

---

# Lago Documentation

*Release 0.44.0*

**David Caro**

**Jun 06, 2018**



<b>1</b>	<b>Lago Introduction</b>	<b>1</b>
<b>2</b>	<b>Getting started</b>	<b>3</b>
2.1	Installing Lago	3
2.1.1	pip	3
2.1.2	RPM Based - Fedora 24+ / CentOS 7.3+	4
2.1.2.1	Install script	4
2.1.2.2	Manual RPM installation	5
2.1.3	FAQ	5
2.1.4	Troubleshooting	6
2.2	LagoInitFile Specification	6
2.2.1	Sections	6
2.2.1.1	domains	7
2.2.1.2	nets	8
2.3	Lago SDK	8
2.3.1	Starting an environment from the SDK	9
2.3.1.1	Prerequisites	9
2.3.1.2	Prepare the environment	9
2.3.1.3	Starting the environment	10
2.3.1.4	Controlling the environment	10
2.3.2	Differences from the CLI	11
2.4	Getting started with some Lago Examples!	11
2.4.1	Available Examples	11
2.5	Lago Templates	11
2.5.1	Available templates	12
2.5.2	Repository metadata	12
2.6	Configuration	12
2.6.1	lago.conf format	13
2.6.2	lago.conf look-up	13
2.6.3	Overriding parameters with environment variables	13
2.7	Lago build	14
2.7.1	Builders	14
2.7.1.1	virt-customize	14
2.7.2	Relation to bootstrap	15
2.7.3	Example	15
2.8	Lago CPU Models in detail	15

<b>3</b>	<b>Developing</b>	<b>17</b>
3.1	CI Process . . . . .	17
3.1.1	Starting a branch . . . . .	17
3.1.2	A clean commit history . . . . .	17
3.1.3	Rerunning the tests . . . . .	18
3.1.4	Asking for reviews . . . . .	18
3.1.5	Getting the pull request merged . . . . .	18
3.2	Environment setup . . . . .	18
3.2.1	Requirements . . . . .	19
3.2.2	Style formatting . . . . .	19
3.2.3	Testing your changes . . . . .	19
3.2.3.1	Setting up mock_runner.sh with mock (fedora) . . . . .	19
3.2.3.2	Running the tests inside mock . . . . .	20
3.3	Getting started developing . . . . .	20
3.3.1	Python! . . . . .	20
3.3.2	Bash . . . . .	21
3.3.3	Libvirt + qemu/kvm . . . . .	21
3.3.4	Git + Github . . . . .	21
3.3.5	Unit tests with py.test . . . . .	22
3.3.6	Functional tests with bats . . . . .	22
3.3.7	Packaging . . . . .	22
3.3.8	Where to go next . . . . .	22
<b>4</b>	<b>Contents</b>	<b>23</b>
4.1	lago package . . . . .	23
4.1.1	Subpackages . . . . .	23
4.1.1.1	lago.plugins package . . . . .	23
4.1.1.2	lago.providers package . . . . .	35
4.1.2	Submodules . . . . .	43
4.1.3	lago.brctl module . . . . .	43
4.1.4	lago.build module . . . . .	43
4.1.5	lago.cmd module . . . . .	46
4.1.6	lago.config module . . . . .	46
4.1.7	lago.constants module . . . . .	47
4.1.8	lago.export module . . . . .	48
4.1.9	lago.guestfs_tools module . . . . .	50
4.1.10	lago.lago_ansible module . . . . .	52
4.1.11	lago.log_utils module . . . . .	53
4.1.12	lago.paths module . . . . .	58
4.1.13	lago.prefix module . . . . .	59
4.1.14	lago.sdk module . . . . .	67
4.1.15	lago.sdk_utils module . . . . .	68
4.1.16	lago.service module . . . . .	69
4.1.17	lago.ssh module . . . . .	70
4.1.18	lago.subnet_lease module . . . . .	71
4.1.19	lago.sysprep module . . . . .	75
4.1.20	lago.templates module . . . . .	76
4.1.21	lago.utils module . . . . .	81
4.1.22	lago.validation module . . . . .	85
4.1.23	lago.virt module . . . . .	85
4.1.24	lago.vm module . . . . .	87
4.1.25	lago.workdir module . . . . .	88
<b>5</b>	<b>Releases</b>	<b>91</b>

