

Kadeploy 3: Installation, configuration and use

June 28, 2021

Contents

0	Overview	5
0.1	What is it?	5
0.2	How it works?	5
0.3	How does Kadeploy control the boot of the nodes ?	5
1	Installation	7
1.1	Requirements	7
1.1.1	Packages	7
1.1.2	DHCP and TFTP	7
1.1.3	HTTP server (optional)	9
1.1.4	MySQL	9
1.1.5	TakTuk	9
1.2	Kadeploy installation	10
1.2.1	Installation with Rake	10
1.2.2	Build packages	10
1.2.3	RedHat packages	11
1.3	Launching the Kadeploy server	12
1.3.1	Automatic launch on a Debian and a RedHat based distribution	12
2	Server side configuration	13
2.1	General configuration file	15
2.1.1	Example of a general configuration file	15
2.1.2	Explanation of the fields used in the general configuration file	18
2.2	Booting over the network	24
2.3	Clusters file	25
2.3.1	Example of a clusters file	26
2.3.2	Explanation of the fields used in the clusters file	26
2.4	Cluster-specific configuration files	26
2.4.1	Example of a cluster-specific configuration file	27
2.4.2	Explanation of the fields used in the cluster-specific configuration file	30
2.5	Bootloader install script	39
2.6	Partitioning script	40

2.7	Formating script	40
2.8	Specific commands configuration files	40
2.8.1	Example of a commands file	40
2.8.2	Explanation of the fields used in the commands file	40
2.9	Deployment environment	41
2.9.1	Configuration of the production environment	41
2.9.2	Creation of the dedicated environment	41
2.9.3	Creation of the NFSRoot environment	42
2.10	Configuration of the deploy user	43
2.11	Configuration of SSH-agent	43
3	Client side configuration	44
4	User guide	46
4.1	Overview of the Kadeploy tools	46
4.1.1	Kadeploy	46
4.1.2	Kareboot	46
4.1.3	Kaenv	46
4.1.4	Kaconsole	46
4.1.5	Kastat	46
4.1.6	Kanodes	46
4.1.7	Kapower	46
4.1.8	Karights	47
4.2	Use the Kadeploy tools	47
4.2.1	Kadeploy server	47
4.2.2	Kadeploy client	47
4.2.3	Kareboot	52
4.2.4	Kaenv	53
4.2.5	Kaconsole	57
4.2.6	Kastat	58
4.2.7	Kanodes	59
4.2.8	Kapower	60
4.2.9	Karights	60
4.3	What you should know if you want to do kernel development on deployed nodes . .	61
4.3.1	Kadeploy 3 behavior	61
4.3.2	Tips to simply use your new kernel	62
4.4	Extra	63
4.4.1	Kadeploy3 Environment variables	63
4.4.2	Specifying files to the server	64
4.4.3	Build a custom pre-install	64
4.4.4	Do a custom partitioning	65
4.4.5	Fsarchiver environnements	66

About this document

This is the Kadeploy 3.6.0.rc3 documentation file.

It contains a short Overview of Kadeploy, followed by the Installation instructions, a description of the Server side configuration and the Client side configuration , to finish with the User guide.

For a better understanding of how Kadeploy3 works see this publication:
<http://hal.inria.fr/docs/00/71/06/38/PDF/RR-8002.pdf>

More informations, source code and bug tracker available here:
<https://kadeploy.gitlabpages.inria.fr>

Chapter 0

Overview

0.1 What is it?

Kadeploy is a scalable, efficient and reliable deployment system (cluster provisioning solution) for cluster and grid computing. It provides a set of tools for cloning, configuring (post installation) and managing cluster nodes. It can deploy a 300-nodes cluster in a few minutes, without intervention from the system administrator. It can deploy Linux, *BSD, Windows, Solaris.

It plays a key role on the Grid'5000 testbed, where it allows users to reconfigure the software environment on the nodes.

0.2 How it works?

This is how Kadeploy works:

1. Minimal environment setup The nodes reboot into a trusted minimal environment that contains all the tools required for the deployment (partitioning tools, archive management, ...) and the required partitioning is performed.
2. Environment installation The environment is sent to all the nodes and extracted on the disks. Some post-installations operations can also be performed.
3. Reboot on the deployed environment

Kadeploy3 takes as input an archive containing the operating system to deploy, called an **environment**, and copies it on the target nodes. As a consequence, Kadeploy3 does not install an operating system following a classical installation procedure and the user has to provide an archive of the operating system's filesystem (as a tarball, for Linux environments).

0.3 How does Kadeploy control the boot of the nodes ?

This is how Kadeploy controls the boot process of the nodes in order to be able to perform the installation tasks:

1. Kadeploy writes PXE profiles on a TFTP or HTTP server
2. Kadeploy triggers the reboot of compute nodes using SSH, IPMI or a manageable PDU
3. Nodes get their configuration using DHCP
4. Nodes retrieve their PXE profile using TFTP
5. Nodes boot on the specified system (which can either be located on the node's hard disk or on the network)

Chapter 1

Installation

1.1 Requirements

1.1.1 Packages

Kadeploy requires the following softwares (the given packages names are valid for the Debian/Wheezy distribution):

- ruby \geq 1.8.7
- ruby-mysql
- taktuk \geq 3.6
- isc-dhcp-server
- syslinux
- tftpd-hpa

1.1.2 DHCP and TFTP

The DHCP service

A DHCP server (isc-dhcp-server on Debian for instance) must be configured to provide a static IP address to the set of nodes that must be deployed. Furthermore, the DHCP response must contain the hostname of the node (see the use-host-decl-names on; option in dhcpd.conf).

Here is an example of a configuration for PXELinux:

```
default-lease-time 28800;  
max-lease-time 86400;  
allow booting;  
allow bootp;  
not-authoritative;  
use-host-decl-names on;
```



```
subnet 192.168.0.0 netmask 255.255.255.0 {
    option subnet-mask 255.255.255.0;
    option broadcast-address 192.168.0.255;
    option routers 192.168.0.254;
    option domain-name "testbed.lan";
    filename "pxelinux.0";
    next-server 192.168.0.1;

    host node-1.testbed.lan {
        hardware ethernet 00:09:3d:12:33:e6;
        fixed-address 192.168.0.10;
        option host-name "node-1";
    }
    host node-2.testbed.lan {
        hardware ethernet 00:09:3d:12:33:e7;
        fixed-address 192.168.0.11;
        option host-name "node-2";
    }
}
```

More information about the configuration of PXELinux can be found at <http://www.syslinux.org/wiki/index.php/PXELINUX>.

Booting over the network

To allow the network booting, you must specify in the DHCP configuration file the filename option. This option defines the name of file which will be downloaded at the boot time. This file name can be pxelinux.0, gpxelinux.0, or ipxeLinux.0.

Finally, the TFTP repository (see 2.1 part) must contain the following files and directories:

- pxelinux.0 (or similar)
- chain.c32
- mboot.c32
- a kernels/ directory (can be changed in the server configuration file)
- a pxelinux.cfg/ directory

These files can be found in the Syslinux software (<http://syslinux.org>) or directly downloaded on the kernel.org website (<https://www.kernel.org/pub/linux/utils/boot/syslinux/>), the 3.73 version is at least required.

The TFTP service

A TFTP server (tftpd-hpa on Debian for instance) must be installed. Configuration example:

```
# /etc/default/tftpd-hpa
TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/var/lib/tftpboot"
TFTP_ADDRESS="0.0.0.0:69"
TFTP_OPTIONS="-v -l -c -s"
```

1.1.3 HTTP server (optional)

In order to use the HTTP fetching capabilities of gpxlinux or iPXE, an HTTP server must be configured and must contain the production environment kernel/initrd and the deployment environment kernel/initrd (see 2.4.2 part).

1.1.4 MySQL

A MySQL server must be configured with a database and a user dedicated to Kadeploy. The rights on this database must be granted to the chosen user, from the Kadeploy server. The server used to host the database, the database name, the dedicated user and its password must be specified in the general Kadeploy configuration (see 2.1 part).

Just provided as an example, let's see a way to create the database `deploy3` and to give the suitable rights to the `deploy` user.

```
mysql> CREATE DATABASE deploy3;
mysql> GRANT select, insert, update, delete, create, drop, alter, \
      create temporary tables, lock tables ON deploy3.* \
      TO 'deploy'@'kadeploy.site.grid5000.fr';
```

Once the database is created and the user granted, you can use the SQL script provided in the distribution (`db/db_creation.sql`) to create the tables in the database.

```
mysql> use deploy3
mysql> source /usr/share/doc/kadeploy/db_creation.sql;
```

1.1.5 TakTuk

Kadeploy3 requires TakTuk, a powerful tool to achieve remote executions. Thus it must be installed on the Kadeploy3 server.

Note for Debian users

TakTuk is available in the official repositories.

Note for RedHat users

TakTuk is not packaged for RedHat-based distributions. Some RPM packages are available on the Kadeploy3 website: <http://kadeploy3.gforge.inria.fr/files/taktuk/>. It's also possible to install TakTuk from the sources, more information on the project's website: <http://taktuk.gforge.inria.fr/>.

1.2 Kadeploy installation

Since Kadeploy is based on a client/server architecture, it must be installed both on the server and on the client side (in case of distinct hosts).

Two ways are provided to install Kadeploy: 1) the packages available in the release section at <https://kadeploy.gitlabpages.inria.fr/> (for Debian) and 2) Rake installation from sources. In both cases, you have to ensure that a user deploy exists on the system since it is used to execute the Kadeploy server. Furthermore, all the installation operations must be performed with root rights.

1.2.1 Installation with Rake

First of all, you have to uncompress the Kadeploy tarball.

```
> tar xzf kadeploy-3.6.0.rc3.tar.gz -C DESTINATION_DIR
```

Then, if you want to install the server part, just execute:

```
> rake install_server
```

If you want to install the client part, execute:

```
> rake install_client
```

If you want to install the server part and the client part on the same host, execute:

```
> rake install
```

remark: If there are already configuration files, the new example of configuration files will be installed with .dist extension.

If you want to install the rc script, you can add the DISTRIB flag. Currently, only Debian (it includes Ubuntu at least) and RedHat (it should include CentOS and RHEL) values are supported. For instance, you can execute:

```
> rake install[root_dir,redhat]
```

or if you do not install the server side on the same machine than the client side:

```
> rake install_server[root_dir,redhat]
```

Finally, Kadeploy can be simply uninstalled by executing:

```
> rake uninstall
```

In case of uninstallation, the configuration directory /etc/kadeploy3 is not removed.

1.2.2 Build packages

The following installation method works only on Debian based distribution.

Build Debian Package

First, you have to uncompress the Kadeploy tarball.

```
> tar xzf kadeploy-3.6.0.rc3.tar.gz -C DESTINATION_DIR
```

Then you must generate the packages. So you have to execute:

```
> rake deb
```

This will generate three Debian package: `kadeploy-common-3.6.0.rc3 .deb`, `kadeploy-client-3.6.0.rc3 .deb`, and `kadeploy-3.6.0.rc3 .deb`.

Installation

On the server side, you have to install the `kadeploy-common-3.6.0.rc3 .deb` and `kadeploy-3.6.0.rc3 .deb` packages.

```
> dpkg -i kadeploy-common-3.6.0.rc3.deb
> dpkg -i kadeploy-3.6.0.rc3.deb
```

On the client side, you have to install the `kadeploy-common-3.6.0.rc3 .deb` and `kadeploy-client-3.6.0.rc3 .deb` packages.

```
> dpkg -i kadeploy-common-3.6.0.rc3.deb
> dpkg -i kadeploy-client-3.6.0.rc3.deb
```

In the want to use the same host for the client and the server part, just install the three packages:

```
> dpkg -i kadeploy-common-3.6.0.rc3.deb
> dpkg -i kadeploy-client-3.6.0.rc3.deb
> dpkg -i kadeploy-3.6.0.rc3.deb
```

Warning In order to preserve your configuration files, the removal of a Kadeploy package will preserve the configuration files (unless you specify the `--purge` tag).

1.2.3 RedHat packages

The following installation method works only on a RedHat based distribution. We assume that you have a configured rpm build environment. Furthermore, Taktuk must be installed on the server side.

Build

First, you have to uncompress the Kadeploy tarball.

```
> tar xzf kadeploy-3.6.0.rc3.tar.gz -C DESTINATION_DIR
```

Then you must generate the packages. So you have to execute with root rights:

```
> rake rpm
```

This will generate three rpm package in the RPMS package of your build environment, for instance: `kadeploy-client-3.6.0.rc3 .noarch.rpm`, `kadeploy-server-3.6.0.rc3 .noarch.rpm`, and `kadeploy-common-3.6.0.rc3 .noarch.rpm`.

Installation

On the server side, you have to install the `kadeploy-common-3.6.0.rc3.noarch.rpm` and `kadeploy-server-3.6.0.rc3.noarch.rpm` packages.

```
> rpm -i kadeploy-common-3.6.0.rc3.noarch.rpm
> rpm -i kadeploy-server-3.6.0.rc3.noarch.rpm
```

On the client side, you have to install the `kadeploy-common-3.6.0.rc3.noarch.rpm` and `kadeploy-client-3.6.0.rc3.noarch.rpm` packages.

```
> rpm -i kadeploy-common-3.6.0.rc3.noarch.rpm
> rpm -i kadeploy-client-3.6.0.rc3.noarch.rpm
```

In the want to use the same host for the client and the server part, just install the three packages:

```
> rpm -i kadeploy-common-3.6.0.rc3.noarch.rpm
> rpm -i kadeploy-server-3.6.0.rc3.noarch.rpm
> rpm -i kadeploy-client-3.6.0.rc3.noarch.rpm
```

1.3 Launching the Kadeploy server

After being installed and configured, the Kadeploy server can be run either interactively:

```
> /usr/sbin/kadeploy3d
```

or in background using the rc script:

```
> service kadeploy start
```

1.3.1 Automatic launch on a Debian and a RedHat based distribution

On a these distributions, if you use the provided packages, the rc script will be automatically launched at the startup.

