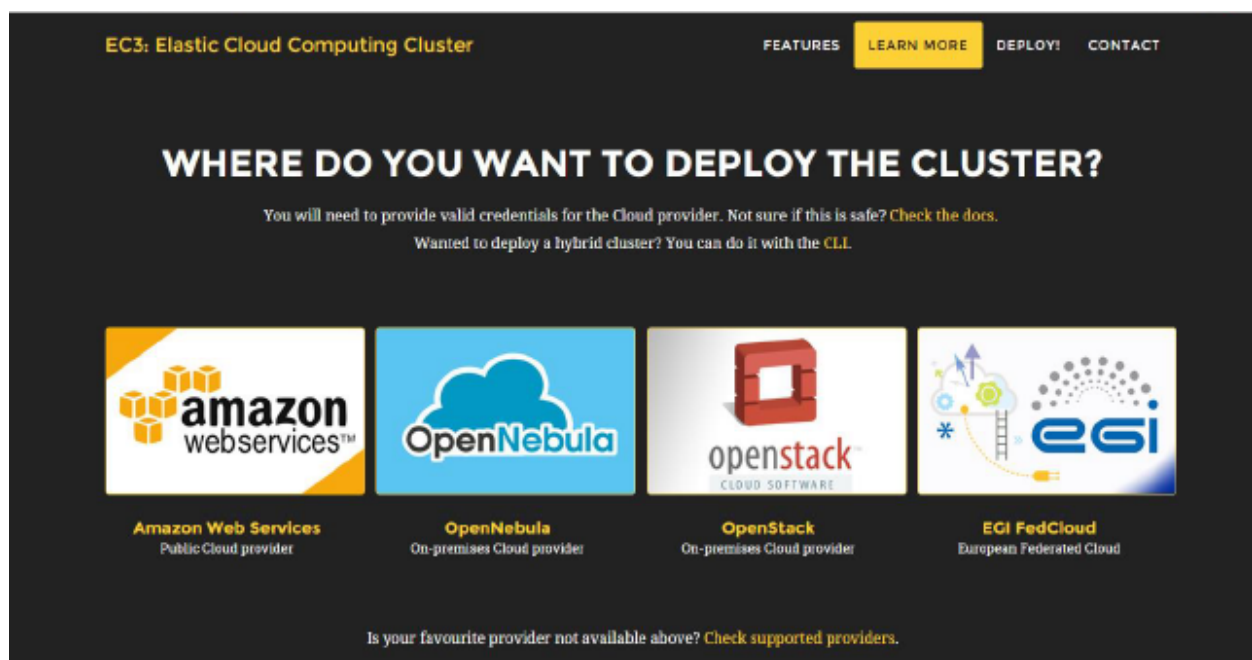
Fig. 1: Fig 1. EC3aaS homepage.

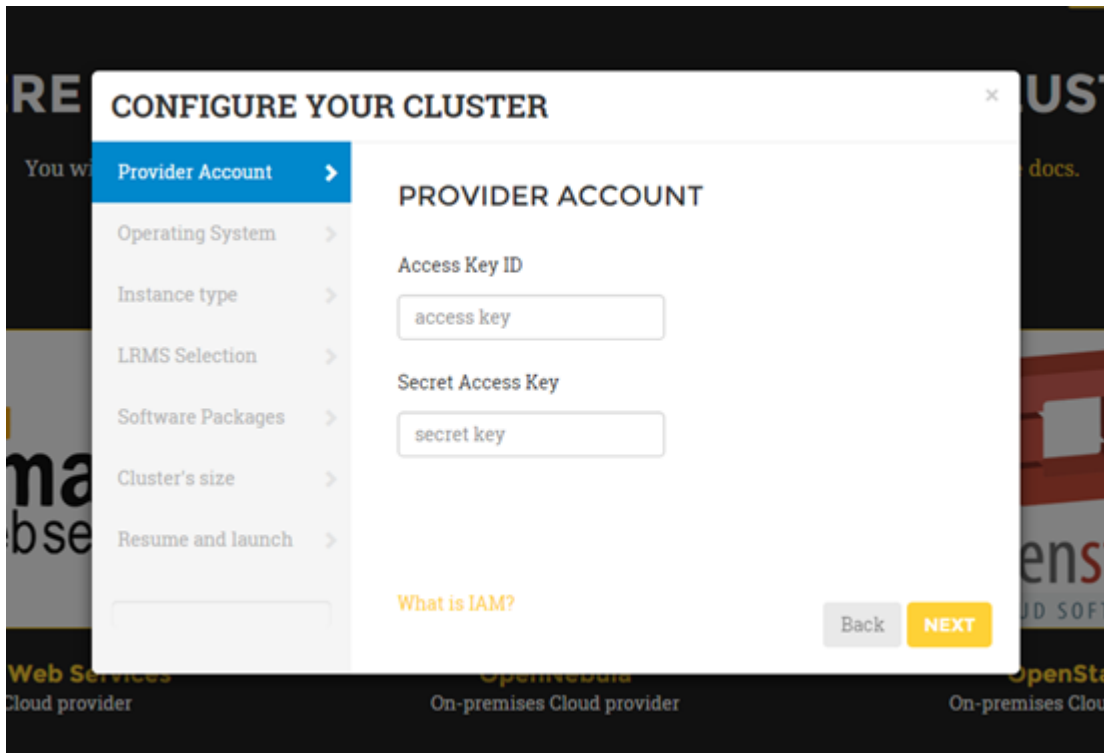Fig. 2: Fig 2. List of Cloud providers supported by EC3aaS.