5.1	Release process . . . . .	91
5.1.1	Versioning . . . . .	91
5.1.2	RPM Versioning . . . . .	92
5.1.3	Repository layout . . . . .	92
5.1.4	Promotion of artifacts to stable, aka. releasing . . . . .	93
5.1.5	How to mark a major version . . . . .	93
5.1.6	The release procedure on the maintainer side . . . . .	93
<b>6</b>	<b>Changelog</b>	<b>95</b>
<b>7</b>	<b>Indices and tables</b>	<b>97</b>
	<b>Python Module Index</b>	<b>99</b>



# CHAPTER 1

---

## Lago Introduction

---

Lago is an add-hoc virtual framework which helps you build virtualized environments on your server or laptop for various use cases.

It currently utilizes ‘libvirt’ for creating VMs, but we are working on adding more providers such as ‘containers’.





## 2.1 Installing Lago

Lago is officially supported and tested on Fedora 24+ and CentOS 7.3+ distributions. However, it should be fairly easy to install it on any Linux distribution that can run libvirt and qemu-kvm using *pip*, here we provide instructions also for Ubuntu 16.04 which we test from time to time. Feel free to open PR if you got it running on a distribution which is not listed here so it could be added.

### 2.1.1 pip

1. Install system package dependencies:

- (a) CentOS 7.3+

```
$ yum install -y epel-release centos-release-qemu-ev
$ yum install -y python-devel libvirt libvirt-devel \
    libguestfs-tools libguestfs-devel gcc libffi-devel \
    openssl-devel qemu-kvm-ev
```

- (b) Fedora 24+

```
$ dnf install -y python2-devel libvirt libvirt-devel \
    libguestfs-tools libguestfs-devel gcc libffi-devel \
    openssl-devel qemu-kvm
```

- (c) Ubuntu 16.04+

```
$ apt-get install -y python-dev build-essential libssl-dev \
    libffi-dev qemu-kvm libvirt-bin libvirt-dev pkg-config \
    libguestfs-tools libguestfs-dev
```

2. Install libguestfs Python bindings, as they are not available on PyPI [\[3\]](#):

```
$ pip install http://download.libguestfs.org/python/guestfs-1.36.4.tar.gz
```

### 3. Install Lago with pip:

```
$ pip install lago
```

### 4. Setup permissions(replacing USERNAME accordingly):

- Fedora / CentOS:

```
$ sudo usermod -a -G qemu,libvirt USERNAME
$ sudo usermod -a -G USERNAME qemu
$ sudo chmod g+x $HOME
```

- Ubuntu 16.04+ :

```
$ sudo usermod -a -G libvirtd,kvm USERNAME
$ chmod 0644 /boot/vmlinuz*
```

### 5. Create a global share for Lago to store templates:

```
$ sudo mkdir -p /var/lib/lago
$ sudo mkdir -p /var/lib/lago/{repos,store,subnets}
$ sudo chown -R USERNAME:USERNAME /var/lib/lago
```

*Note:* If you'd like to store the templates in a different location look at the [Configuration](#) section, and change `lease_dir`, `template_repos` and `template_store` accordingly. This can be done after the installation is completed.

### 6. Restart libvirt:

```
$ systemctl restart libvirtd
```

### 7. Log out and login again

Thats it! Lago should be working now. You can jump to [Lago Examples](#).

## 2.1.2 RPM Based - Fedora 24+ / CentOS 7.3+

### 2.1.2.1 Install script

#### 1. Download the installation script and make it executable:

```
$ curl https://raw.githubusercontent.com/lago-project/lago-demo/master/install_
↪scripts/install_lago.sh \
  -o install_lago.sh \
  && chmod +x install_lago.sh
```

#### 2. Run the installation script(replacing username with your username):

```
$ sudo ./install_lago.sh --user [username]
```

#### 3. Log out and login again.

### 2.1.2.2 Manual RPM installation

1. Add the following repository to a new file at `/etc/yum.repos.d/lago.repo`:

For Fedora:

```
[lago]
baseurl=http://resources.ovirt.org/repos/lago/stable/0.0/rpm/fc$releasever
name=Lago
enabled=1
gpgcheck=0
```

For CentOS:

```
[lago]
baseurl=http://resources.ovirt.org/repos/lago/stable/0.0/rpm/el$releasever
name=Lago
enabled=1
gpgcheck=0
```