Chapter 2

Server side configuration

Configuration files

Normally, the configuration of Kadeploy is located in `/etc/kadeploy3` but it can be located anywhere else if you set the `KADEPLOY_CONFIG_DIR` variable in the environment.

The file `load_kadeploy_env` in the configuration directory contains the `KADEPLOY_INSTALL_DIR` variable. You should probably fill this variable with the Kadeploy installation directory you used. This directory can be anywhere in the filesystem.

Description format: YAML

In Kadeploy configuration settings are given using the YAML markup language. You should be aware that, in this language, indentation is very important. Also, in the YAML language, fields are typed, the value `"16"` is not equivalent to the value `16`.

YAML types

In Kadeploy configuration files, values can have the YAML data types: *Integer*, *Float*, *Boolean* and *String*.

YAML provides a way to describe hierarchy between elements using *Associative arrays* (key → value) and *Ordered lists*. It's possible to mix this structures.

Here are some examples:

example-array: # This is an Associative array containing 3 elements

 elem1: 8 # Integer

 elem2: "8" # String

 elem3: vREF1

example-list: # This is an Ordered list of 2 elements

- true # Boolean

- "true" # String

example-mix-1: # An Ordered list of identical Associative arrays

- elem1: value1 # String

 elem2: vREF2

- elem1: value2 # String

 elem2: vREF3

```

example-mix-2: # An Associative array of Ordered lists
  elem1:
    - 1.42 # Float
    - value1
  elem2:
    - value2
    - value3
example-complex: # Complex structure
  mylist:
    - size: 16
      name: vREF4
    - size: 32
      name: vREF5
  value: myval
  myexample:
    file: filename
    ext: ext
    mode: vREF6

```

Documentation: paths

Kadeploy configuration settings are described in YAML files. A *path* defines the hierarchy structure to specify a setting in the configuration file. In a *path*, */* describes a nested *Associative array*, [...] describes an *Ordered list* of identical *Associative arrays*.

Example of *paths*:

- /example-array/elem3 refers to the value vREF1;
- /[example-mix-1]/elem2 refers to values such as vREF2 and vREF3;
- /example-complex/[mylist] refers to the *Ordered List* mylist;
- /example-complex/[mylist]/name refers to values such as vREF4 and vREF5;
- /example-complex/myexample/mode refers to value vREF5.

Documentation: configuration files fields

In the following, fields descriptions are given using the formalism:

- /path/to/the/field
 - fieldname {*YAML type*} (default value): description of the field

If no default value is specified, the field is mandatory.

Example of field description:

- /example-complex
 - myvalue {*String*}: the value of the element
- /example-complex/myexample

- file *{String}* (example): the name of the example file
- ext *{String}* (txt): the extension of the example file
- /example-complex/[mylist]
 - name *{String}*: the name of the element
 - size *{Integer}* (8): the maximal size (MB) of the element

2.1 General configuration file

The general configuration file is named `server.conf` and is located in the Kadeploy configuration directory.

2.1.1 Example of a general configuration file

```
---
database:
  host: mysql.lan
  name: deploy3
  login: deploy_user
  passwd: deploy_password
  kind: mysql
rights:
  kind: db
  almighty_users: root,superuser
  purge_deployment_timer: 900
authentication:
  global:
    headers_prefix: X-Kadeploy-
  certificate:
    #ca_public_key:
    # algorithm: RSA
    # file: /etc/kadeploy/ca_key.pub
    ca_cert: ca_cert.pem
  whitelist:
    - 192.168.0.0/24
    - kadeploy.mydomain.tld
  #http_basic:
  # dbfile: kadeploy.htpasswd
  # realm: Kadeploy
  # whitelist:
  # - frontend.mydomain.tld
  # - 192.168.0.4
  # - 192.168.0.8
  ident:
    whitelist:
      - /^.*\.mydomain\.tld$/
      - 192.168.0.0/24
```



```
- kadeploy.mydomain.tld
security:
  secure_server: true
  local_only: false
  #certificate: cert.pem
  #private_key:
  # algorithm: RSA
  # file: /home/deploy/key.pem
  force_secure_client: false
logs:
  logfile: /var/log/kadeploy/kadeploy.log
  debugfile: /var/log/kadeploy/kadeploy.debug
  database: true
  debug: true
verbosity:
  clients: 3
  logs: 4
cache:
  directory: /var/cache/kadeploy
  size: 8000
  concurrency_level: 10
network:
  server_hostname: kadeploy.lan
  vlan:
    set_cmd: kavlan NODES -s -i VLAN_ID -u USER
    hostname_suffix: -kavlan-VLAN_ID
ports:
  ssh: 22
  kadeploy_server: 25300
  test_deploy_env: 25300
  tcp_buffer_size: 8192
windows:
  check:
    size: 90
  reboot:
    size: 100
    sleep_time: 10
environments:
  deployment:
    extraction_dir: /mnt/dest
    tarball_dir: /tmp
    rambin_dir: /rambin
  allowed_name_regex: '[A-Za-z0-9\-\_\_]+\''
  max_postinstall_size: 10
  max_preinstall_size: 10
pxe:
  dhcp:
    method: PXELinux
    repository: /var/lib/tftpboot
    export:
```

```

    kind: tftp
    server: kadeploy-server
profiles:
  directory: pxelinux.cfg
  filename: ip_hex
userfiles:
  directory: userfiles
  max_size: 200
  concurrency_level: 10
localboot:
  method: GrubPXE
  binary: grubpxe.0
  repository: /var/lib/tftpboot
export:
  kind: tftp
  server: kadeploy-server
profiles:
  directory: grub.cfg
  filename: ip
autoclean_threshold: 360
hooks:
  end_of_reboot: echo REBOOT_ID
  end_of_power: echo POWER_ID
  end_of_deployment: echo WORKFLOW_ID
external:
  taktuk:
    auto_propagate: false
    connector: |-
      ssh -A -q -o StrictHostKeyChecking=no \
      -o UserKnownHostsFile=/dev/null \
      -o PreferredAuthentications=publickey \
      -o BatchMode=yes
    tree_arity: 0
  bittorrent:
    tracker_ip: 10.0.0.4
    download_timeout: 1800
  mkfs:
    - fstype: ext2
      args: -b 4096 -O sparse_super,filetype,resize_inode,dir_index
    - fstype: ext3
      args: -b 4096 -O sparse_super,filetype,resize_inode,dir_index
  tar: --warning=no-timestamp
  kastafor:
    binary: /usr/bin/kastafor
  kascade:
    binary: /usr/bin/kascade
    args: -v

```

2.1.2 Explanation of the fields used in the general configuration file

- /database
 - host *{String}*: hostname of the database
 - name *{String}*: name of the Kadeploy database
 - login *{String}*: login for the Kadeploy database
 - passwd *{String}*: password for the Kadeploy database
 - kind *{String}*: database kind (only mysql is available now).
- /rights
 - kind *{String}* (db): authentication kind (use db for a true rights management or dummy to bypass the rights management)
 - almighty_users *{String}* (root): list of users allowed to perform special operations on the environments like publishing environments or moving files
 - purge_deployment_timer *{Integer}* (900): limeout used to consider that a deployment is finished. This is used to avoid several deployment on the same nodes at the same time.
- /authentication/global Define global configuration for authentication methods.
 - headers_prefix *{String}* (X-Kadeploy-): Clients have to provide authentication information through HTTP headers, this is the prefix for each information kind (sample of authentication header information with the default setting: X-Kadeploy-User, X-Kadeploy-Certificate)
- /authentication/acl Access Control List based authentication. When using ACL authentication, specifying a whitelist is mandatory.
- /authentication/certificate authentication based on the certificate of a Certification Authority (the username must be provided in the CN subject's field), at least a public key or a x509 certificate as to be specified.
 - ca_cert *{String}*: the path to a file containing the x509 certificate of the Certification Authority.
- /authentication/certificate/ca_public_key
 - algorithm *{String}*: the algorithm that was used to generate the public key (expected values are *RSA*, *DSA* or *EC*).
 - file *{String}*: the path to a file containing the public key of the Certification Authority.
- /authentication/http_basic authentication using the HTTP Basic Authentication method (see RFC 2617).
 - dbfile *{String}*: The file containing a database of user/passwords (his file can be generated using tools such as *htpasswd*).

- realm *{String}* (SERVERS_URL): Overwrite the realm value with a custom one (for sample, if the service is accessed through a proxy).
- /authentication/ident authentication using the Ident protocol (see RFC 1413) on the client machine. When using Ident authentication, specifying a whitelist is mandatory.
- /authentication/*/[whitelist] For each authentication method, a whitelist can be specified. Authentication attempts will only be allowed from the specified hosts. The whitelist is an array of hosts (Strings). Hosts can be specified by IP address, IP address with CIDR notation, hostname and Regular Expression (using the char / as prefix and suffix of the expression).
- /security
 - secure_server *{Boolean}* (true): launch the server in secure (SSL) mode
 - local_only *{Boolean}* (false): only listen for local connection (/authentication/*/whitelist fields become useless)
 - certificate *{String}* (""): path to an x509 certificate that will be used to launch secure connections. If none is specified and the secure mode is enabled, a self-signed certificate will be generated.
 - force_secure_client *{Boolean}* (false): specify if files have to be exported to the server using a secured connection (see section 4.4.2).
- /security/private_key the private key associated with the certificate of the server. If none is specified and the secure mode is enabled, a new one will be generated.
 - algorithm *{String}*: the algorithm that was used to generate the public key (expected values are *RSA*, *DSA* or *EC*).
 - file *{String}*: the path to a file containing the private key.
- ssh_private_key *{String}* (/etc/kadeploy3/keys/id_deploy): specify private key loaded in ssh-agent.
- /logs
 - logfile *{String}* (""): path of a file that will contain the log information. If you do not wish to use a log file, do not set this field.
 - database *{Boolean}* (true): use the Kadeploy database to export the log information.
 - debugfile *{String}* (""): path of a file that will contain the log information. If you do not wish to use a debug file, do not set this field.
- /verbosity
 - clients *{Integer}* (3): number between 0 and 5 that specifies the default verbose level for the client. 0 means “no verbose” and 5 means “full verbose”.
 - logs *{Integer}* (3): debug level of the output exported to Syslog.
- /cache

- directory *{String}* (/tmp): absolute path of the Kadeploy cache. The cache dir is used to store the files of a user in a deployment.
- size *{Integer}*: size (MB) of the Kadeploy cache.
- concurrency_level *{Integer}*: Number of concurrent threads that can write to the cache simultaneously. Default value is 10.
- /network
 - server_hostname *{String}* (ruby Socket.gethostname): hostname of the Kadeploy server
 - tcp_buffer_size *{Integer}* (8192): TCP buffer size (Bytes) for the Kadeploy file server
- /network/vlan
 - hostname_suffix *{String}* (""): this specifies the suffix to add to the hostname to define the hostname in the given VLAN. The pattern VLAN_ID can be used in the definition, it is replaced at the runtime.
 - set_cmd *{String}* (""): command to launch in order to put a set of nodes in a VLAN. The patterns NODES, USER and VLAN_ID can be used.
- /network/ports
 - ssh *{Integer}* (22): port used by SSH
 - kadeploy_server *{Integer}* (25300): port of the Kadeploy server
 - test_deploy_env *{Integer}* (25300): port used as a tag in the deployment environment to ensure that the deployment environment is successfully booted
- /windows/check
 - size *{Integer}* (22): size of the nodes check window.
- /windows/reboot
 - size *{Integer}* (50): global size of the reboot window (ie. maximum number of nodes able to reboot at the same time). This might be useful to avoid high electricity peak.
 - sleep_time *{Integer}* (10): time to wait if the reboot window is full
- /environments
 - allowed_name_regex *{String}* (A): regex that control how an environnement should be name.
 - max_preinstall_size *{Integer}* (20): maximum size (MB) of the preinstall files
 - max_postinstall_size *{Integer}* (20): maximum size (MB) of the postinstall files
- /environments/deployment
 - extraction_dir *{String}* (/mnt/dest): extraction directory for the tarball in the deployment environment

- `tarball_dir {String}` (`/tmp`): destination directory for the tarball download in the deployment environment. This is used when the tarballs are sent with Bittorrent.
- `rambin_dir {String}` (`/rambin`): path of the ramdisk directory in the deployment environment
- `/pxe/dhcp` the default method used to PXE boot. Further information are available in the paragraph Booting over the network.
 - `method {String}` (PXElinux): the PXE method used to boot over the network (expected values are PXElinux, LPXElinux, IPXE or GrubPXE)
 - `repository {String}`: absolute path of the repository where PXE files are accessibles (TFTP, HTTP, ...). Warning, as far as the Kadeploy server is launched by the deploy user, deploy must have the rights to write in this directory.
- `/pxe/dhcp/export`
 - `kind {String}` (tftp): The method used to export PXE files (expected values are *tftp*, *http*, *ftp* and *auto*). The path to the files in the PXE profiles depends on this method. *auto* doesn't change the path in the PXE profiles, so the method of export results from the DHCP configuration.
 - `server {String}` (hostname): The server where PXE files are stored. To be complicant with most NBPs, it's recommended to specify this server by IP address (it will also make the nodes boot faster since there is no need to make a DNS request).
- `/pxe/dhcp/profiles`
 - `directory {String}` (`""`): The directory where PXE profiles have to be written. This path is relative to the PXE repository path unless you specify an absolute path. If the pathname is empty it defaults to the value of `/pxe/dhcp/repository`.
For example, with PXElinux, this directory is *pxelinux.cfg*.
 - `filename {String}`: The way to name the file of each node's profile (expected values are: *hostname*, *hostname_short* (the hostname without the domain name), *ip*, *ip_hex* (hexadecimal representation of the IP)).
The information used to generate this filenames are the one specified for each nodes in the clusters configuration file (see section 2.3). For example, with PXElinux, it will be *ip_hex*.
- `/pxe/dhcp/userfiles` PXE user custom files (option `--upload-pxe-files`). **Be careful**, this directory is emptied at each server launch.
 - `directory {String}`: The directory where PXE user custom files (option `--upload-pxe-files`) are to be saved. This path is relative to the PXE repository path.
 - `max_size {Integer}`: maximal size (MB) of the PXE user custom files sub-directory.
 - `concurrency_level {Integer}`: Maximal number of threads that can write to the PXE user custom files sub-directory simultaneously. Default value is 10.