Fig. 3: Fig 3. Wizard to configure and deploy a virtual cluster in Amazon EC2.

3. **Instance details**: the user must indicate the instance details, like the number of CPUs or the RAM memory, for the front-end and also the working nodes of the cluster. In case of using Amazon EC2, a select box is provided with the instance types offered by this provider. In case of using EGI FedCloud, the user must indicate the instance type desired from the ones available in the endpoint selected.

4. **LRMS Selection**: the user can choose the Local Resource Management System preferred to be automatically installed and configured by EC3. Currently, SLURM, Torque, Grid Engine, Mesos, Kubernetes and OSCAR (just in Opennebula and EGI Fedcloud) are supported.

5. **Software Packages**: a set of common software packages is available to be installed in the cluster, such as Docker Engine, Spark, Galaxy, OpenVPN, BLCR, GNUPlot, Tomcat or Octave. EC3 can install and configure them automatically in the contextualization process. If the user needs another software to be installed in his cluster, a new Ansible recipe can be developed and added to EC3 by using the CLI interface.

6. **Cluster's size**: the user can introduce the maximum number of nodes of the cluster, without including the front-end node. This value indicates the maximum number of working nodes that the cluster can scale. Remember that, initially the cluster only is created with the front-end, and the nodes are powered on on-demand.

7. **Resume and Launch**: a summary of the chosen configuration of the cluster is showed to the user at the last step of the wizard, and the deployment process can start by clicking the Submit button.

Notice that not all the Cloud providers need or allow to configure the same data. Because of that, the wizards are customized for each Cloud provider. For example, in the EGI FedCloud, a new wizard step appears, in order to collect the data about the MyProxy account of the user, a particularity of this Cloud provider. At the same time, the credentials needed to contact with the Cloud providers have different formats, depending on the provider selected. This is another reason why the wizards are customized for each one. Moreover, the instance options (CPU, RAM and disk) are presented different for each provider. Since Amazon EC2 and EGI FedCloud offer a predefined list of instance types, OpenNebula and OpenStack let the user indicate the values of CPU and RAM for the instance.

Finally, when all the steps of the wizard are filled correctly, the submit button starts the deployment process of the