For CentOS only, you need **EPEL** and **centos-release-qemu-ev** repositories, those can be installed by running:

```
$ sudo yum install -y epel-release centos-release-qemu-ev
```

2. With the Lago repository configured, run(for Fedora use `dnf` instead):

```
$ sudo yum install -y lagoon
```

3. Setup group permissions:

```
$ sudo usermod -a -G lagoon USERNAME
$ sudo usermod -a -G qemu USERNAME
$ sudo usermod -a -G USERNAME qemu
```

4. Add group execution rights to your home directory:<sup>1</sup>

```
$ chmod g+x $HOME
```

5. Restart libvirtd:

```
$ sudo systemctl enable libvirtd
$ sudo systemctl restart libvirtd
```

6. Log out and login again.

### 2.1.3 FAQ

- *Q*: After using the install script, how do I fix the permissions for another username?

A: Run:

```
$ ./install_lago.sh -p --user [new_user]
```

- *Q*: Can Lago be installed in a Python virtualenv?

<sup>1</sup> If the installation script does not work for you on the supported distributions, please open an issue at <https://github.com/lago-project/lago-demo.git>

**A: Follow the same procedure as in the [pip](#) instructions, only run the `pip` installation under your virtualenv.**  
Consult [\[3\]](#) if you want to install `libguestfs` Python bindings not from `pip`.

## 2.1.4 Troubleshooting

- *Problem:* QEMU throws an error it can't access files in my home directory.

*Solution:* Check again that you have setup the permissions described in the [Manual RPM Installation](#) section. After doing that, log out and log in again. If QEMU has the proper permissions, the following command should work( replace `some/nested/path` with a directory inside your home directory):

```
$ sudo -u qemu ls $HOME/some/nested/path
```

For more information why this step is needed see <https://libvirt.org/drvqemu.html>, at the bottom of “POSIX users/groups” section.

- *Problem:* When trying to start the environment Libvirt throws the following error:

```
libvirtError: internal error: Check the host setup: enabling IPv6 forwarding with RA routes without accept_ra set to 2 is likely to cause routes loss. Interfaces to look at: INTERFACE
```

*Solution:* Nat networks that created by Lago are IPv6 enabled by default. In the latest versions of Libvirt, `accept_ra` kernel parameter should be set to 2 in order to create IPv6 enabled networks. This can be achieved with the following command (replace `INTERFACE` with the name of the interface shown in the error message):

```
sudo echo 2 > /proc/sys/net/ipv6/conf/INTERFACE/accept_ra
```

In order to apply and make this change permanent, use the following commands (don't forget to specify your interface):

```
sudo echo "net.ipv6.conf.INTERFACE.accept_ra=2" >> "/etc/sysctl.conf"
sudo sysctl -p
```

## 2.2 LagoInitFile Specification

Note: this is work under progress, if you'd like to contribute to the documentation, please feel free to open a PR. In the meanwhile, we recommend looking at LagoInitFile examples available at:

<https://github.com/lago-project/lago-examples/tree/master/init-files>

Each environment in Lago is created from an init file, the recommended format is YAML, although at the moment of writing JSON is still supported. By default, Lago will look for a file named `LagoInitFile` in the directory it was triggered. However you can pick a different file by running:

```
$ lago init <FILENAME>
```

### 2.2.1 Sections

The init file is composed out of two major sections: domains, and nets. Each virtual machine you wish to create needs to be under the `domains` section. `nets` will define the network topology, and when you add a nic to a domain, it must be defined in the `nets` section.

Example:

```
domains:
  vm-el73:
    memory: 2048
    service_provider: systemd
    nics:
      - net: lagoon
    disks:
      - template_name: el7.3-base
        type: template
        name: root
        dev: vda
        format: qcow2
    artifacts:
      - /var/log
nets:
  lagoon:
    type: nat
    dhcp:
      start: 100
      end: 254
    management: true
    dns_domain_name: lagoon.local
```

### 2.2.1.1 domains

<name>: The name of the virtual machine.

**memory(int)** The virtual machine memory in GBs.

**vcpu(int)** Number of virtual CPUs.

**service\_provider(string)** This will instruct which service provider to use when enabling services in the VM by calling `lago.plugins.vm.VMPlugin.service()`, Possible values: `systemd`, `sysvinit`.