- `/pxe/networkboot` the method used to boot operating system images sent from the network (the deployment environment kernel). This setting is optional by default, the DHCP method is used.
 - method `{String}`: the PXE method used to boot the nodes (expected values are PX-Elinux, LPXElinux, IPXE or GrubPXE)
 - binary `{String}`: the binary of the Network Bootstrap Program (if this method is different than the DHCP one, this file will be loaded by the PXE method). For example, for PXElinux, this file is `pxelinux.0`.
 - repository `{String}`: absolute path of the repository where PXE files (deployment environments kernels) are accessibles. Warning, as far as the Kadeploy server is launched by the deploy user, deploy must have the rights to write in this directory.
- `/pxe/networkboot/export`
 - kind `{String}`: The method used to export PXE files (expected values are `tftp`, `http` and `ftp`). The path to the files in the profiles will be generated depending on this method.
 - server `{String}`: The server where PXE files are stored. To be complicant with most NBPs, it's recommended to specify this server by IP address (it will also make the nodes boot faster since there is no need to make a DNS request).
- `/pxe/networkboot/profiles`
 - directory `{String}` (`"`): The directory where PXE profiles have to be written. This path is relative to the PXE repository path unless you specify an absolute path. If the pathname is empty it defaults to the value of `/pxe/networkboot/repository`. For example, with PXElinux, this directory is `pxelinux.cfg`.
 - filename `{String}`: The way to name the file of each node's profile (expected values are: `hostname`, `hostname_short` (the hostname without the domain name), `ip`, `ip_hex` (hexadecimal representation of the IP)).
The information used to generate this filenames are the one specified for each nodes in the clusters configuration file (see section 2.3). For example, with PXElinux, it will be `ip_hex`.
- `/pxe/localboot` the method used to boot an operating system that's installed on nodes hard disk. This setting is optional by default, the DHCP method is used.
 - method `{String}`: the PXE method used to boot the nodes (expected values are PX-Elinux, LPXElinux, IPXE or GrubPXE)
 - binary `{String}`: the binary of the Network Bootstrap Program (if this method is different than the DHCP one, this file will be loaded by the PXE method). For example, for PXElinux, this file is `pxelinux.0`.
 - repository `{String}`: absolute path of the repository where PXE files are accessibles (TFTP, HTTP, ...). Warning, as far as the Kadeploy server is launched by the deploy user, deploy must have the rights to write in this directory.
- `/pxe/localboot/export`

- kind *{String}*: The method used to export PXE files (expected values are *tftp*, *http* and *ftp*). The path to the files in the profiles will be generated depending on this method.
 - server *{String}*: The server where PXE files are stored. To be compliant with most NBPs, it's recommended to specify this server by IP address (it will also make the nodes boot faster since there is no need to make a DNS request).
- /pxe/localboot/profiles
 - directory *{String}* (""): The directory where PXE profiles have to be written. This path is relative to the PXE repository path unless you specify an absolute path. If the pathname is empty it defaults to the value of /pxe/localboot/repository.
For example, with PXELinux, this directory is *pxelinux.cfg*.
 - filename *{String}*: The way to name the file of each node's profile (expected values are: *hostname*, *hostname_short* (the hostname without the domain name), *ip*, *ip_hex* (hexadecimal representation of the IP)).
The information used to generate this filenames are the one specified for each nodes in the clusters configuration file (see section 2.3). For example, with PXELinux, it will be *ip_hex*.
 - /hooks
 - end_of_reboot *{String}* (""): command to launch at the end of an asynchronous reboot. The REBOOT_ID can be used in the command, it is replaced at the runtime.
 - end_of_power *{String}* (""): command to launch at the end of an asynchronous power operation. The POWER_ID can be used in the command, it is replaced at the runtime.
 - end_of_deployment *{String}* (""): command to launch at the end of an asynchronous deployment. The WORKFLOW_ID can be used in the command, it is replaced at the runtime.
 - autoclean_threshold *{Fixnum}* (360): at the end of an operation (deploy/reboot/power) it's status kept until the user explicitly deletes them. This value fix the maximal time (in minutes) this information will be kept in memory by the server until the autoclean loop delete them.
 - /external/default_connector *{String}* (ssh -A -l root -q -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null -o PreferredAuthentications=publickey -o BatchMode=yes): define alias for DEFAULT_CONNECTOR in Taktuk connector and remoteops for cluster specific file.
 - /external/taktuk
 - connector *{String}* (DEFAULT_CONNECTOR): connector used by Taktuk
 - tree_arity *{Integer}* (0): Taktuk tree arity for command executed through a tree. Use 0 if you want to use the work stealing algorithm of Taktuk and thus a dynamic tree arity. Use another value >0 to specify a static tree arity (should be avoided).
 - auto_propagate *{Boolean}* (true): use of the auto propagation feature of Taktuk. You should use this feature if the deployment environment doesn't contain Taktuk.

- `outputs_size` {*Integer*} (20000): to avoid big Taktuk outputs to be loaded in the server's memory, it's possible to setup a limit of the per-node output size. If TakTuk returns an output bigger than (`NODES_NUMBER * outputs_size`) an error will be returned. This limit can be disabled by setting the 0 value.
- `/external/bittorrent`
 - `tracker_ip` {*String*} (nil): ip of the Bittorrent tracker
 - `download_timeout` {*Integer*} (nil): timeout for the Bittorrent file download
- `/external/[mkfs]` Options for mkfs. The options for several FS can be defined here.
 - `fstype` {*String*} (nil): the filesystem type
 - `args` {*String*} (nil): the specific options for this filesystem type
- `tar` {*String*} (""): Options for tar (used in the deployment environment)
- `/external/kastafior`
 - `binary` {*String*} (kastafior): the command used to launch kastafior
- `/external/kascade`
 - `binary` {*String*} (kascade): the command used to launch kascade
 - `args` {*String*} (""): extra options for the kascade command

2.2 Booting over the network

In the `/pxe` configuration field, you can define how the nodes are booting from the network.

As far as you are using Kadeploy3 on your cluster, a PXE boot have to be setup on your nodes, so that implies that your nodes have a compatible Network Interface Card and that your DHCP server is configured a specific way. There is a lot of different software (Network Bootstrap Programs or NBP) that can be use to make nodes boot over the network: PXELinux, LPXELinux, iPXE, Etherboot, Grub disks,

For most of Network Bootstrap Programs, the booting method is the same:

1. The Network Iterface Card ask an IP address to the DHCP server;
2. The DHCP answer and give an extra instruction that specifiers to download and run a specific software, the Network Bootstrap Program (for example, with PXELinux, this file is `pxelinux.0`)
3. The NBP download a boot profile on the network that specifies how to boot the node. The profile file has a specific name (the hostname of the node, it's IP address, ...) so that the software will be sure to download the profile of this specific node.
4. The NBP read the profile and boot the node according to it's instructions (download and boot a specific kernel, boot on node's local hard disk, ...)

To be able to control node's boot, Kadeploy3 is editing this profiles. That's why it's necessary to tell it what NBP is used to boot your nodes over the network and how you want their profiles to be written.

Some of this softwares gives you the choice of the method used to download the files you want to boot (TFTP/HTTP/FTP/...), i.e. how you want your PXE files to be exported. You'll be able to specify how you want your files to be download in Kadeploy3 configuration.

Kadeploy will boot the nodes three different ways:

- (1) Booting a minimal kernel that was downloaded from the network (done in the first macrostep (SetupDeploymentEnv) to boot on the deployment environment);
- (2) Booting a kernel located on a partition of the local hard disk (done in the third macrostep (BootNewEnv) to boot the installed system);
- (3) Booting with some user's custom files with a user specified profile (done when the user is using the options *-x* and *-w*).

In the field `/pxe/dhcp` of the global configuration file, you will specify which NBP is setup on your DHCP server and give some information about the exports and profiles. This settings will be used every times Kadeploy3 will reboot your nodes unless you defile the fields `/pxe/networkboot` or `/pxe/localboot`.

If you define the field `/pxe/networkboot`, it will overwrite the settings `/pxe/dhcp` when Kadeploy3 will make a reboot kind (1).

If you define the field `/pxe/localboot`, it will overwrite the settings `/pxe/dhcp` when Kadeploy3 will make a reboot kind (2).

When the method (NBP) specified in `/pxe/networkboot` or `/pxe/localboot` is different than the method specified `/pxe/dhcp`, Kadeploy3 make the NBPs chainload each other. For instance, if `/pxe/dhcp` is set to boot with PXELinux and `/pxe/networkboot` with a GRUB disk *grubpxe.0*, Kadeploy3 will make PXELinux load *grubpxe.0* by writting *PXE grubpxe.0* in the PXELinux profile. Then it writes the GRUB profile (that will be loaded by *grubpxe.0*), that profile will make the node download and boot a kernel from the network.

One important thing is that every NBP have to be available in you PXE (TFTP in most of the cases) repository. Another one is that the Kadeploy user should be able (have the rights) to write the profiles inside the PXE repository.

Some scripts are provided in the distribution to help to generate a GRUB NBP that'll download profiles on the network (in the directory `addons/grubpxe/`).

2.3 Clusters file

This file describes the list of all the clusters, the location of their specific setting files and their nodes. All the nodes of the clusters that aim to be deployed must be declared in this file. It must be defined in the `/etc/kadeploy3/clusters.conf` file.

Warning, a cluster-specific configuration file and some partitioning/bootloader_install scripts must be defined for each cluster define in this file.

2.3.1 Example of a clusters file

```
---
clusters:
- name: graphene
  prefix: gra
  conf_file: graphene-cluster.conf
  nodes:
    - name: graphene-1.nancy.grid5000.fr # Full version
      address: 10.0.66.1
    - name: graphene-2.nancy.grid5000.fr
      address: 10.0.66.2
    - name: graphene-3.nancy.grid5000.fr
      address: 10.0.66.3
    - name: graphene-4.nancy.grid5000.fr
      address: 10.0.66.4
- name: griffon
  conf_file: griffon-cluster.conf
  nodes:
    - name: griffon-[1-92].nancy.grid5000.fr # Digest version
      address: 10.0.65.[1-92]
```

2.3.2 Explanation of the fields used in the clusters file

- /clusters
 - name *{String}*: the name of the cluster
 - prefix *{String}*: the prefix that will be used for display purpose when deploying nodes from several clusters (if not set, the prefix will be a unique integer identifier)
 - conf_file *{String}*: the path to the cluster-specific configuration file of this cluster (see section 2.4.2)
- /clusters/[nodes]
 - name *{String}*: the hostname of the node(s). Ranges can also be used to define hostnames: griffon-[1-92].nancy.grid5000.fr.
 - address *{String}*: the IP address of the node(s). Ranges can also be used to define addresses: 10.0.65.[1-92] .

2.4 Cluster-specific configuration files

To define the specific configuration of a cluster, you must create a specific file for each cluster in the configuration directory. The name of the file must be `specific_conf_CLUSTER` where `CLUSTER` is the cluster name.

2.4.1 Example of a cluster-specific configuration file

```

---
partitioning:
  script: parted_sample
  formatting_script: formatting.sh
  deploy_label: DEPLOY
  block_device: disk0
  disk_path:
    disk0: /dev/disk/by-path/pci-0000:00:0d.0-ata-1
trusted_deployment:
  user: deploy
  partition: "/dev/sda3"
  envs:
    - name: "debian"
      user: "deploy"
      version: 1578454
kexec:
  server_precmd: "true"
  script: kexec.sh
  repository: /dev/shm/kexec_repository
boot:
  install_bootloader: install_grub2
  kernels:
    user:
      params: console=tty0 console=ttyS1,38400n8
    deploy:
      vmlinuz: deploy-vmlinuz-2.6.27.8-bt
      initrd: deploy-initrd-2.6.27.8-bt
      params: console=tty0 console=ttyS0,38400n8 ramdisk_size=260000 rw
      supported_fs: ext2, ext3, vfat
      drivers: ata_piix,ata_generic
    nfsroot:
      vmlinuz: deploy-vmlinuz-2.6.27.7-nfsroot
      params: rw console=tty0 root=/dev/nfs ip=dhcp nfsroot=10.0.100.35:/mnt/nfsroot/rootfs
timeouts:
  reboot: 200 + 150 * Math.log(n)
  kexec: 60
localops:
  broadcastenv:
    cmd: /usr/bin/taktuk -c "TAKTUK_CONNECTOR" -f NODEFILE broadcast exec [ DECOMPRESS ]\; broadcast input file [ E
remoteops:
  reboot:
    - name: soft
      cmd: |-
        ssh -q \
        -o BatchMode=yes -o StrictHostKeyChecking=no \
        -o PreferredAuthentications=publickey \
        -o ConnectTimeout=2 -o UserKnownHostsFile=/dev/null \
        root@HOSTNAME_FQDN