cluster. Only the front-end will be deployed, because the working nodes will be automatically provisioned by EC3 when the workload of the cluster requires them. When the virtual machine of the front-end is running, EC3aaS provides the user with the necessary data to connect to the cluster (*Fig. 4*) which is composed by the username and password to connect to the cluster, the front-end IP and the name of the cluster. The user must keep this data during the lifetime of the cluster, since it is used also to terminate it. The cluster may not be configured when the IP of the front-end is returned by the web page, because the process of configuring the cluster is a batch process that takes several minutes, depending on the chosen configuration. However, the user is allowed to log in the front-end machine of the cluster since the moment it is deployed. To know if the cluster is configured, the command is cluster ready can be used. It will check if the configuration process of cluster has finished.
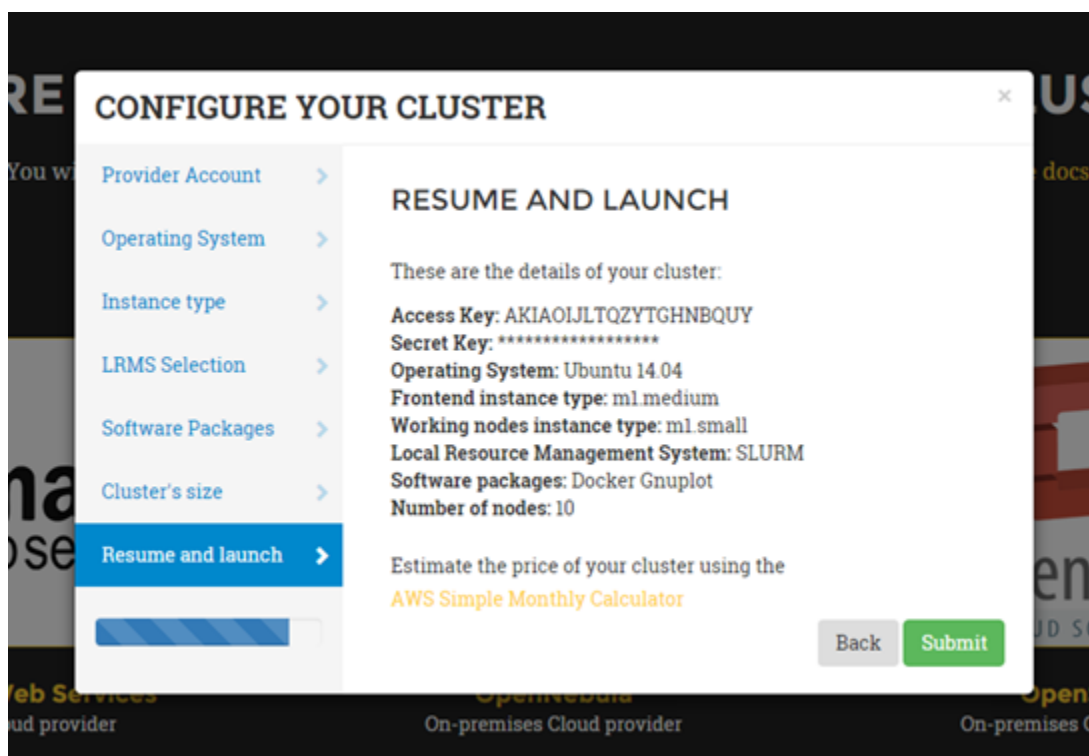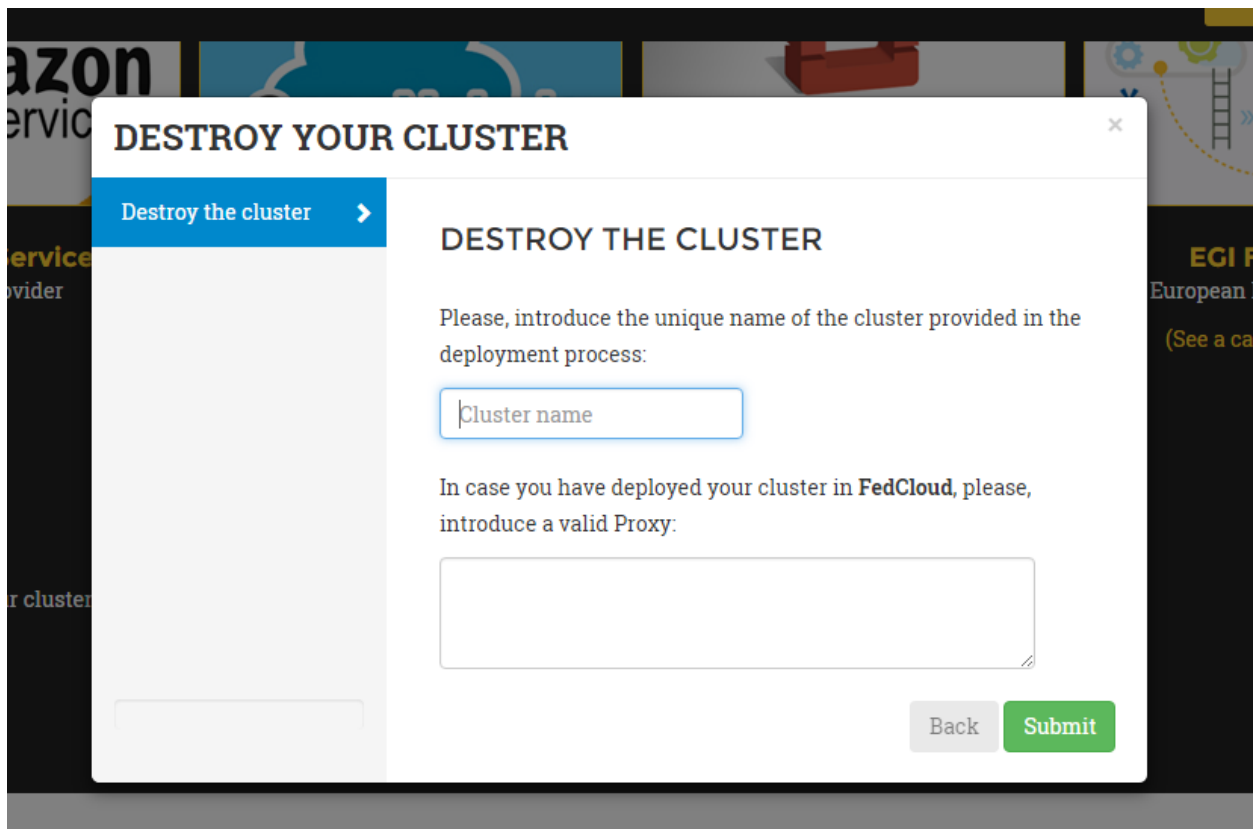

Fig. 4: Fig 4. Information received by the user when a deployment succeeds.

Notice that EC3aaS does not offer all the capabilities of EC3, like hybrid clusters or the usage of spot instances. Those capabilities are considered advanced aspects of the tool and are only available via the EC3 Command-line Interface.

5.3 Termination of a Cluster

To delete a cluster the user only needs to access the EC3aaS webpage, and click on the trash icon (situated underneath the provider buttons in the deployment section of the website) and indicate in the wizard (*Fig. 5*) the cluster name provided to the user in the deployment phase. The cluster name is a string composed by the word cluster followed by a random string of five characters (including numbers and letters). This cluster name is unique and allows EC3 to identify the cluster of the user without using an user account. Moreover, in case the user has developed the cluster in the EGI FedCloud, a valid proxy will be required in order to destroy the cluster.

When the process finishes successfully, the front-end of the cluster and all the working nodes had been destroyed and a message is shown to the user informing the success of the operation. If an error occurs during the deleting process (for example, the indicated cluster name does not exist), an error message is returned to the user.

Fig. 5: Fig 5. Wizard to delete a cluster.