**cpu\_model(string)** CPU Family to emulate for the virtual machine. The list of supported types depends on your hardware and the libvirt version you use, to list them you can run locally:

```
$ virsh cpu-models x86_64
```

**cpu\_custom(dict)** This allows more fine-grained control of the CPU type, see [CPU](#) section for details.

**nics(list)** Network interfaces. Each network interface must be defined in the global `nets` section. By default each nic will be assigned an IP according to the network definition. However, you may also use static IPs here, by writing:

```
nics:
  - net: net-01
    ip: 192.168.220.2
```

The same network can be declared multiple times for each domain.

**disks(list)**

**type** Disk type, possible values:

**template** A Lago template, this would normally be the bootable device.

**file** A local disk image. Lago will thinly provision it during init stage, this device will not be bootable. But can obviously be used for additional storage.

**template\_name(string)** Applies only to disks of type `template`. This should be one of the available Lago templates, see [Templates](#) section for the list.

**size(string)** Disk size to thinly provision in GB. This is only supported in `file` disks.

**format(string)** *TO-DO: no docs yet..*

**device(string)** Linux device: `vda`, `sdb`, etc. Using a device named “`sd*`” will use `virtio-scsi`.

**build(list)** This section should describe how to build/configure VMs. The build/configure action will happen during `init`.

**virt-customize(dict)** Instructions to pass to `virt-customize`, where the key is the name of the option and the value is the arguments for that option.

This operation is only supported on disks which contains OS.

A special instruction is `ssh-inject: ''` Which will ensure Lago’s generated SSH keys will be injected into the VM. This is useful when you don’t want to run the bootstrap stage.

For example:

```
- template_name: el7.3-base
  build:
    - virt-customize:
      ssh-inject: ''
      touch: [/root/file1, /root/file2]
```

See [build](#) section for details.

**artifacts(list)** Paths on the VM that Lago should collect when using `lago collect` from the CLI, or `collect_artifacts()` from the SDK.

**groups(list)** Groups this VM belongs to. This is most usefull when deploying the VM with Ansible.

**bootstrap(bool)** Whether to run bootstrap stage on the VM’s template disk, defaults to `True`.

**ssh-user(string)** SSH user to use and configure, defaults to `root`

**vm-provider(string)** VM Provider plugin to use, defaults to `local-libvirt`.

**vm-type(string)** VM Plugin to use. A custom VM Plugin can be passed here, note that it needs to be available in your Python Entry points. See [lago-ost-plugin](#) for an example.

**metadata(dict)** *TO-DO: no docs yet..*

### 2.2.1.2 nets

**<name>:** The name of the network.

**type(string)** Type of the network. May be `nat` or `bridge`.

## 2.3 Lago SDK

The SDK goal is to automate the creation of virtual environments, by using Lago directly from Python. Currently, most CLI operations are supported from the SDK, though not all of them(specifically, snapshot and export).

## 2.3.1 Starting an environment from the SDK

### 2.3.1.1 Prerequisites

1. Have Lago installed, see the [installation notes](#).
2. Create a LagoInitFile, check out [LagoInitFile syntax](#) for more details.

### 2.3.1.2 Prepare the environment

*Note:* This example is available as a Jupyter notebook [here](#) or converted to reST [here](#).

Assuming the LagoInitFile is saved in `/tmp/el7-init.yaml` and contains:

```
domains:
  vm01:
    memory: 1024
    nics:
      - net: lagoon
    disks:
      - template_name: el7.3-base
        type: template
        name: root
        dev: sda
        format: qcow2
nets:
  lagoon:
    type: nat
    dhcp:
      start: 100
      end: 254
```

Which is a simple setup, containing one CentOS 7.3 virtual machine and one management network. Then you start the environment by running:

```
import logging
from lago import sdk

env = sdk.init(config='/tmp/el7-init.yaml',
               workdir='/tmp/my_test_env',
               logfile='/tmp/lago.log',
               loglevel=logging.DEBUG)
```

**Where:**

1. `config` is the path to a valid init file, in YAML format.
2. `workdir` is the place Lago will use to save the images and metadata.
3. The `logfile` and `loglevel` parameters add a FileHandler to Lago's root logger.

Note that if this is the first time you are running Lago it will first download the template(in this example `el7-base`), which might take a while<sup>1</sup>. You can follow up the progress by watching the log file, or alternatively if working in an interactive session, by running:

<sup>1</sup> On a normal setup, where the templates are already downloaded, the `init` stage should take less than a minute(but probably at least 15 seconds).

```
from lago import sdk
sdk.add_stream_logger()
```

Which will print all the Lago operations to stdout.