```

```

- name: hard
  cmd: /usr/local/kadeploy/bin/hard_reboot.rb HOSTNAME_SHORT
- name: very_hard
  cmd: /usr/local/kadeploy/bin/reboot_RSA.exp HOSTNAME_SHORT
power_on:
# - name: soft
#   cmd: ...
- name: hard
  cmd: /usr/bin/lanpower -c on -m HOSTNAME_SHORT
  name: hard
# - name: very_hard
#   cmd: ...
power_off:
- name: soft
  cmd: |-
    ssh -q -o BatchMode=yes -o StrictHostKeyChecking=no -o \
    PreferredAuthentications=publickey -o ConnectTimeout=2 \
    -o UserKnownHostsFile=/dev/null \
    root@HOSTNAME_FQDN \
    "nohup /sbin/halt &>/dev/null &"
- name: hard
  cmd: /usr/bin/lanpower -c off -m HOSTNAME_SHORT
# - name: very_hard
#   cmd: ...
power_status:
- name: soft
  cmd: /usr/bin/lanpower -m HOSTNAME_FQDN -s
console:
- name: soft
  cmd: /usr/local/conman/bin/conman -d conman HOSTNAME_SHORT
preinstall:
files:
- file: /g5k/admin_pre_install.tgz
  format: tgz
  script: launch.sh
postinstall:
files:
- file: /g5k/admin_post_install.tgz
  format: tgz
  script: launch.sh
pxe:
headers:
dhcp: &id001 |-
  PROMPT 1
  SERIAL 0 38400
  TIMEOUT 50
netboot: *id001
localboot: set timeout=5
hooks:
use_ip_to_deploy: true

```

```

automata:
  macrosteps:
    SetDeploymentEnv:
      - type: Prod
        timeout: 200
        microsteps:
          - name: reboot
            timeout: 10
          - name: create_partition_table
            substitute:
              - action: run
                name: my_partitioning
                file: partitioning.sh
                retries: 1
                scattering: tree
                timeout: 16
          - type: Untrusted
            timeout: 400
    BroadcastEnv:
      - type: Custom
        timeout: 300
      - type: Kastafior
        retries: 1
        timeout: 900
        microsteps:
          - name: send_environment
            post-ops:
              - action: send
                file: hostname
                destination: $KADEPLOY_ENV_EXTRACTION_DIR/etc/
                retries: 1
                scattering: chain
              - action: exec
                command: mkdir -p $KADEPLOY_ENV_EXTRACTION_DIR/mypath
              - action: send
                file: myfile
                destination: $KADEPLOY_ENV_EXTRACTION_DIR/mypath
                timeout: 10
    BootNewEnv:
      - type: Kexec
        timeout: 100
      # - type: Classical
      #   retries: 1
      #   timeout: 200
      - type: HardReboot
        retries: 1
        timeout: 300

```

2.4.2 Explanation of the fields used in the cluster-specific configuration file

- /partitioning
 - block_device *{String}*: default disk used on the nodes
 - deploy_label *{String}*: default label used on the nodes
 - disable_swap *{Boolean}* (false): disable the swap partition on the disk
 - script *{String}*: Path to a script that creates the partition table on the nodes. You can use Kadeploy3 environment variables (see section 4.4.1) in this script. Please refer to section 2.6) for further informations.
 - formatting_script *{String}*: Path to a script that is used to format the partitions. You can use Kadeploy3 environment variables (see section 4.4.1) in this script. Please refer to section 2.7) for further informations.
 - disk_path *{Hash}*: Map the disks name to the real path on the nodes.
- /trusted_deployment
 - user *{String}*: user to trust if they made the last deployment
 - user *{String}*: partition to trust if the last deployment was made on it
- /trusted_deployment/envs
 - name *{String}*: name of the environment to trust
 - user *{String}*: author of the last environment to trust
 - version *{Integer}*: version of the last environment to trust, default is all version
- /kexec
 - server_precmd *{String}*: command to execute on the server before a kexec. Don't kexec from a trusted environment if the command return non-zero exit code.
 - script *{String}*: Path to a script that do the kexec. You can use Kadeploy3 environment variables (see section 4.4.1) in this script.
 - repository *{String}* (/dev/shm/kexec_repository): the directory in the running system where deploy kernel files have to be copied for kexec purpose
- /boot
 - sleep_time_before_boot *{Integer}* (20): seconds of sleep before the first ping request to detect when a node is ready. Generally, it represents the average time during the node power up until it sends a dhcp request.
 - install_bootloader *{String}*: Path to a script that install a bootloader on the deployment partition. You can use Kadeploy3 environment variables (see section 4.4.1) in this script. Please refer to section 2.5) for further informations.
- /boot/kernels Options of OS kernel's that are booted by Kadeploy3

- `/boot/kernels/user` Default options for user kernels
 - `params {String} ("")`: default kernel parameters applied to a Linux based deployed environment. This can be overloaded in the environment description.
- `/boot/kernels/deploy` The deployment environment (see section ??)
 - `vmlinux {String}`: name of the kernel file of the deployment environment. This file will be specified in the PXE profiles using the `pxe/networkboot/export` and `pxe/networkboot/repository` settings of the general configuration file.
 - `initrd {String}`: name of the initrd file of the deployment environment. This file will be specified in the PXE profiles using the `pxe/networkboot/export` and `pxe/networkboot/repository` settings of the general configuration file.
 - `params {String} ("")`: boot parameters of the deployment environment kernel
 - `supported_fs {String} (ext2,ext3,ext4,vfat)`: list of file systems that are supported by the deployment environment. The syntax is: `fstype1,fstype2,fstype3,...` . When deploying an environment with a non supported filesystem type, the deployment workflow will be modified (see section 2.4.2).
 - `drivers {String} ("")`: list of drivers that must be loaded in the deployment environment. The syntax is: `driver1,driver2,driver3,...`
- `/boot/kernels/nfsroot` Used when booting with NFS-root in the `SetDeploymentEnv` macro-step.
 - `vmlinux {String} ("")`: kernel for the NFS-root deployment environment (only used if you use an NFS-root deployment environment)
 - `params {String} ("")`: kernel parameters for the NFS-root deployment environment (only used if you use an NFS-root deployment environment)
- `/timeouts`
 - `reboot {Integer/String} (120)`: classical reboot timeout. A Ruby expression can be used here to represent a function depending on n (the number of nodes currently rebooted)
 - `kexec {Integer/String} (60)`: kexec reboot timeout. A Ruby expression can be used here to represent a function depending on n (the number of nodes currently rebooted)
- `/localops/broadcastenv` A custom command that will be used when sending an environment in a `BroadcastEnv` macro-step of kind `Custom`. Be careful, this command will be launched from the `Kadeploy3` server.
 - `cmd {String}`: The command that will send the environment file. You can use the `ENVFILE`, `NODEFILE`, `TAKTUK_CONNECTOR` and `DECOMPRESS` patterns in the command-line. The pattern `DECOMPRESS` will be replaced by a command that can be used to decompress the file while receiving it (if not specify, the decompress parameter should be set to false). The pattern `TAKTUK_CONNECTOR` will be replaced by the `TakTuk` connector (see section 2.1).

- decompress *{Boolean}* (true): If the script decompress the file while receiving it, it should be set to *true*. If the script don't, an extra micro-step will be add to the deployment process in order to decompress the file after it has been sent.
- level_name *{String Array}* (

"soft", "hard", "veryhard"

): defines level name by priority order.
- /remoteops/[reboot] The reboot commands, an escalation of them will be performed in the order of the List. Warning: at the moment, only the names *soft*, *hard* and *very_hard* can be used.
 - name *{String}*: the name of the command (used in the display)
 - cmd *{String}*: generic reboot command. You can use the HOSTNAME_FQDN and HOSTNAME_SHORT variables in the command-line.
 - group *{String}* (nil): the affinity between nodes (see section 2.4.2)
- /remoteops/[power_on] The power_on commands, an escalation of them will be performed in the order of the List. This commands are not mandatory. Warning: at the moment, only the names *soft*, *hard* and *very_hard* can be used.
 - name *{String}*: the name of the command (used in the display)
 - cmd *{String}*: generic power_on command. You can use the HOSTNAME_FQDN and HOSTNAME_SHORT variables in the command-line.
 - group *{String}* (nil): the affinity between nodes (see section 2.4.2)
- /remoteops/[power_off] The power_off commands, an escalation of them will be performed in the order of the List. This commands are not mandatory. Warning: at the moment, only the names *soft*, *hard* and *very_hard* can be used.
 - name *{String}*: the name of the command (used in the display)
 - cmd *{String}*: generic power_off command. You can use the HOSTNAME_FQDN and HOSTNAME_SHORT variables in the command-line.
 - group *{String}* (nil): the affinity between nodes (see section 2.4.2)
- /remoteops/[power_status] The power_status commands, an escalation of them will be performed in the order of the List. This commands are not mandatory. This commands are not mandatory. Warning: at the moment, only the name *soft* can be used.
 - name *{String}*: the name of the command (used in the display)
 - cmd *{String}*: generic power_status command. You can use the HOSTNAME_FQDN and HOSTNAME_SHORT variables in the command-line.
- /remoteops/[console] The console commands, an escalation of them will be performed in the order of the List. This commands are not mandatory. This commands are not mandatory. Warning: at the moment, only the name *soft* can be used.

- name *{String}*: the name of the command (used in the display)
- cmd *{String}*: generic console command. You can use the `HOSTNAME_FQDN` and `HOSTNAME_SHORT` variables in the command-line.
- /preinstall/[files] list of pre-install to execute at the pre-install of a deployment. This fields are not mandatory.
 - file *{String}*: the absolute path to the archive containing the scripts
 - format *{String}*: the kind of file (expected values are *tgz*, *tbz2*, *tzstd* or *txz*)
 - script *{String}*: the relative path (inside of the archive) to the script to be executed. The *none* value can be if no script must be launched. For debug purpose, you can use the keyword *breakpoint* instead of a script. Thus, the file will be transferred, the deployment workflow will be stopped and you will be able to connect in the deployment environment to debug.
- /postinstall/[files] list of post-install to execute at the post-install of a deployment. This fields are not mandatory.
 - file *{String}*: the absolute path to the archive containing the scripts
 - format *{String}*: the kind of file (expected values are *tgz*, *tbz2*, *tzstd* or *txz*)
 - script *{String}*: the relative path (inside of the archive) to the script to be executed. The *none* value can be if no script must be launched. For debug purpose, you can use the keyword *breakpoint* instead of a script. Thus, the file will be transferred, the deployment workflow will be stopped and you will be able to connect in the deployment environment to debug.
- /pxe/headers PXE headers to be used for the different kind of reboots
 - dhcp *{String}* (""): PXE headers to be used for the the default method. Further information are available in the paragraph Booting over the network.
 - networkboot *{String}* (""): PXE headers to be used when booting operating system images sent from the network. Further information are available in the paragraph Booting over the network.
 - localboot *{String}* (""): PXE headers to be used when booting operating system from hard disk. Further information are available in the paragraph Booting over the network.
- /hooks
 - use_ip_to_deploy *{Boolean}* (false): use IP addresses instead of hostnames to contact the nodes
- /automata/macrosteps list of implementations for each macro-steps of the automata. There are 3 macro-steps, so you must specify each of them.
- /automata/macrosteps/[SetDeploymentEnv] the macro-step in charge of rebooting the nodes on the deployment environment

- type *{String}*: the type of the macro-step (expected values are *Untrusted*, *Kexec*, *Prod*, *Nfsroot*, *UntrustedCustomPreInstall* and *Dummy*)
- retries *{Integer}* (0): the number of retries for this macro-step (by default, one single attempt with no retries)
- timeout *{Integer}*: the timeout (seconds) of this macro-step
- /automata/macrosteps/[SetDeploymentEnv]/[microsteps] Microsteps specific configuration (this field is not mandatory)
 - name *{String}*: the name of the micro-step, see the list bellow to get the different micro-steps names.
 - timeout *{Integer}* (0): the timeout (seconds) of this micro-step
 - retries *{Integer}* (0): the number of retries for this micro-step. Since most of micro-steps perform some modifications on the running system and are do not perform any cleaning operation before their execution, be very careful when using this setting.
- /automata/macrosteps/[SetDeploymentEnv]/[microsteps]/[substitute] Substitute this micro-step with some custom operations (see the paragraph bellow for custom operations description)
- /automata/macrosteps/[SetDeploymentEnv]/[microsteps]/[pre-ops] A list of operations that have to be done before executing the micro-step (see the paragraph bellow for custom operation description)
- /automata/macrosteps/[SetDeploymentEnv]/[microsteps]/[post-ops] A list of custom operations that have to be done after executing the micro-step (see the paragraph bellow for custom operation description)
- /automata/macrosteps/[BroadcastEnv] the macro-step in charge of broadcasting the image of the user's environment image on the nodes
 - type *{String}*: the type of the macro-step (expected values are *Kastafior*, *Chain*, *Tree*, *Bittorrent* and *Dummy*)
 - retries *{Integer}* (0): the number of retries for this macro-step
 - timeout *{Integer}*: the timeout (seconds) of this macro-step (0 for no timeout)
- /automata/macrosteps/[BroadcastEnv]/[microsteps] Microsteps specific configuration (this field is not mandatory)
 - name *{String}*: the name of the micro-step, see the list bellow to get the different micro-steps names.
 - timeout *{Integer}* (0): the timeout (seconds) of this micro-step
 - retries *{Integer}* (0): the number of retries for this micro-step. Since most of micro-steps perform some modifications on the running system and are do not perform any cleaning operation before their execution, be very careful when using this setting.
- /automata/macrosteps/[BroadcastEnv]/[microsteps]/[substitute] Substitute this micro-step with some custom operations (see the paragraph bellow for custom operations description)

- `/automata/macrosteps/[BroadcastEnv]/[microsteps]/[pre-ops]` A list of operations that have to be done before executing the micro-step (see the paragraph below for custom operation description)
- `/automata/macrosteps/[BroadcastEnv]/[microsteps]/[post-ops]` A list of custom operations that have to be done after executing the micro-step (see the paragraph below for custom operation description)
- `/automata/macrosteps/[BootNewEnv]` the macro-step in charge of rebooting the nodes after the installation of the environment
 - type `{String}`: the type of the macro-step (expected values are *Classical*, *Kexec*, *HardReboot*, *PivotRoot* (not implemented yet) and *Dummy*)
 - retries `{Integer}` (0): the number of retries for this macro-step
 - timeout `{Integer}`: the timeout (seconds) of this macro-step
- `/automata/macrosteps/[BootNewEnv]/[microsteps]` Microsteps specific configuration (this field is not mandatory)
 - name `{String}`: the name of the micro-step, see the list below to get the different micro-steps names.
 - timeout `{Integer}` (0): the timeout (seconds) of this micro-step
 - retries `{Integer}` (0): the number of retries for this micro-step. Since most of micro-steps perform some modifications on the running system and are do not perform any cleaning operation before their execution, be very careful when using this setting.
- `/automata/macrosteps/[BootNewEnv]/[microsteps]/[substitute]` Substitute this micro-step with some custom operations (see the paragraph below for custom operations description)
- `/automata/macrosteps/[BootNewEnv]/[microsteps]/[pre-ops]` A list of operations that have to be done before executing the micro-step (see the paragraph below for custom operation description)
- `/automata/macrosteps/[BootNewEnv]/[microsteps]/[post-ops]` A list of custom operations that have to be done after executing the micro-step (see the paragraph below for custom operation description)

Custom operations

With custom operations you can send files or execute commands.