Templates

EC3 recipes are described in a superset of RADL, which is a specification of virtual machines (e.g., instance type, disk images, networks, etc.) and contextualization scripts.

6.1 Basic structure

An RADL document has the following general structure:

```
network <network_id> (<features>)

system <system_id> (<features>)

configure <configure_id> (<Ansible recipes>)

deploy <system_id> <num> [<cloud_id>]
```

The keywords `network`, `system` and `configure` assign some *features* or *recipes* to an identity `<id>`. The features are a list of constrains separated by `and`, and a constrain is formed by `<feature name> <operator> <value>`. For instance:

```
system tomcat_node (
   cpu.count = 4 and
   memory.size >= 1024M and
   net_interface.0.connection = 'net'
)
```

This RADL defines a *system* with the feature `cpu.count` equal to four, the feature `memory.size` greater or equal than `1024M` and with the feature `net_interface.0.connection` bounded to `'net'`.

The `deploy` keyword is a request to deploy a number of virtual machines. Some identity of a cloud provider can be specified to deploy on a particular cloud.

6.2 EC3 types of Templates

In EC3, there are three types of templates:

- `images`, that includes the `system` section of the basic template. It describes the main features of the machines that will compose the cluster, like the operating system or the CPU and RAM memory required;

- `main`, that includes the `deploy` section of the frontend. Also, they include the configuration of the chosen LRMS.

- `component`, for all the recipes that install and configure software packages that can be useful for the cluster.

In order to deploy a cluster with EC3, it is mandatory to indicate in the `ec3 launch` command, *one* recipe of kind `main` and *one* recipe of kind `image`. The `component` recipes are optional, and you can include all that you need.

To consult the type (*kind*) of template from the ones offered with EC3, simply use the `ec3 templates` command like in the example above:

```
$ ./ec3 templates

        name              kind                                          summary
--------------------------------------------------------------------------------
↪-----------------------------
        blcr          component Tool for checkpoint the applications.
    centos-ec2         images   CentOS 6.5 amd64 on EC2.
    ckptman            component Tool to automatically checkpoint applications␣
↪running on Spot instances.
        docker         component An open-source tool to deploy applications inside␣
↪software containers.
        gnuplot        component A program to generate two- and three-dimensional␣
↪plots.
        nfs            component Tool to configure shared directories inside a␣
↪network.
        octave         component A high-level programming language, primarily␣
↪intended for numerical computations
        openvpn        component Tool to create a VPN network.
        sge               main    Install and configure a cluster SGE from␣
↪distribution repositories.
        slurm             main    Install and configure a cluster SLURM 14.11 from␣
↪source code.
        torque            main    Install and configure a cluster TORQUE from␣
↪distribution repositories.
    ubuntu-azure        images   Ubuntu 12.04 amd64 on Azure.
    ubuntu-ec2          images   Ubuntu 14.04 amd64 on EC2.
```

6.3 Network Features

Under the keyword `network` there are the features describing a Local Area Network (LAN) that some virtual machines can share in order to communicate to themselves and to other external networks. The supported features are:

outbound = yes|no Indicate whether the IP that will have the virtual machines in this network will be public (accessible from any external network) or private. If `yes`, IPs will be public, and if `no`, they will be private. The default value is `no`.

6.4 System Features

Under the keyword `system` there are the features describing a virtual machine. The supported features are:

image_type = vmdk|qcow|qcow2|raw Constrain the virtual machine image disk format.

virtual_system_type = '<hypervisor>-<version>' Constrain the hypervisor and the version used to deploy the virtual machine.

price <=|=|=> <positive float value> Constrain the price per hour that will be paid, if the virtual machine is deployed in a public cloud.

cpu.count <=|=|=> <positive integer value> Constrain the number of virtual CPUs in the virtual machine.

cpu.arch = i686|x86_64 Constrain the CPU architecture.

cpu.performance <=|=|=> <positive float value>ECU|GCEU Constrain the total computational performance of the virtual machine.

memory.size <=|=|=> <positive integer value>B|K|M|G Constrain the amount of *RAM* memory (main memory) in the virtual machine.

net_interface.<netId> Features under this prefix refer to virtual network interface attached to the virtual machine.

net_interface.<netId>.connection = <network id> Set the virtual network interface is connected to the LAN with ID <network id>.

net_interface.<netId>.ip = <IP> Set a static IP to the interface, if it is supported by the cloud provider.

net_interface.<netId>.dns_name = <string> Set the string as the DNS name for the IP assigned to this interface. If the string contains #N# they are replaced by a number that is distinct for every virtual machine deployed with this `system` description.

instance_type = <string> Set the instance type name of this VM.

disk.<diskId>.<feature> Features under this prefix refer to virtual storage devices attached to the virtual machine. `disk.0` refers to system boot device.

disk.<diskId>.image.url = <url> Set the source of the disk image. The URI designates the cloud provider:

- `one://<server>:<port>/<image-id>`, for OpenNebula;
- `ost://<server>:<port>/<ami-id>`, for OpenStack;
- `aws://<region>/<ami-id>`, for Amazon Web Service;
- `gce://<region>/<image-id>`, for Google Cloud;
- `azr://<image-id>`, for Microsoft Azure Clasic; and
- `azr://<publisher>/<offer>/<sku>/<version>`, for Microsoft Azure; and
- `<fedcloud_endpoint_url>/<image_id>`, for FedCloud OCCI connector.
- `appdb://<site_name>/<apc_name>?<vo_name>`, for FedCloud OCCI connector using AppDB info (from ver. 1.6.0).
- `docker://<docker_image>`, for Docker images.
- `fbw://<fogbow_image>`, for FogBow images.