### 2.3.1.3 Starting the environment

Once `init()` method returns, the environment is ready to be started, taking up from the last example, executing:

```
env.start()
```

Would start the VMs specified in the init file, and make them available (among others) through SSH:

```
>>> vm = env.get_vms() ['vm01']
>>> vm.ssh(['hostname', '-f'])
CommandStatus(code=0, out='vm01.lago.local\n', err='')
```

You can also run an interactive SSH session:

```
>>> res = vm.interactive_ssh()
[root@vm01 ~]# ls -lsah
total 20K
0 dr-xr-x---.  3 root root 103 May 28 03:11 .
0 dr-xr-xr-x. 17 root root 224 Dec 12 17:00 ..
4.0K -rw-r--r--. 1 root root  18 Dec 28 2013 .bash_logout
4.0K -rw-r--r--. 1 root root 176 Dec 28 2013 .bash_profile
4.0K -rw-r--r--. 1 root root 176 Dec 28 2013 .bashrc
4.0K -rw-r--r--. 1 root root 100 Dec 28 2013 .cshrc
0 drwx-----. 2 root root  29 May 28 03:11 .ssh
4.0K -rw-r--r--. 1 root root 129 Dec 28 2013 .tcshrc
[root@vm01 ~]# exit
exit
>>> res.code
0
```

### 2.3.1.4 Controlling the environment

You can start or stop the environment by calling `start()` and `stop()`, finally you can destroy the environment with `lago.sdk.SDK.destroy()` method, note that it will stop all VMs, and remove the provided working directory.

```
>>> env.destroy()
>>>
```

### Disk consumption for the workdir

Generally speaking, the workdir disk consumption depends on which operation you run inside the underlying VMs. Lago uses QCOW2 layered images by default, so that each environment you create, sets up its own layer on top of the original template Lago downloaded the first time `init` was ran with that specific template. So when the VM starts, it usually consumes less than 30MB. As you do more operations - the size might increase, as your current image diverges from the original template. For more information see [qemu-img](#)



### 2.3.2 Differences from the CLI

1. Creating Different `prefixes` inside the `workdir` is not supported. In the CLI, you can have several prefixes inside a `workdir`. The reasoning behind that is that when working from Python, you can manage the environment directly by your own(using a temporary or fixed path).
2. Logging - In the CLI, all log operations are kept in the current `prefix` under `logs/lago.log` path. The SDK keeps that convention, but allows you to add additional log files by passing log filename and level parameters to `init()` command. Additionally, you can work in debug mode, by logging all commands to stdout and stderr, calling the module-level method `add_stream_logger()`. Note that this will log everything for all environments.
3. `Prefix` class. This is more of an implementation issue: the core per-environment operations are exposed both for the CLI and SDK in that class. In order to provide consistency and ease of use for the SDK, only the methods which make sense for SDK usage are exposed in the SDK, the CLI does not require that, as the methods aren't exposed at all(only verbs in `py`).

## 2.4 Getting started with some Lago Examples!

Get Lago up & running in no time using one of the available examples

Important: make sure you followed the installation step before to have Lago installed.

### 2.4.1 Available Examples

- [LagoInitFiles](#) examples
- Simple Jenkins server + slaves: [Jenkins\\_Example](#)
- Advanced oVirt example (using nested virtualization): [oVirt\\_Example](#)
- SDK Usage example - [GitHub](#), or in [reST](#)
- Integrating Lago with [Pytest](#)

## 2.5 Lago Templates

We maintain several templates which are publicly available [here](#), and Lago will use them by default. We try to ensure each of those templates is fully functional out of the box. All templates are more or less the same as the standard cloud image for each distribution.

The templates specification and build scripts are managed in a different [repository](#), and it should be fairly easy to create your own templates repository.

## 2.5.1 Available templates

Template name	OS
el7-base	CentOS 7.2
el7.3-base	CentOS 7.3
el7.4-base	CentOS 7.4
fc23-base	Fedora 23
fc24-base	Fedora 24
fc25-base	Fedora 25
fc26-base	Fedora 26
el6-base	CentOS 6.7
debian8-base	Debian 8
ubuntu16.04-base	Ubuntu 16.04

## 2.5.2 Repository metadata

A templates repository should contain a *repo.metadata* file describing all available templates. The repository build script creates this file automatically. The file contains a serialized JSON object with the members detailed below. For an example, see the above repository's [metadata file](#).