Here is a custom operation description:

- name `{String}`: The name of the custom operation
- action `{String}`: The action that have to be performed (expected values are *send*, *run* and *exec*)
- file `{String}`: (To be specified if the action is *send* or *run*) The path to the file to be send/executed (if the action is *send* the file name will remains the same, if the action is *run* this file need to contain a script)

- destination `{String}`: (To be specified if the action is *send*) The destination directory on the nodes (Kadeploy3 environment variables are substituted in the path)
- params `{String}:""` (To be specified if the action is *run*) The parameters of the script.
- command `{String}`: (To be specified if the action is *exec*) The command to be executed. If you want to call a script, dont forget to add a `.` (or use *source*) before the script name to be able to use Kadeploy3 environment variables inside of it (example: command: `./myscript.sh`).
- timeout `{Integer}` (0): the timeout (seconds) of this custom operation
- retries `{Integer}` (0): the number of retries for this custom operation
- scattering `{String}` ('tree'): The scattering kind for this custom operation (expected values are *tree* and *chain*)

The automata macro-steps

Here is the list of the macro-step and their implementation:

- SetDeploymentEnv
 - SetDeploymentEnvProd
 - * check_nodes
 - * format_deploy_part
 - * mount_deploy_part [only with non-fsa/dd and supported fs]
 - * format_tmp_part [only if non-multipart]
 - SetDeploymentEnvUntrusted
 - * switch_pxe
 - * reboot
 - * wait_reboot
 - * send_key_in_deploy_env
 - * create_partition_table
 - * format_deploy_part [only if supported fs]
 - * mount_deploy_part [only if non-fsa/dd and supported fs]
 - * format_tmp_part [only if non-multipart]
 - * format_swap_part [only if non-multipart]
 - SetDeploymentEnvKexec
 - * send_deployment_kernel
 - * kexec
 - * wait_reboot
 - * send_key_in_deploy_env
 - * create_partition_table
 - * format_deploy_part [only if supported fs]
 - * mount_deploy_part [only if non-fsa/dd and supported fs]

- * format_tmp_part [only if non-multipart]
 - * format_swap_part [only if non-multipart]
 - SetDeploymentTrusted

Check if the environment is trusted based on the cluster configuration if so, use SetDeploymentEnvKexec else use SetDeploymentEnvUntrusted.
 - SetDeploymentEnvUntrustedCustomPreInstall
 - * switch_pxe
 - * reboot
 - * wait_reboot
 - * send_key_in_deploy_env
 - * manage_admin_pre_install
 - SetDeploymentEnvNfsroot
 - * switch_pxe
 - * reboot
 - * wait_reboot
 - * send_key_in_deploy_env
 - * create_partition_table
 - * format_deploy_part [only if supported fs]
 - * mount_deploy_part [only if non-fsa/dd and supported fs]
 - * format_tmp_part [only if non-multipart]
 - SetDeploymentEnvDummy
- BroadcastEnv
 - BroadcastEnvChain
 - * send_environment(“chain”)
 - * decompress_environment [only with fsa]
 - * mount_deploy_part [only with fsa/dd and supported fs]
 - * manage_admin_post_install [only if supported fs]
 - * manage_user_post_install [only if supported fs]
 - * check_kernel_files [only if supported fs]
 - * send_key [only if supported fs]
 - * install_bootloader [only if supported fs]
 - BroadcastEnvKastafior
 - * send_environment(“kastafior”)
 - * decompress_environment [only with fsa]
 - * mount_deploy_part [only with fsa/dd and supported fs]
 - * manage_admin_post_install [only if supported fs]
 - * manage_user_post_install [only if supported fs]
 - * check_kernel_files [only if supported fs]

- * send_key [only if supported fs]
 - * install_bootloader [only if supported fs]
- BroadcastEnvTree
 - * send_environment(“tree”) [only if supported fs]
 - * decompress_environment [only with fsa]
 - * mount_deploy_part [only with fsa/dd and supported fs]
 - * manage_admin_post_install [only if supported fs]
 - * manage_user_post_install [only if supported fs]
 - * check_kernel_files [only if supported fs]
 - * send_key [only if supported fs]
 - * install_bootloader [only if supported fs]
- BroadcastEnvBittorrent
 - * send_environment(“bittorrent”)
 - * decompress_environment [only with fsa]
 - * mount_deploy_part [only with fsa/dd and supported fs]
 - * manage_admin_post_install [only if supported fs]
 - * manage_user_post_install [only if supported fs]
 - * check_kernel_files [only if supported fs]
 - * send_key [only if supported fs]
 - * install_bootloader [only if supported fs]
- BroadcastEnvCustom
 - * send_environment(“custom”)
 - * decompress_environment [only with fsa or if no custom decompress]
 - * mount_deploy_part [only with fsa/dd and supported fs]
 - * manage_admin_post_install [only if supported fs]
 - * manage_user_post_install [only if supported fs]
 - * check_kernel_files [only if supported fs]
 - * send_key [only if supported fs]
 - * install_bootloader [only if supported fs]
- BroadcastEnvDummy
- BootNewEnv
 - BootNewEnvClassical
 - * switch_pxe
 - * umount_deploy_part [only if supported fs]
 - * reboot_from_deploy_env
 - * wait_reboot
 - BootNewEnvKexec [only if supported fs and linux]

- * switch_pxe
- * umount_deploy_part
- * mount_deploy_part
- * kexec
- * wait_reboot
- BootNewEnvHardReboot
 - * switch_pxe
 - * reboot(“hard”)
 - * wait_reboot
- BootNewEnvDummy

Note about the reboot/power-on/power-off commands In some special cases, a such command can affect a group of nodes. Thus, you can specify group of nodes for a given command using the following syntax :

```
---
# ...
remoteops:
  reboot:
    - name: very_hard
      cmd: /usr/sbin/very_hard_power_off GROUP_SHORT
      group: path_to_group_of_node_for_hard_power_off_cmd
# ...
```

You can remark two things :

- the HOSTNAME_SHORT and HOSTNAME_FQDN patterns are not used in these commands, instead you must use the GROUP_SHORT and GROUP_FQDN patterns.
- the affinity between nodes is specified in a file (here path_to_group_of_node_for_hard_power_off_cmd) that contains as much lines as the number of groups. Then, each line contains the nodes of a group, for instance: node1,node2,node3.

For a given command on a given cluster, if you specify some group of nodes (with GROUP_SHORT or GROUP_FQDN patterns), you will also be able to specify, for some nodes of the cluster, a command that does not imply a group of nodes. To do this, you must specify these commands in the specific commands configuration files.

2.5 Bootloader install script

A script that installs a bootloader on the nodes must be provided for each cluster.

You can use the Kadeploy3 environment variables in your script (see section 4.4.1). The tools you can use in your script are the ones that are installed in your deployment environment (see section 2.9).

Be careful to install the bootloader on the deployment partition (Kadeploy3 environment variable KADEPLOY_DEPLOY_PART) in order for kadeploy to be able to chainload on it.

Examples of scripts are provided in the distribution (in the directory scripts/bootloader/).

2.6 Partitioning script

A partitioning script must be provided for each cluster.

You can use the Kadeploy3 environment variables in your script (see section 4.4.1). The tools you can use in your script are the ones that are installed in your deployment environment (see section 2.9).

The script must write GPT label on each partition use by Kadeploy3. The label must be prefixed by KDPL_ and end by the disk name (which is given by the KADEPLOY_DISK_NAME environment variable). For exemple, KDPL_SWAP_disk0 is a valid label. The SWAP and TMP label are used to identify the swap and tmp partition. The default deploy partition is specify in the cluster configuration of Kadeploy3.

You should also use the */sbin/partprobe* at the end of your script to inform the OS (deployment environment) of partition table changes.

Examples of scripts are provided in the distribution (in the directory scripts/partitioning/).

2.7 Formating script

A formating script can be provided for each cluster. You can use the Kadeploy3 environment variables in your script (see section 4.4.1).

The same script is used for all the formating operations (swap, tmp and deploy partitions). The KADEPLOY_FORMAT_PART environment variable allow you to distinguish each operation.

2.8 Specific commands configuration files

In the part 2.4.2 we saw that generic commands can be given to all the nodes that belong to a cluster. It is also possible to override these generic values for some specific nodes. To do this, you must fill the file named *command.conf* in the configuration directory.

Note: it is not mandatory to override all the commands for a given node.

2.8.1 Example of a commands file

```
---
vm-001:
  reboot_soft: ssh -q root@vm-001 /sbin/special_reboot_for_vm
  reboot_hard: vmware-cmd /home/vmware/vm-001/vm-001.vmx reset hard
vm-002:
  reboot_soft: ssh -q root@vm-002 /sbin/special_reboot_for_vm
```

2.8.2 Explanation of the fields used in the commands file

- /NODENAME The nodes are specified by hostname (as declared in the clusters configuration file)

- `COMMAND {String}`: the setting to override and the command

The COMMAND can should be:

- `reboot_soft` to override `/remoteops/reboot/[]/name=soft`
- `reboot_hard` to override `/remoteops/reboot/[]/name=hard`
- `reboot_very_hard` to override `/remoteops/reboot/[]/name=very_hard`
- `power_on_soft` to override `/remoteops/power_on/[]/name=soft`
- `power_on_hard` to override `/remoteops/power_on/[]/name=hard`
- `power_on_very_hard` to override `/remoteops/power_on/[]/name=very_hard`
- `power_off_soft` to override `/remoteops/power_off/[]/name=soft`
- `power_off_hard` to override `/remoteops/power_off/[]/name=hard`
- `power_off_very_hard` to override `/remoteops/power_off/[]/name=very_hard`
- `power_status` to override `/remoteops/power_status/[]/name=soft`
- `console` to override `/remoteops/console/[]/name=soft`

2.9 Deployment environment

There are three ways to set a deployment environment: using the production environment, using a dedicated environment, using an NFSRoot environment.

2.9.1 Configuration of the production environment

TODO

2.9.2 Creation of the dedicated environment

Debian: Debirf based method (recommended)

This methods consists in creating a kernel/initrd that contains all the tools required to perform a deployment. The *debirf* software (<http://cmrg.fifthhorseman.net/wiki/debirf>) is to ease the creation of the deployment environment. To use these method, go to the `adons/deploy_env_generation/debirf` directory and execute with root rights:

```
> make all
```

The `kadeploy-deploy-kernel/debirf.conf` configuration file can be tuned if you want to add or remove some packages in the filesystem. To do this, you can modify the `INCLUDE` and `EXCLUDE` values.

You can also add custom debirf modules in the `kadeploy-deploy-kernel/modules/` directory. You can find a module example named `blacklist_example` in this directory. Please refer to debirf documentation for further information.

Once you've executed the `make` command, you can find the kernel/initrd files of the deployment environment in the directory `kadeploy-deploy-kernel/`.

Debian: Debootstrap based method

This methods consists in creating a kernel/initrd that contains all the tools required to perform a deployment. Two scripts are provided to ease the creation of the deployment environment. To use these scripts, go to the addons/deploy_env_generation/debootstrap directory and execute with root rights:

```
> sh make_debootstrap.sh
> sh make_kernel.sh
```

The make_debootstrap.sh script can be tuned if you want to add or remove some packages in the filesystem. To do this, you can modify the DEBOOTSTRAP_INCLUDE_PACKAGES and DEBOOTSTRAP_EXCLUDE_PACKAGES values.

The make_kernel.sh script prompts the user the following things:

- the size of the uncompressed initrd in KB;
- the kernel version;
- the absolute path to a kernel config file;
- the use of automatic configuration for the new fields in kernel configuration.

The size of the uncompressed initrd depends on what you have to put in your deployment environment. If you use the make_debootstrap.sh script, the initrd size should be at least 200MB. Depending on the kernel version you choose, the script will fetch the vanilla kernel corresponding to this version. Once a kernel has been fetched, it won't be fetched again in another run. Thus, you have to delete the kernel file if you want to fetch it again. At the opposite, if you do not want to use the sources of the vanilla kernel but your own sources, you can put your own kernel (tar.bz2 compressed) in the current directory. The only requirement is to name the file with the following pattern: linux-version.tar.bz2. Then, at the kernel version prompt, just enter the *version* value.

After the execution of make_kernel.sh, a directory prefixed with built- will be created. This directory contains the kernel and the initrd files, prefixed with deploy-.

Centos: Kadeploy provided scripts

This methods consists in creating a kernel/initrd that contains all the tools required to perform a deployment. A custom ruby script can be used to ease the creation of the deployment environment. To use these method, go to the addons/deploy_env_generation/centirf directory and execute with root rights:

```
> make all
```

Once you've executed the make command, you can find the kernel/initrd files of the deployment environment in the directory centirf/.

2.9.3 Creation of the NFSRoot environment

TODO

2.10 Configuration of the deploy user

In order to use the Kastafor based file broadcaster (`BroadcastEnvKastafor` macro-step), the server must be able to perform an ssh connection on itself. Thus, you must add the deploy key installed in the `/etc/kadeploy3/keys/id_deploy.pub` in the `.ssh/authorized_keys` file of the deploy user. This step is optional if you do not plan to use the `BroadcastEnvKastafor` macro-step.