Either `disk.0.image.url` or `disk.0.image.name` must be set.

disk.<diskId>.image.name = <string> Set the source of the disk image by its name in the VMRC server. Either `disk.0.image.url` or `disk.0.image.name` must be set.

disk.<diskId>.type = swap|iso|filesystem Set the type of the image.

disk.<diskId>.device = <string> Set the device name, if it is disk with no source set.

disk.<diskId>.size = <positive integer value>B|K|M|G Set the size of the disk, if it is a disk with no source set.

disk.0.free_size = <positive integer value>B|K|M|G Set the free space available in boot disk.

disk.<diskId>.os.name = linux|windows|mac os x Set the operating system associated to the content of the disk.

disk.<diskId>.os.flavour = <string> Set the operating system distribution, like `ubuntu`, `centos`, `windows xp` and `windows 7`.

disk.<diskId>.os.version = <string> Set the version of the operating system distribution, like `12.04` or `7.1.2`.

disk.0.os.credentials.username = <string> and **disk.0.os.credentials.password = <string>** Set a valid username and password to access the operating system.

disk.0.os.credentials.public_key = <string> and **disk.0.os.credentials.private_key = <string>** Set a valid public-private keypair to access the operating system.

disk.<diskId>.applications contains (name=<string>, version=<string>, preinstalled=yes|no) Set that the disk must have installed the application with name `name`. Optionally a version can be specified. Also if `preinstalled` is `yes` the application must have already installed; and if `no`, the application can be installed during the contextualization of the virtual machine if it is not installed.

6.5 Special EC3 Features

There are also other special features related with EC3. These features enable to customize the behaviour of EC3:

ec3_max_instances = <integer value> Set maximum number of nodes with this system configuration; a negative value means no constrain. The default value is -1. This parameter is used to set the maximum size of the cluster.

ec3_destroy_interval = <positive integer value> Some cloud providers require paying in advance by the hour, like AWS. Therefore, the node will be destroyed only when it is idle and at the end of the interval expressed by this option (in seconds). The default value is 0.

ec3_destroy_safe = <positive integer value> This value (in seconds) stands for a security margin to avoid incurring in a new charge for the next hour. The instance will be destroyed (if idle) in up to (`ec3_destroy_interval` - `ec3_destroy_safe`) seconds. The default value is 0.

ec3_if_fail = <string> Set the name of the next system configuration to try when no more instances can be allocated from a cloud provider. Used for hybrid clusters. The default value is ''.

ec3_inherit_from = <string> Name of the already defined `system` from which inherit its characteristics. For example, if we have already defined a `system wn` where we have specified cpu and os, and we want to change memory only for a new system, instead of writing again the values for cpu and os, we inherit these values from the specified system like `ec3_inherit_from = system wn`. The default value is 'None'.

ec3_reuse_nodes = <boolean> Indicates that you want to stop/start working nodes instead of powering off/on them. The default value is 'false'.

ec3_golden_images = <boolean> Indicates that you want to use the golden images feature. See golden images for more info. The default value is 'false'.

ec3_additional_vm = <boolean> Indicates that you want this VM to be treated as an additional VM of the cluster, for example, to install server services that you do not want to put in the front machine. The default value is 'false'.

ec3_node_type = <string> Indicates the type of the node. Currently the only supported value is `wn`. It enables to distinguish the WNs from the rest of nodes. The default value is 'None'.

ec3_node_keywords = <string> Comma separated list of pairs key=value that specifies some specific features supported by this type of node (i.e. gpu=1,infiniband=1). The default value is 'None'.

ec3_node_queues_list = <string> Comma separated list of queues this type of node belongs to. The default value is 'None'.

ec3_node_pattern = <string> A pattern (as a Python regular expression) to match the name of the virtual nodes with the current node type The value of this variable must be set according to the value of the variable `ec3_max_instances`. For example if `ec3_max_instances` is set to 5 a valid value can be: 'wn[1-5]'. This variable has preference over `ec3_if_fail` so if a virtual node to be switched on matches with the specified pattern ''ec3_if_fail' variable will be ignored. The default value is 'None'.

6.6 System and network inheritance

It is possible to create a copy of a system or a network and to change and add some features. If feature `ec3_inherit_from` is presented, ec3 replaces that object by a copy of the object pointed out in `ec3_inherit_from` and appends the rest of the features.