2.11 Configuration of SSH-agent

It is possible to make the Kadeploy server load an SSH-agent at launch time. This can be helpful to use SSH functionalities such as the SSH-agent forwarding to communicate with the nodes. For sample this functionality can be used in the TakTuk connector (`/external/taktuk/connector`, section 2.1) or in the reboot and power operations (`/remoteops`, section 2.4.2).

To enable this fonctionnality, the SSH private key to be used with the SSH-agent must be present as `keys/id_deploy` of the server configuration directory (see section 2). By default (if the file `/etc/kadeploy3/keys/id_deploy` does not exist), no agent is loaded at launch time.

Chapter 3

Client side configuration

On the client side, you only have to configure the file named `client.conf`. This file defines Kadeploy servers and a default server.

Example of a client configuration file

```
---
default: nancy
servers:
- name: lille
  hostname: frontend.lille.grid5000.fr
  port: 25300
  auth_headers_prefix: X-Kadeploy-
- name: nancy
  hostname: nancy.lille.grid5000.fr
  port: 25300
  secure: true
```

Explanation of the fields

- `/default {String}`: the default Kadeploy server to use (name should be included in the list of `/[servers]/name`)
- `/[servers]` The different servers
 - `name {String}`: the Kadeploy server name
 - `hostname {String}`: the Kadeploy server hostname
 - `port {Integer}`: the port the Kadeploy server is listening on
 - `secure {Boolean}` (true): specify if the server use a secure connection

- `auth_headers_prefix {String} ("")`: The client provides authentication information to the server through HTTP headers, this is the prefix for each information kind (sample of authentication header information with the default setting: `X-Kadeploy-User`). This setting depends on the server's configuration. When not specified (empty), the client do an extra GET HTTP request on the server to determine this value.

Chapter 4

User guide

4.1 Overview of the Kadeploy tools

4.1.1 Kadeploy

The Kadeploy tool is base on a client/server architecture. Thus, it is composed both of a server part and a client part. The server must be run with the root rights and the client is used with standard rights.

4.1.2 Kareboot

Kareboot is designed to perform several reboot operations on the nodes.

4.1.3 Kaenv

Kaenv is designed to manage the users environments.

4.1.4 Kaconsole

Kaconsole is designed to provide a user to access to the consoles of the nodes on which the user has the deployment rights.

4.1.5 Kastat

Kastat is designed to show several statistics about the deployments.

4.1.6 Kanodes

Kanodes is designed to show the state of the nodes.

4.1.7 Kapower

Kapower is designed to control the power state of the nodes.

4.1.8 Karights

Karights is designed to allow users to perform some deployments on a set of nodes throughout a reservation. This tool is typically called by the resource manager at the prologue and epilogue steps.

4.2 Use the Kadeploy tools

4.2.1 Kadeploy server

All the Kadeploy tools use the Kadeploy server. On a well configured system, the Kadeploy server can be launched with the following command (with root rights):

```
> kadeploy3d
```

4.2.2 Kadeploy client

The Kadeploy client is actually the user interface for the Kadeploy software. It can be used by using the kadeploy3 command. The CLI looks like this:

```
> kadeploy3 -h
__HELP_kadeploy3_HELP__
```

At least, Kadeploy must be called with one node and an environment. The nodes to deploy can be specified by using several `-m|--machine` options, or the `-f|--file` options (one node per line in the file), or a mix of both. The environment can be specified with the `-e|--env-name` option if you want to use an environment recorded in the environment database or with the `-a|--env-file` options if you want to use an environment described in a file. Refer to the 4.2.4 part for information about the environment description. Here are some examples:

```
> kadeploy3 -m gdx-5.orsay.grid5000.fr -e lenny-x64-nfs-1.0 -o nodes_ok -n nodes_ko
> kadeploy3 -m gdx-[5-12].orsay.grid5000.fr -e lenny-x64-base -o nodes_ok -n nodes_ko
> kadeploy3 -f nodes -a custom_env.dsc
> kadeploy3 -f nodes -m gdx-5.orsay.grid5000.fr -a custom_env.dsc
> cat nodefile|kadeploy3 -f - -e lenny-x64-base
```

We present now several use cases.

Use case 1 - basic usage - deployment of a node

```
> kadeploy3 -m gdx-5.orsay.grid5000.fr \
    -e lenny-x64-nfs-1.0 \
    --verbose-level 5 \
    -k ~/.ssh/id_rsa.pub
```

This command performs the deployment of the environment *lenny-x64-nfs-1.0* on the node *gdx-5.orsay.grid5000.fr* and copies the SSH public key *~/.ssh/id_rsa.pub* of the user in the deployed environment to allow a direct connection with the root account. Furthermore, the verbose level is set to 5, which means that you want the maximum verbose information.

Use case 2 - basic usage - deployment of a range of nodes

```
> kadeploy3 -m gdx-[45-51].orsay.grid5000.fr \
-e lenny-x64-base \
-k
```

This command performs the deployment of the environment *lenny-x64-base* on the nodes *gdx-45.orsay.grid5000.fr*, *gdx-46.orsay.grid5000.fr*, ..., *gdx-51.orsay.grid5000.fr*. Furthermore, it copies the entries of the `./ssh/authorized_keys` user file in the `/root/.ssh/authorized_keys` of the deployed nodes.

Use case 3 - basic usage - deployment of a set of nodes

```
> kadeploy3 -f ~/machinefile \
-e custom_env \
-l johnsmith \
-o nodes_ok -n nodes_ko
```

This command uses the environment *custom_env* of the user *johnsmith* to deploy the nodes specified in */machinefile*. The list of the nodes correctly deployed will be written in the file specified with the `-o|--output-ok-nodes` option. Idem for the nodes not correctly deployed with the `-n|--output-ko-nodes` option. Refer to the part 4.2.4 about Kaenv to know more about the environment management.

Use case 4 - basic usage - execution of a script after deployment

```
> kadeploy3 -f $OAR_NODE_FILE \
-a ~/my-lenny.dsc \
-r ext3 \
-p 4 \
-s ~/launcher.sh
```

This command performs the deployment of the environment described by the file */my-lenny.dsc* (useful if you don't want to share your environment with the other users) on the nodes specified in the file pointed by `$OAR_NODE_FILE` (typically a variable set by the resource manager). We specify here that we want the `/tmp` partition to be reformatted. Furthermore, we specify that we want to deploy the environment on the 4th disk partition, instead of the default one. Finally, we ask to execute the script */launcher.sh* at the end of the deployment.

Use case 5 - advanced usage - play with breakpoint

```
> kadeploy3 -m gdx-5.orsay.grid5000.fr \
-e lenny-x64-nfs-1.0 \
--verbose-level 5 \
--breakpoint BroadcastEnvKastafior:manage_user_post_install \
-d
```

This kind of command can be used for debug purpose. It performs a deployment with the maximum verbose level and it asks to stop the deployment workflow just before executing the *manage_user_post_install* micro-step of the *BroadcastEnvKastafior* macro-step. Thus you will be able to connect in the deployment environment and to debug what you want. Furthermore, the full output of the distant commands performed is shown.

Use case 6 - advanced usage - specific PXE profile

```
> kadeploy3 -m gdx-[5-10].orsay.grid5000.fr \
    -e lenny-x64-nfs-1.0 \
    -w ~/pxe_profile -x "~/custom-kernel,~/custom-initrd" \
    --set-pxe-pattern ~/singularities
```

In some specific case, you may want to use a specific PXE profile to boot your nodes. To do this, you have to provide a PXE profile. Warning, the files used in your PXE profile (Comboot, kernel, initrd, ...) must be readable by the TFTP server on the Kadeploy server. So Kadeploy offers a feature to stage some files in an area where the files can be read by the TFTP server. This can be achieved with the `-x|--upload-pxe-files` option. You must know that such uploaded files will be copied in the `tftp_images_path`. Those files will then be available with the prefix `FILES_PREFIX--`.

Here is an example of PXE profile that uses uploaded files:

PROMPT 1

SERIAL 0 38400

DEFAULT bootlabel

DISPLAY messages

TIMEOUT 50

label bootlabel

KERNEL FILES_PREFIX--custom-kernel

APPEND initrd=FILES_PREFIX--custom-initrd root=/dev/sda3 node_id=NODE_SINGULARITY

In this example, `FILES_PREFIX--` will be replaced by the prefix added to each files sent into the TFTP repository via the `-x|--upload-pxe-files` option (be careful not to forget the `--` suffix).

You can notice the `NODE_SINGULARITY` pattern used in the PXE profile. Thanks to the `--set-pxe-pattern` option, you can also provide a file that defines a value in the PXE profile that depends on the node concerned. This file must define on each line a couple of value as follows : `hostname,node singularity`. In our example, the file `/singularities` can contains something like:

```
gdx-5.orsay.grid5000.fr,1
gdx-6.orsay.grid5000.fr,2
gdx-7.orsay.grid5000.fr,3
gdx-8.orsay.grid5000.fr,3
gdx-9.orsay.grid5000.fr,4
gdx-10.orsay.grid5000.fr,5
```

Use case 7 - advanced usage - specific bootloader requirement

```
> kadeploy3 -m gdx-5.orsay.grid5000.fr \
    -e Custom_linux_env \
    --disable-bootloader-install
```

If you deploy a Linux based environment and if the administrator choose to boot the nodes with the *chainload* fashion, Kadeploy will install automatically a bootloader on the deployment partition. In some cases, you may want to bypass this installation because you have installed at the time

of a previous deployment another bootloader. This allows to avoid the overriding of the installed bootloader. However, if no bootloader is installed or if the installed bootloader is not able to boot your environment, the won't be reachable at the end of the deployment.

Use case 8 - advanced usage - get a workflow id for an external deployment tracking

```
> kadeploy3 -m gdx-5.orsay.grid5000.fr \
    -e Custom_linux_env \
    --write-workflow-id wid_file
```

This command performs the deployment of the *Custom_linux_env* environment and write the workflow id of this deployment in the file *wid_file*. The aim of getting the deployment id is to monitor the deployment from an extern tool thanks to the Kanodes tool.

Use case 9 - expert usage - modify the deployment workflow

```
> kadeploy3 -m gdx-5.orsay.grid5000.fr \
    -e "FreeBSD 7.1" \
    --force-steps "SetDeploymentEnv|SetDeploymentEnvProd:2:100&
                  BroadcastEnv|BroadcastEnvKastafior:2:300&
                  BootNewEnv|BootNewEnvKexec:1:150"
```

If you are a power user, you can specify the full Kadeploy workflow and bypass the default configuration. Use it at your own risk since the nodes may not support all the Kadeploy features like the *Kexec* optimization for instance. The syntax for the `--force-steps` option is the same that for the `/automata/macrosteps` field if the Kadeploy configuration. The difference is that the three macrostep are defined on the same line, with the `&` character as a delimiter between the macrosteps. Warning, you must define at least one implementation for each macro-step, without newline (unlike the example).

Use case 10 - expert usage - insert custom operations in the deployment workflow

```
> kadeploy3 -m griffon-1.nancy.grid5000.fr \
    -e squeeze-x64-base \
    --set-custom-operations ~/custom_ops.yml
```

For very specific purpose, you can add some custom operations in the deployment workflow. To do this, you have to specify these operations in a YAML file where you can specify the operations that must be executed before/after/instead a micro-step.

Here is a description of the YAML file:

- `/MacroStepName` The name of the target macro-step (see section 2.4.2 for a list of allowed macro-steps)
- `/MacroStepName/MicroStepName` The name of the target micro-step (see section 2.4.2 for a list of allowed micro-steps)
 - `override {Boolean}` (false): Override custom steps that have been defined in the cluster configuration.

- `/MicroStepName/MicroStepName/[substitute]` Substitute this micro-step with some custom operations (see section 2.4.2 for custom operations description)
- `/MicroStepName/MicroStepName/[pre-ops]` A list of operations that have to be done before executing the micro-step (see section 2.4.2 for custom operation description)
- `/MicroStepName/MicroStepName/[post-ops]` A list of custom operations that have to be done after executing the micro-step (see section 2.4.2 for custom operation description)

When you are executing a command or a script (via the `exec` or `run` action), you can use the `Kadeploy3` environment variables (see section 4.4.1).

Note: This variables will also be substituted in your destination directory specification when you are using the `send` action.

Here is an example of a file that contains custom operations:

```
> cat ~/custom_ops.yml
SetDeploymentEnvUntrusted:
  create_partition_table:
    substitute:
      - action: run
        name: my_partitioning
        file: partitioning.sh
        timeout: 16
        scattering: tree
BroadcastEnvKastafior:
  send_environment:
    post-ops:
      - action: exec
        command: echo 'net.ipv4.ip_forward = 1' >> $KADEPLOY_ENV_EXTRACTION_DIR/etc/sysctl.conf
      - action: exec
        command: mkdir -p $KADEPLOY_ENV_EXTRACTION_DIR/mypath
      - action: send
        file: my_custom_file
        destination: $KADEPLOY_ENV_EXTRACTION_DIR/mypath
        retries: 1
        timeout: 24
        scattering: chain
```

Use case 11 - expert usage - deploying a dd image on a block device

```
> kadeploy3 -m griffon-1.nancy.grid5000.fr \
  -e ddgz_fulldisk_image \
  -b /dev/sda \
  -c 1
```

For some specific purpose, you may want to deploy a dd image of an entire -partitioned- disk. To do this, you first have to create a ddgz image of the disk you want to deploy. Then you have to specify on which block device you want your image to be deployed with the `-b` option. You also need to tell on which partition the chainloaded reboot have to be performed with the `-c` option (typically a partition where a bootloader was installed, use 0 if the bootloader is installed on the MBR).