Next example shows a system `wn_ec2` that inherits features from system `wn`:

```
system wn (
    ec3_if_fail = 'wn_ec2' and
    disk.0.image.url = 'one://myopennebula.com/999' and
    net_interface.0.connection='public'
)

system wn_ec2 (
    ec3_inherit_from = system wn and
    disk.0.image.url = 'aws://us-east-1/ami-e50e888c' and
    spot = 'yes' and
    ec3_if_fail = ''
)
```

The system `wn_ec2` that ec3 sends finally to IM is:

```
system wn_ec2 (
    net_interface.0.connection='public' and
    disk.0.image.url = 'aws://us-east-1/ami-e50e888c' and
    spot = 'yes' and
    ec3_if_fail = ''
)
```

In case of systems, if system *A* inherits features from system *B*, the new configure section is composed by the one from system *A* followed by the one of system *B*. Following the previous example, these are the configured named after the systems:

```
configure wn (
@begin
- tasks:
```

(continues on next page)

```
    - user: name=user1    password=1234
@end
)

configure wn_ec2 (
@begin
- tasks:
    - apt: name=pkg
@end
)
```

Then the configure `wn_ec2` that ec3 sends finally to IM is:

```
configure wn_ec2 (
@begin
- tasks:
    - user: name=user1    password=1234
- tasks:
    - apt: name=pkg
@end
)
```

6.7 Configure Recipes

Contextualization recipes are specified under the keyword `configure`. Only Ansible recipes are supported currently. They are enclosed between the tags `@begin` and `@end`, like that:

```
configure add_user1 (
@begin
---
  - tasks:
    - user: name=user1    password=1234
@end
)
```

6.7.1 Exported variables from IM

To easy some contextualization tasks, IM publishes a set of variables that can be accessed by the recipes and have information about the virtual machine.

IM_NODE_HOSTNAME Hostname of the virtual machine (without the domain).

IM_NODE_DOMAIN Domain name of the virtual machine.

IM_NODE_FQDN Complete FQDN of the virtual machine.

IM_NODE_NUM The value of the substitution `#N#` in the virtual machine.

IM_MASTER_HOSTNAME Hostname (without the domain) of the virtual machine doing the *master* role.

IM_MASTER_DOMAIN Domain name of the virtual machine doing the *master* role.

IM_MASTER_FQDN Complete FQDN of the virtual machine doing the *master* role.

6.7.2 Including a recipe from another

The next RADL defines two recipes and one of them (`add_user1`) is called by the other (`add_torque`):

```
configure add_user1 (
@begin
---
  - tasks:
    - user: name=user1   password=1234
@end
)

configure add_torque (
@begin
---
  - tasks:
    - include: add_user1.yml
    - yum: name=torque-client,torque-server state=installed
@end
)
```

6.7.3 Including file content

If in a `vars` map a variable has a map with key `ec3_file`, ec3 replaces the map by the content of file in the value.

For instance, there is a file `slurm.conf` with content:

```
ControlMachine=slurmserver
AuthType=auth/munge
CacheGroups=0
```

The next ansible recipe will copy the content of `slurm.conf` into `/etc/slurm-llnl/slurm.conf`:

```
configure front (
@begin
  - vars:
      SLURM_CONF_FILE:
        ec3_file: slurm.conf
    tasks:
    - copy:
        dest: /etc/slurm-llnl/slurm.conf
        content: "{{SLURM_CONF_FILE}}"
@end
)
```

> **Warning:** Avoid using variables with file content in compact expressions like this:
> ```
> - copy: dest=/etc/slurm-llnl/slurm.conf content={{SLURM_CONF_FILE}}
> ```

6.7.4 Include RADL content

Maps with keys `ec3_xpath` and `ec3_jpath` are useful to refer RADL objects and features from Ansible vars. The difference is that `ec3_xpath` prints the object in RADL format as string, and `ec3_jpath` prints objects as YAML maps. Both keys support the next paths:

- /<class>/*: refer to all objects with that <class> and its references; e.g., /system/* and /network/*.

- /<class>/<id> refer to an object of class <class> with id <id>, including its references; e.g., /system/front, /network/public.

- /<class>/<id>/* refer to an object of class <class> with id <id>, without references; e.g., /system/front/*, /network/public/*

Consider the next example:

```
network public ( )

system front (
    net_interface.0.connection = 'public' and
    net_interface.0.dns_name = 'slurmserver' and
    queue_system = 'slurm'
)

system wn (
  net_interface.0.connection='public'
)

configure slum_rocks (
@begin
  - vars:
      JFRONT_AST:
          ec3_jpath: /system/front/*
      XFRONT:
          ec3_xpath: /system/front
    tasks:
    - copy: dest=/tmp/front.radl
      content: "{{XFRONT}}"
      when: JFRONT_AST.queue_system == "slurm"
@end
)
```

RADL configure slurm_rocks is transformed into:

```
configure slum_rocks (
@begin
- vars:
    JFRONT_AST:
      class: system
      id: front
      net_interface.0.connection:
        class: network
        id: public
        reference: true
      net_interface.0.dns_name: slurmserver
      queue_system: slurm
    XFRONT: |
      network public ()
      system front (
          net_interface.0.connection = 'public' and
          net_interface.0.dns_name = 'slurmserver' and
          queue_system = 'slurm'
      )
  tasks:
```

Chapter 6. Templates

```
  - content: '{{XFRONT}}'
    copy: dest=/tmp/front.radl
    when: JFRONT_AST.queue_system == "slurm"
@end
)
```

6.8 Adding your own templates

If you want to add your own customized templates to EC3, you need to consider some aspects:

- For `image` templates, respect the frontend and working nodes nomenclatures. The system section for the frontend *must* receive the name `front`, while at least one type of working node *must* receive the name `wn`.

- For `component` templates, add a `configure` section with the name of the component. You also need to add an `include` statement to import the configure in the system that you want. See Including a recipe from another for more details.

Also, it is important to provide a `description` section in each new template, to be considered by the `ec3 templates` command.

Frequently Asked Questions

These are some frequently asked questions that might solve your doubts when using EC3.

7.1 General FAQs

What Cloud Providers are supported by EC3 (Elastic Cloud Computing Cluster)?

Currently, EC3 supports OpenNebula, Amazon EC2, OpenStack, OCCI, LibCloud, Docker, Microsoft Azure, Google Cloud Engine and LibVirt. All providers and interfaces are supported by the CLI interface. However, from the EC3aaS interface, only support for Amazon EC2, Openstack, OpenNebula and EGI FedCloud is provided. More providers will be added soon, stay tunned!

What Local Resource Management Systems (LRMS) are supported by EC3?

Currently, EC3 supports SLURM, Torque, Apache Mesos, SGE, HTCondor and Kubernetes.

Is it necessary to indicate a LRMS recipe in the deployment?

Yes, it is *mandatory*, because the cluster needs to have an LRMS system installed. This is why the LRMS recipes are considered *main* recipes, needed to perform a deployment with EC3.

Is it secure to provide my credentials to EC3?

The user credentials that you specify are *only* employed to provision the resources (Virtual Machines, security groups, keypairs, etc.) on your behalf. No other resources will be accessed/deleted. However, if you are concerned about specifying your credentials to EC3, note that you can (and should) create an additional set of credentials, perhaps with limited privileges, so that EC3 can access the Cloud on your behalf. In particular, if you are using Amazon Web Services, we suggest you use the Identity and Access Management (IAM) service to create a user with a new set of credentials. This way, you can rest assured that these credentials can be cancelled at anytime.

Can I configure different software packages than the ones provided with EC3 in my cluster?

Yes, you can configure them by using the EC3 CLI interface. Thus, you will need to provide a valid Ansible recipe to automatically install the dependence. You can also contact us by using the contact section, and we would try to add the software package you need.

Why am I experimenting problems with Centos 6 when trying to deploy a Mesos cluster?

Because the recipe of Mesos provided with EC3 is optimized for Centos 7 as well as Ubuntu 14.04. If you want to deploy a Mesos cluster, we encourage you to use one of each operative systems.

Which is the best combination to deploy a Galaxy cluster?

The best configuration for a elastic Galaxy cluster is to select Torque as a LRMS and install the NFS package. Support for Galaxy in SGE is not provided. Moreover, we have detected problems when using Galaxy with SLURM. So, we encourage you to use Torque and NFS in the EC3aaS and also with the EC3 CLI.

7.2 EC3aaS Webpage

Is my cluster ready when I receive its IP using the EC3aaS webpage?

Probably not, because the process of configuring the cluster is a batch process that takes several minutes, depending on the chosen configuration. However, you are allowed to log in the front-end machine of the cluster since the moment it is deployed. To know if the cluster is configured, you can use the command *is_cluster_ready*. It will check if the cluster has been configured or if the configuration process is still in progress. If the command *is_cluster_ready* is not recognised, wait a few seconds and try again, because this command is also installed in the configuration process.

Why can't I deploy an hybrid cluster using the EC3aaS webpage?

Because no support is provided yet by the EC3aaS service. If you want to deploy a hybrid cluster, we encourage you to use the CLI interface.

Why can I only access to Amazon EC2, Openstack, OpenNebula and EGI FedCloud Cloud providers while other Cloud providers are supported by EC3?

Because no support is provided yet by the EC3aaS service. If you want to use another supported Cloud provider, like Microsoft Azure or Google Cloud Engine, we encourage you to use the CLI interface.

What is the correct format for the "endpoint" in the OpenNebula and Openstack wizards?

The user needs to provide EC3 the endpoint of the on-premises Cloud provider. The correct format is *name_of_the_server:port*. For example, for Openstack *ostserver:5000*, or for OpenNebula *oneserver:2633*. The same format is employed in the authorization file required to use the CLI interface of EC3.

Why am I receiving this error "InvalidParameterCombination - Non-Windows instances with a virtualization type of 'hvm' are currently not supported for this instance type" when I deploy a cluster in Amazon EC2?

This error is shown by the Cloud provider, because the instance type and the Amazon Machine Image selected are incompatible. The Linux AMI with HVM virtualization cannot be used to launch a non-cluster compute instance. Select another AMI with a virtualization type of paravirtual and try again.

Why am I receiving this error "VPCResourceNotSpecified - The specified instance type can only be used in a VPC. A subnet ID or network interface ID is required to carry out the request." when I deploy a cluster in Amazon EC2?

This error is shown by the Cloud provider, because the instance type selected can only be used in a VPC. To use a VPC, please, employ the CLI interface of EC3. You can specify the name of an existent VPC in the RADL file. More info about Amazon VPC.

Why can't I download the private key of my cluster?

If you are experimenting problems downloading the private key of your cluster (deployed in Amazon EC2), please, try with another browser. The website is currently optimized for Google Chrome.