4.2.3 Kareboot

Kareboot can be used by using the `kareboot3` command. The CLI looks like this:

```
> kareboot3 -h
__HELP_kareboot3_HELP__
```

At least, Kareboot must be called with one node and a reboot kind. The nodes to reboot can be specified by using several `-m|--machine` options, or the `-f|--file` options (one node per line in the file), or a mix of both. The expected values for the `-r|--reboot-kind` are:

- `simple_reboot`: perform a simple reboot of the nodes. Kareboot firstly tries to perform a soft reboot, then a hard reboot is performed and lastly a very hard reboot if it doesn't success before.
- `set_pxe`: modify the PXE profile with the one given with the `-w|--set-pxe-profile` options and perform a simple reboot.
- `env_recorded`: perform a reboot on an environment that is already deployed (for instance, the production environment on the production part). This operation must be used with the `-e` and `-p` options at least.
- `deploy_env`: perform a reboot on the deployment environment. This can be used with the `-k|--key` option.

Here are some basic examples:

```
> kareboot3 -m gdx-5.orsay.grid5000.fr -r simple_reboot
> kareboot3 -m gdx-[5-8].orsay.grid5000.fr -r simple_reboot
> cat nodefile|kareboot3 -f - -r simple_reboot
> kareboot3 -m gdx-5.orsay.grid5000.fr -r simple_reboot -o reboot_ok.txt \
    -n reboot_ko.txt
> kareboot3 -f nodes -r set_pxe -w ~/customized_pxe_profile
> kareboot3 -f nodes -r set_pxe -w ~/customized_pxe_profile -l hard \
    -x "~/custom_kernel,~/custom_initrd" \
    --set-pxe-pattern ~/singularities (Cf. Kadeploy use case 6)
> kareboot3 -f nodes -r deploy_env -k .ssh/id_rsa
> kareboot3 -r env_recorded -e production_environment \
    -p 2 -u root -m gdx-5.orsay.grid5000.fr
> kareboot3 -r env_recorded -e production_environment \
    -p 2 -u root -m gdx-5.orsay.grid5000.fr \
    --no-wait
```

Kareboot can be used to manage the destructive environment. Typically, at the end of a reservation with deployment, the resource manager will perform a reboot on the production environment. By using the `-c|--check-destructive-tag` option (for instance: `kareboot -f nodes -r env_recorded -c`), Kareboot firstly checks if the deployed environment on the involved nodes is tagged like a destructive environment. If the environment is considered as *destructive*, Kareboot does not perform a reboot and returns the 2 value. In this case, the recorded environment has been destroyed and should be deployed again. If the environment is not considered as *destructive*, the reboot is performed.

If the nodes are correctly rebooted on the production environment, Kareboot returns the 0 value. Otherwise it returns the 1 value, what means that the recorded environment has been destroyed and that it should be deployed again.

In the Kaenv part you can find the way to remove the *destructive* tag of an environment.

Warning, if the `--no-wait` option is used, Kareboot won't wait the end of the reboot to exit. Thus, this option cannot be used with the `-o`, `--output-ok-nodes` and `-n`, `--output-ko-nodes` options.

4.2.4 Kaenv

Command line interface

Kaenv can be used by using the `kaenv3` command. The CLI looks like this:

```
> kaenv3 -h
__HELP__kaenv3_HELP__
```

We present now several use cases.

Use case 1 - list the environments

```
> kaenv3 -l
```

This command lists the environment that you have previously recorded, and the public environments.

Use case 2 - list the shared environments recorded by another user

```
> kaenv3 -l -u johnsmith -s
```

This command lists the environment of the user *johnsmith*. If you use “*” as a user value, it lists the environments of all the users. Furthermore, the `-s|--show-all-versions` option is used to show all the versions of each environment. If this option is not specified, only the version is displayed.

Use case 3 - print an environment

```
> kaenv3 -p FreeBSD --env-version 3 -u johnsmith
```

This command lists prints the version *3* of the environment *FreeBSD* that belongs to *johnsmith*. If no version number is given, the last version of the environment is printed. To print an environment you own, there is no need to use the `-u|--user` option.

Use case 4 - add an environment described in a file

```
> kaenv3 -a ~/new_env.dsc
```

This command adds the environment defined in the file */new_env.dsc*.

Use case 5 - add an environment described in an http file

```
> kaenv3 -a http://www.grid5000.fr/pub/johnsmith/env.desc
```

This command adds the environment defined in the file *http://www.grid5000.fr/pub/johnsmith/env.desc*.

Use case 6 - delete an environment

```
> kaenv3 -d FreeBSD --env-version 2
```

This command deletes the version *2* of the environment *FreeBSD* from the environment database. If no version number is given, all the versions are deleted.

Use case 7 - remove the destructive property of an environment

```
> kaenv3 --toggle-destructive-tag FreeBSD --env-version 3
```

This command toggle the *destructive* tag of the version *3* of the environment *FreeBSD*. If no version number is given, the latest version of the environment is considered.

Use case 8 - update the tarball of an environment

```
> kaenv3 --update-image-checksum sidx64-base
```

This command is useful if you modify the tarball of the environment *sidx64-base* without modifying the kernel or the initrd and if you do not want to record a new environment. Thus, it will update the MD5 of the tarball file. This operation is required if something change in the tarball, otherwise the environment will be unusable.

Use case 9 - update the postinstalls of an environment

```
> kaenv3 --update-postinstalls-checksum sidx64-base
```

This command does the same thing than the precedent one but it concerns the post-install files. This operation is required if something change in the post-install files, otherwise the environment will be unusable.

Use case 10 - define the visibility of an environment

```
> kaenv3 --set-visibility-tag sidx64-base --env-version 3 -t private
```

This command allows to define the environment *sidx64-base* version *3* as a private environment. Note that the environment version is required and only the almighty environment users are allowed to define an environment as public.

Environment description

The description of an environment is made with a YAML file (see section 2).

Here is an example of an environment description:

```
---
name: debian-xen
version: 3
description: https://www.grid5000.fr/index.php/Etch-x64-xen-1.0
author: John Smith
visibility: shared
image:
  file: /grid5000/debian-x64-xen-1.0.tgz
  kind: tar
  compression: gzip
preinstall:
  archive: /home/john/test/pre_install.tgz
  compression: gzip
  script: launch.sh
postinstalls:
- archive: /home/john/test/post_install1.tgz
  compression: gzip
  script: traitement.sh
- archive: /home/john/test/post_install2.tgz
  compression: gzip
  script: start.sh
boot:
  kernel: /boot/vmlinuz-2.6.18-6-xen-amd64
  kernel_params: console=tty0 console=ttyS1,38400n8
  initrd: /boot/initrd.img-2.6.18-6-xen-amd64
  hypervisor: /boot/xen-3.0.3-1-amd64.gz
  hypervisor_params: dom0_mem=1000000
partition_type: 0x83
filesystem: ext2
```

Another (shorter) example:

```
---
name: freebsd
version: 1
image:
  file: /grid5000/freebsd.ddgz
  kind: dd
  compression: gzip
```

Explanation of the fields used in the environment description:

- name *{String}*: name of the environment. The spaces are allowed in the name but remember to use some quotes around it when you use Kadeploy or Kaenv.
- version *{Integer}* (1): the version of the environment

- description `{String}` (`""`): the description of the environment
- author `{String}` (`""`): the author of the environment
- visibility `{String}` (`'private'`): define the visibility level of an environment. Three levels are available:
 - private: only the owner of the environment can see and use it ;
 - shared: the environment can be used by everybody but it must explicitly use with the owner name ; furthermore, it won't be listed unless the owner name is specified ;
 - public: the environment can be used by everybody and it is listed without specifying its owner name.
- destructive `{Boolean}` (`false`): specify that the environment is destructive
- multipart `{Boolean}` (`false`): specify that the environment is multi-partitioned. Be careful, with multi-partitioned environment, specific options have to be set ((Multi-partitioned) tagged ones).
- os `{String}`: kind of environment (expected values are linux, xen or other).
- /image the disk image of the environment
 - file `{String}`: the path to the disk image of the environment (can be local path or URL)
 - kind `{String}`: specify the kind of image (expected values are tar (a tarball archive of the environment), dd (a dd image of the environment) or fsa (a fsarchiver image of the environment, see section 4.4.5))
 - compression `{String/Integer}`: the compression of the disk image file (expected values are Strings gzip, xz, zstd or bzip2 and Integer [0..9] for FSA image)
- /preinstall a pre-installation script that will be executed before the environment is sent and installed (useless unless the kind of image is tar)
 - archive `{String}`: the path to the archive that contains the scripts (can be local path or URL)
 - compression `{String}`: the compression of archive (expected values are gzip or bzip2)
 - script `{String}` (`none`): the script to execute. For debug purpose, you can use the keyword breakpoint instead of a script. Thus, the file will be transferred, the deployment workflow will be stopped and you will be able to connect in the deployment environment to debug. Finally, the script value can be *none* if no script must be launched. Warning, if the preinstall field is fulfilled, the entire SetDeploymentEnv step defined by the administrator will be bypassed. Refer to the 4.4.3 part concerning build of a pre-install.
- /[postinstalls] some post-installation script that will be executed after the environment is sent and installed (useless unless the kind of image is tar). It will only be executed if the filesystem is readable/writable.
 - archive `{String}`: the path to the archive that contains the scripts (can be local path or URL)

- compression *{String}*: the compression of the archive (expected values are gzip or bzip2)
- script *{String}* (none): the script to execute. For debug purpose, you can use the keyword breakpoint instead of a script. Thus, the file will be transferred, the deployment workflow will be stopped and you will be able to connect in the deployment environment to debug. Finally, the script value can be *none* if no script must be launched. pre-install.
- /boot information about the system that will be installed (useless if the kind of image is dd)
 - kernel *{String}* (""): path of the kernel in the tarball.
 - kernel_params *{String}* (""): set of parameters that must be applied to the kernel for a correct boot
 - initrd *{String}* (""): path of the initrd in the tarball.
 - hypervisor *{String}* (""): path of the hypervisor in the tarball. This fields is only required for the Xen based environments
 - hypervisor_params *{String}* (""): set of parameters that must be applied to the hypervisor for a correct boot. This fields is only required for the Xen based environments
 - block_device *{String}*: (Multi-partitioned) specify the block_device that contains the partition to boot
 - partition *{Integer}*: (Multi-partitioned) specify the partition that contains the system to be booted
- partition_type *{Integer}* (0): the MS-DOS partition type, you can specify hexadecimal values using the prefix 0x. For example, 0x83 or 131 for Linux, 0xa5 for FreeBSD, ...
- filesystem *{String}* (""): type of filesystem wished on the deployment partition. It must be known by the mkfs command (useless unless the kind of image is tar)
- /options Extra options
- /options/[partitions] (Multi-partitioned) Information about the dispatching of the partitions included inside the image file on the partitions of the hard disk.
 - id *{Integer}*: (Multi-partitioned) the ID of the partition in the multi-partitioned image file
 - device *{String}*: (Multi-partitioned) the physical partition (for example: /dev/sda3) where the partition #id of the multi-partitioned image file has to be saved

4.2.5 Kaconsole

Kaconsole can be used by using the kaconsole3 command. It has only one use case that is opening a console on a given node, for instance:

```
> kaconsole3 -m gdx-25.orsay.grid5000.fr
```

Kaconsole can't be used on a node on which a user doesn't have the deployment rights. Furthermore, as soon as the deployments rights are revoked for a user, ever open console is automatically closed.

4.2.6 Kastat

Kastat can be used by using the `kastat3` command. The CLI looks like this:

```
> kastat3 -h
__HELP__kastat3_HELP__
```

We present now the use cases. Note that all the commands can be filtered with a period by using the `-x|--date-min` and `-y|--date-max` options.

Use case 1 - get the information about the deployments performed on a node

```
> kastat3 -d -m gdx-25.orsay.grid5000.fr
```

This command prints all the deployment performed on the node *gdx-25.orsay.grid5000.fr*.

Use case 2 - get the information about deployments performed on a range of node

```
> kastat3 -d -m gdx-[25-130].orsay.grid5000.fr
```

This command prints all the deployment performed on the nodes *gdx-25.orsay.grid5000.fr*, *gdx-26.orsay.grid5000.fr*, ..., *gdx-130.orsay.grid5000.fr*.

Use case 3 - print only a subset of the information about the deployments performed

```
> kastat3 -d -f hostname -f env -f success
```

This command prints all the deployment performed. Because the `-f|--field` option is used, only the fields *hostname*, *env*, and *success* are printed. If the option `-f|--field` is not used, all the fields are printed.

Use case 4 - print the failure rate about the nodes wrt the deployments that occurs between two dates

```
> kastat3 -b -x 2009:02:12:08:00:00 -y 2009:02:13:08:00:00
```

This command prints the failure rate of all the nodes (at least deployed one time) during the period between the 2009/02/12 - 8h00 and the 2009/02/13 - 8h00. The `-x|--date-min` and `-y|--date-max` options can be used separately or can be omitted.

Use case 5 - print the information about the nodes that have at least a given failure rate

```
> kastat3 -c 25 -x 2009:02:12:08:00:00
```

This command prints the nodes that have a failure rate of at least 25% from the 2009/02/12 - 8h00.

Use case 6 - print the information about the nodes that require several retries to deploy correctly

```
> kastat3 -a 3 -s 1
```

This command prints the information about the deployments that requires at least 3 retries in the macro-step 1. If the `-s|--step` option is not set, the information about the deployments that requires at least 3 retries in any macro-step are printed.

4.2.7 Kanodes

Kanodes can be used by using the `kanodes3` command. The CLI looks like this:

```
> kanodes3 -h
__HELP__kanodes3_HELP__
```

We present now the use cases.

Use case 1 - print the deployment state of the nodes

```
> kanodes3 -d
```

This command prints the global state of all the nodes managed by a Kadeploy server. The output is as follows 1,2,3,4,5,6, where :

- 1 is the hostname ;
- 2 is the deployment state of the node (`prod_env`, `deployed`, `deploy_failed`, `aborted`) ;
- 3 is the username who launched the last deployment ;
- 4 is the environment name ;
- 5 is the environment version ;
- 6 is the environment owner.

Use case 2 - print the deployment state of some nodes

```
> kanodes3 -d -m gdx-25.orsay.grid5000.fr -m netgdx-[1-30].orsay.grid5000.fr -f machine_file
```

This command prints the global state of the node `gdx-25.orsay.grid5000.fr` the nodes `netgdx-1.orsay.grid5000.fr`, `netgdx-2.orsay.grid5000.fr`, ..., `netgdx-30.orsay.grid5000.fr` and of the nodes listed in the file `machine_file`.

Use case 3 - get information about all the current deployment workflows

```
> kanodes3 -s
```

This command prints a YAML output of the deployment state of all the nodes currently in deployment. On the YAML output, the nodes are sorted according to the deployment workflow they belong to.

Use case 4 - get information about a specific deployment workflows

```
> kanodes3 -s -w 78
```

This command prints a YAML output of the deployment state of all the nodes currently in the deployment number 78. The deployment number, or workflow id, can be obtained thanks to a Kadeploy option.

4.2.8 Kapower

Kapower can be used by using the `kapower3` command. The CLI looks like this:

```
> kapower3 -h
__HELP__kapower3__HELP__
```

Use case 1 - print the power status of some nodes

```
> kapower3 --status -m gdx-[25-35].orsay.grid5000.fr -o nodes_up -n nodes_down
```

This command print the power status of the nodes *gdx-25.orsay.grid5000.fr* to *gdx-35.orsay.grid5000.fr*. Furthermore, the list of the powered up nodes is stored in `nodes_up` and the list of the powered off nodes is stored in `nodes_down`.

Use case 2 - power off some nodes

```
> kapower3 --off -f machine_file --server lille
```

This command powers off the nodes contained in the `machine_file` file. Since the `--server` is used, the nodes of a distant site are concerned by the operation ; in this example, the lille site is concerned.

Use case 3 - power on some nodes

```
> kapower3 --on -m gdx-25.orsay.grid5000.fr --no-wait
```

This command powers on the node *gdx-25.orsay.grid5000.fr* without waiting the end of the operation to return.

4.2.9 Karights

Karights can be used by using the `karights3` command (it is designed for administrators in order to allow users to perform deployments). The CLI looks like this:

```
> karights3 -h
__HELP__karights3__HELP__
```

We present now the use cases.

Use case 1 - give some rights to a user on a node

```
> karights3 -a -m gdx-25.orsay.grid5000.fr -p /dev/sda3 -u johnsmith
```

This command gives some rights for a given user.

Use case 1 - give some rights to a user on several nodes

```
> karights3 -a -m gdx-[25-32].orsay.grid5000.fr -p /dev/sda3 -u johnsmith
```

This command gives some rights for a given user on a range of nodes.

Use case 3 - give all the rights to a user on all the nodes

```
> karights3 -a -m "*" -p "*" -u root
```

This command gives all the rights on all the nodes to the user *root*.

Use case 4 - give some rights on a node and remove existing ones

```
> karights3 -a -m gdx-25.orsay.grid5000.fr -p /dev/sda3 -u johnsmith -o
```

This command gives some rights for a given user. Furthermore, if some rights (excepted those specified with ***) were previously given on the node *gdx-25.orsay.grid5000.fr*, they are deleted.

Use case 5 - remove som rights

```
> karights3 -d -m gdx-25.orsay.grid5000.fr -p /dev/sda3 -u johnsmith
```

This command removes some rights for a given user.

Use case 6 - show the rights of a user

```
> karights3 -s -u johnsmith
```

This command shows the rights given to user.

4.3 What you should know if you want to do kernel development on deployed nodes

Kernel development implies to know what Kadeploy do concerning the boot of the deployed environments.

4.3.1 Kadeploy 3 behavior

Kadeploy 3 has a different behavior depending on the kind of deployed environment. Reminder: the kind of environment is defined in the environment description.

Linux environments

On a *Linux* environment, Kadeploy 3 automatically installs the Grub 2 bootloader on the deployed partition once the tarball is broadcasted. Then it modifies the PXE profile of the concerned nodes in order to ask the chainload on the deployed partition. This is performed thanks to *pxelinux* and the *comboot chain.c32*.

Xen environments

On a *Xen* environment, Kadeploy 3 doesn't install the Grub 2 bootloader since Grub 2 there are some known issues when booting a Xen Dom0 with Grub 2. Thus Kadeploy 3 uses the old method that consists in booting the nodes in a pure PXE fashion. To do that, Kadeploy extracts the kernel, initrd and hypervisor files from the environment tarball and modifies the PXE profile of the concerned nodes in order to ask their in pure PXE. This is performed thanks to pxelinux and the comboot mboot.c32.

Other environments

On an *Other* environment, Kadeploy 3 assumes that a bootloader is already installed on the partition since a full partition image (dd.gz image) has been copied. Thus, it only modifies the PXE profile of the concerned nodes in order to ask the chainload on the deployed partition, like in the *Linux* case.

4.3.2 Tips to simply use your new kernel

If you do kernel development on the deployed nodes, you will probably want to update you kernel without recording a new image and redeploying it to save time, especially to perform small tests.

Linux environments

On a *Linux* environment, after having updated your kernel/initrd, 2 cases are imaginable:

1. your kernel/initrd have the same name, so you can reboot the node without modifying anything.
2. your kernel/initrd have a new name, so you will have to update the grub configuration file (/boot/grub/grub.cfg) of your node in order to allow grub to select the new kernel and then you can reboot the node.

Xen environments

On a *Xen* environment, the things are a little bit more complicated. As far as the kernel/initrd/hypervisor are extracted by Kadeploy in a dedicated cache, changing them on the deployed nodes won't have any effect for the next reboot. So you have to use a feature of Kareboot that allows to reboot a node after having changed the PXE profile of the node. For instance:

```
> kareboot3 -m gdx-25.orsay.grid5000.fr -r set_pxe -w ~/pxe_profile_xen \
-x "~/custom_kernel,~/custom_initrd,~/custom_hypervisor"
```

Kadeploy has the same feature, so please refer to the use case about *specific PXE profile* for more information.

Other environments

On an *Other* environment, you eventually have to update your bootloader in order to boot on the new kernel.

4.4 Extra

4.4.1 Kadeploy3 Environment variables

When writing a script for an admin pre-install, an admin/user post-install or some custom operations, you can use the following environment variables :

- `KADEPLOY_CLUSTER` : cluster on which the pre/post install is launched
- `KADEPLOY_ENV` : environment deployed
- `KADEPLOY_ENV_KERNEL` : the path to the kernel file inside the deployed environment directory
- `KADEPLOY_ENV_INITRD` : the path to the initrd file inside the deployed environment directory
- `KADEPLOY_ENV_KERNEL_PARAMS` : the deployed environment kernel parameters
- `KADEPLOY_ENV_HYPERVISOR` : the path to the hypervisor's kernel file inside the deployed environment directory (usefull when deploying a Xen environment for example)
- `KADEPLOY_ENV_HYPERVISOR_PARAMS` : the deployed environment hypervisor parameters
- `KADEPLOY_DEPLOY_PART` : path to the deployment partition
- `KADEPLOY_TMP_PART` : path to the tmp partition
- `KADEPLOY_SWAP_PART` : path to the swap partition
- `KADEPLOY_DISK_NAME` : the Kadeploy3 name of the disk used in deployment
- `KADEPLOY_BLOCK_DEVICE` : the path to the block device used in deployment
- `KADEPLOY_ENV_EXTRACTION_DIR` : path where the environment tarball is extracted
- `KADEPLOY_PREPOST_EXTRACTION_DIR` : path where the pre/post tarball are extracted
- `KADEPLOY_TMP_DIR` : a temporary directory (to be used for your scripts)
- `KADEPLOY_OS_KIND` : the kind of operating system being deployed (the value of the *environment_kind* field of the environment description)
- `KADEPLOY_PART_TYPE` : the MSDOS partition type of the deployment partition (the value of the *fdisk_type* field of the environment description)
- `KADEPLOY_FS_TYPE` : the filesystem format of the deployment partition (the value of the *filesystem* field of the environment description)
- `KADEPLOY_FS_TYPE_TMP` : the filesystem format of the tmp partition (if the tmp partition is reformat with the option `-reformat-tmp`)

- `KADEPLOY_KEXEC_KIND` : used for kexec script. Specify what kind of kexec is happening. value are *to_deployed_env* or *to_deploy_kernel*
- `KADEPLOY_FORMAT_PART` : used for formatting script. Specify what disk is being formatted. value are *DEPLOY,TMP* or *SWAP*
- `KADEPLOY_DEPLOY_ENV_KERNEL_PARAMS` : the deployment kernel kernel parameters
- `KADEPLOY_ENV_KEXEC_REPOSITORY` : the location where deploy kernel are copied for kexec purpose

4.4.2 Specifying files to the server

Files can be specified to the server using three different URI-based notations:

- `http://` The file is hosted on some HTTP server, sample: `http://testbed.lan/file.tgz`;
- `server://` The file locally hosted on the Kadeploy server, sample: `server:///tmp/file.tgz`;
- `local://` or no URI prefix The file is hosted on the client and will be exported to the server, sample: `local:///home/user/file.tgz`.

4.4.3 Build a custom pre-install

The goal of the pre-install in the Kadeploy workflow is to prepare the disk of the nodes before the copy of the environment. It can include:

- setting disk parameters (with `hdparm` for instance) ;
- partitioning the disk (with `fdisk` or `parted`) ;
- formatting the deployment and the `/tmp` partition ;
- mounting partition(s).

To setup a custom pre-install you first have to create an archive that contains your scripts. After that you have to tell kadeploy which script of your archive has to be executed, this is done by specifying the `preinstall` field in your environment description file (see section 4.2.4). Please be careful to use relative paths in your scripts since you don't know where they will be uncompressed.

You can do what you want in the pre-install but you must know that Kadeploy will extract the environment in the directory defined by the `/environments/deployment/extraction_dir` field of the general configuration file. Commonly, this directory is `/mnt/dest`. Thus, you have to mount all the partitions you need in this directory. If you wish to deploy the environment onto several partitions, you can use for instance the following map:

- `/dev/sda3` \mapsto `/mnt/dest`
- `/dev/sda4` \mapsto `/mnt/dest/var`
- `/dev/sda5` \mapsto `/mnt/dest/usr`

- `/dev/sda6 ↦ /mnt/dest/tmp`

If you choose to mount more than one partition in the pre-install, remember to umount all the partitions excepted the one mounted on `/environments/deployment/extraction_dir` (`/mnt/dest` in principle) in the post-install step. Indeed, the common Kadeploy workflow will automatically umount the partition mounted on `/environments/deployment/extraction_dir`. Thus, if other partitions are mounted, the umount will fail.

4.4.4 Do a custom partitioning

To perform a custom partitioning, you can use a substitute custom operation.

You can use the Kadeploy3 client's option `--set-custom-operations` (see 4.2.2) to setup custom micro-step operations.

The field to configure is the field `/SetDeploymentEnv/create_partition_table/[substitute]`.

Be careful to use the same partitioning scheme than the one which is configured by default on the server in order for the deployment process to perform properly.

If you want to change the partitioning scheme, you'll have to substitute the microsteps `format_deploy_part`, `mount_deploy_part`, `umount_deploy_part`, `format_swap_part` and `format_tmp_part` in order to mount and format the right partitions during the deployment.

Here is a example of the client command:

```
> kadeploy3 -m griffon-1.nancy.grid5000.fr \
  -e squeeze-x64-base \
  --set-custom-operations ~/custom_ops.yml
```

Here is a example of a the custom operations file:

```
> cat ~/custom_ops.yml
SetDeploymentEnvUntrusted:
  create_partition_table:
    substitute:
      - action: run
        name: my_custom_partitioning
        file: partitioning.sh
SendEnvironment:
  reboot:
    post-ops:
      - action: send
        file: hostname
        destination: $KADEPLOY_ENV_EXTRACTION_DIR/etc/
        retries: 1
        scattering: chain
```

Note: It is also possible for administrators to add systematic custom operations in the deployment process in order to perform a custom partitioning: the setting to modify in the cluster-specific configuration file (see section 2.4.2) is `/automata/macrosteps/[SetDeploymentEnv]/[microsteps]/format_deploy_part/[substitute]`. The custom operations to add will look the same as the ones created in the `custom_ops.yml` script.

4.4.5 Fsarchiver environnements

Fsarchiver allow to save several partition in one single file. It supports a bunch of different filesystems, it's also possible to specify the compression algorithm used to compress the archive.

A documentation about creating fsarchiver images is available on the project's website (<http://www.fsarchiver.org/>).

Note: Be careful to clean the system that will be booted in the image from node-specific files (for example, udev files on Linux systems).

Once the image is generated, it's possible to install it with Kadeploy3. In the environment file (see section 4.2.4), the type of the image file (field `/kind`) have to be set to `fsa`. If several partitions are saved in the fsarchiver image, the field `/multipart` need to be set to `true`, it's also necessary to specify where each partitions should be installed on the node (field `/options/[partitions]`) and the partition where the system to boot is locates (fields `/boot/block_device` and `/boot/partition`).

Note: The ID number of each partition (field `/options/[partitions]/id`) is affected depending on the order the partitions was saved in the fsarchiver file when it was generated.

Here is an example of an `fsa` environment description:

```
---
name: debian-min
version: 1
description: https://www.grid5000.fr/mediawiki/index.php/Squeeze-x64-base-1.0
author: John Smith
visibility: shared
destructive: true
multipart: true
os: linux
image:
  kind: fsa
  compression: 3
  file: /grid5000/debian-multipart.fsa
boot:
  kernel: /vmlinuz
  initrd: /initrd.img
  block_device: /dev/sda
  partition: 2
filesystem: ext3
partition_type: 0x83
options:
  partitions:
    - id: 0
      device: /dev/sda1
    - id: 1
      device: /dev/sda2
    - id: 2
      device: /dev/sda3
```