Where can I get the endpoint and VMI identifier for the EGI FedCloud wizard?

In the EGI FedCloud case, the endpoint and VMI identifier can be obtained from the AppDB portal. In the cloud marketplace select the desired VMI then select the site to launch it (considering your VO) and click the "get IDs" button. The field "Site endpoint" shows the value of the endpoint to specify in the wizard (without a "/" character after the port) and the value after the "#" char of the OCCI ID field the VMI Indentifier. Finally the value after the "#" char of the Template ID field shows the type of the instance type (In some OpenStack sites you must replace the "." char with a "-", e.g. m1.small to m1-small).

Can I configure software packages in my cluster that are not available in the wizard?

You can configure them by using the EC3 CLI interface. Thus, you will need to provide a valid Ansible recipe to automatically install the dependence. You can also contact us by using the contact section, and we would try to add the software package you need.

What is the OSCAR option that appears as a LRMS?

In OpenNebula and EGI Fedcloud there is an option to deploy as an LRMS the OSCAR (Open Source Serverless Computing for Data-Processing Applications) framework, that is an open-source platform to support the Functions as a Service (FaaS) computing model for file-processing applications. This option deploys a Kubernetes cluster with the OSCAR framework and all its dependences.

About

EC3 has been developed by the Grid and High Performance Computing Group (GRyCAP) at the Instituto de Instrumentación para Imagen Molecular (I3M) from the Universitat Politècnica de València (UPV).

This development has been supported by the following research projects:

The following publications summarise both the development and integration in larger architecture. Please acknowledge the usage of this software by citing the last reference:

- Caballer, M.; de Alfonso, C.; Alvarruiz, F. and Moltó, G.; "EC3: Elastic Cloud Computing Cluster". Journal of Computer and System Sciences, Volume 78, Issue 8, December 2013, Pages 1341-1351, ISSN 0022-0000,

10.1016/j.jcss.2013.06.005.

- Calatrava, A.; Caballer, M.; Moltó, G.; and de Alfonso, C.; "Virtual Hybrid Elastic Clusters in the Cloud". Proceedings of 8th IBERIAN GRID INFRASTRUCTURE CONFERENCE (Ibergrid), pp. 103 - 114 ,2014.

- "Custom elastic clusters to manage Galaxy environments". In: EGI Inspired Newsletter (Issue 22), pp 2, January 2016. Available here.

- Calatrava, A,; Romero, E.; Caballer, M.; Moltó, G.; and Alonso, J.M.; "Self-managed cost-efficient virtual elastic clusters on hybrid Cloud infrastructures". Future Generation Computer Systems, 2016. doi:10.1016/j.future.2016.01.018.

Preprints are available here.

Also, EC3 has been integrated in the EGI Platform for the long-tail of science (access available through here), and it is available as one of the services of the European Open Science Cloud Marketplace

CHAPTER 9

Indices and tables

- genindex
- search

C

clustername
    ec3-launch command line option, 15
    ec3-restart command line option, 19
    ec3-stop command line option, 19
    ec3-transfer command line option, 19

E

ec3 command line option
    -l <file>, -log-file <file>, 15
    -ll <level>, -log-level <level>, 15
    -q, -quiet, 15
ec3-clone command line option
    -a <file>, -auth-file <file>, 18
    -d <provider>, -destination
        <provider>, 18
    -e, -eliminate, 18
    -u <url>, -restapi-url <url>, 18
ec3-destroy command line option
    -force, 17
ec3-launch command line option
    -add, 15
    -dry-run, 16
    -json, 16
    -on-error-destroy, 16
    -a <file>, -auth-file <file>, 16
    -g, -golden-images, 16
    -n, -not-store, 16
    -p, -print, 16
    -u <url>, -restapi-url <url>, 16
    -y, -yes, 16
    clustername, 15
    template_0 ..., 15
ec3-list command line option
    -json, 17
    -r, -refresh, 17
ec3-migrate command line option
    -a <file>, -auth-file <file>, 18
    -b <bucket_name>, -bucket
        <bucket_name>, 18
    -d <provider>, -destination
        <provider>, 19
    -e, -eliminate, 19
    -u <url>, -restapi-url <url>, 19
ec3-reconfigure command line option
    -add, 16
    -template, -t, 16
    -a <file>, -auth-file <file>, 16
    -r, -reload, 16
ec3-restart command line option
    -a <file>, -auth-file <file>, 19
    -u <url>, -restapi-url <url>, 19
    clustername, 19
ec3-show command line option

    -json, 17
    -r, -refresh, 17
ec3-ssh command line option
    -show-only, 17
ec3-stop command line option
    -a <file>, -auth-file <file>, 19
    -u <url>, -restapi-url <url>, 19
    -y, -yes, 19
    clustername, 19
ec3-templates command line option
    -json, 18
    -f, -full-description, 18
    -n, -name, 18
    -s, -search, 18
ec3-transfer command line option
    -a <file>, -auth-file <file>, 20
    -u <url>, -restapi-url <url>, 20
    clustername, 19
ec3-update command line option
    -add, 20
    -a <file>, -auth-file <file>, 20

T

template_0 ...
    ec3-launch command line option, 15