**name:** The name of the repository.

**sources:**

<name>: Name of a source.

**type:** Source type. May be either `http` or `file`.

**args:** Varies depending on the source type.

For an `http` source, should contain a `baseurl` member pointing to the root of the repository on the web.

For a `file` source, should contain a `root` member pointing to the root of the repository on the filesystem.

**templates:**

<name>: Unique template name.

**versions:**

<version>: Unique version string.

**source:** Name of the source from which this template version was created.

**timestamp:** Creation time of the template version.

**handle:** Either a base file name of the template located in the root directory of the repository, or a root-relative path to the template file.

## 2.6 Configuration

The recommend method to override the configuration file is by letting lago auto-generate them:

```
$ mkdir -p $HOME/.config/lago
$ lago generate-config > $HOME/.config/lago/lago.conf
```

This will dump the current configuration to `$HOME/.config/lago/lago.conf`, and you may edit it to change any parameters. Take into account you should probably comment out parameters you don't want to change when editing the file. Also, all parameters in the configuration files can be overridden by passing command line arguments or with environment variables, as described below.

### 2.6.1 lago.conf format

Lago runs without a configuration file by default, for reference-purposes, when lago is installed from the official packages(RPM or DEB), a commented-out version of `lago.conf`(INI format) is installed at `/etc/lago/lago.conf`.

In `lago.conf` global parameters are found under the `[lago]` section. All other sections usually map to subcommands(i.e. `lago init` command would be under `[init]` section).

*Example:*

```
$ lago generate-config
> [lago]
> # log level to use
> loglevel = info
> logdepth = 3
> ....
> [init]
> # location to store repos
> template_repos = /var/lib/lago/repos
> ...
```

### 2.6.2 lago.conf look-up

Lago attempts to look `lago.conf` in the following order:

1. `/etc/lago/lago.conf`
2. According to [XDG standards](#) , which are by default:
  - `/etc/xdg/lago/lago.conf`
  - `/home/$USER/.config/lago/lago.conf`
3. Any environment variables.
4. CLI passed arguments.

If more than one file exists, all files are merged, with the last occurrence of any parameter found used.

### 2.6.3 Overriding parameters with environment variables

To differentiate between the root section in the configuration file, lago uses the following format to look for environment variables:

```
'LAGO_GLOBAL_VAR' -> variable in [lago] section
'LAGO__SUBCOMMAND__PARAM_1' -> variable in [subcommand] section
```

Example: changing the `template_store` which `init` subcommand uses to store templates:

```
# check current value:
$ lago generate-config | grep -A4 "init"
> [init]
> # location to store repos
> template_repos = /var/lib/lago/repos
> # location to store temp
> template_store = /var/lib/lago/store

$ export LAGO__INIT__TEMPLATE_STORE=/var/tmp
$ lago generate-config | grep -A4 "init"
> [init]
> # location to store repos
> template_repos = /var/lib/lago/repos
> # location to store temp
> template_store = /var/tmp
```

## 2.7 Lago build

Lago allows to build / configure VM disks during init stage. In the init file, the key `build` should be added to each disk that needs to be configured.

`build` should map to a list of Builders, where each builder in the list is a one entry dictionary that maps to a dictionary that holds the options for that builder.

Options are key-value pairs, where the key is the name of the option (without leading dashes), and the value is the argument for that option. If the option takes no arguments, the empty string should be set as the value. If the builder allows specifying an options multiple times, the value should be a list of arguments.

*Note:* The build process runs “behind-the-back” of the OS (Before the VM starts), thus should be used with care.

### 2.7.1 Builders

Builders are commands that can be used to build/configure VMs. Builders are called by the order they appear in the init file.

#### 2.7.1.1 virt-customize

A tool for customizing a virtual machine (install packages, copying files, etc...). *virt-customize* is part of the *libguestfs* tool set which is part of Lago’s dependencies.

*virt-customize* can be called only on disks which contains an OS.

Depends on the version of *virt-customize* installed on your system (it can vary between different OS), all the valid options for *virt-customize* can be specified in the init file. For the full list of options please refer to [virt-customize documentation](#).

There is a special case when using *virt-customize* to inject ssh keys. If the empty string is provided to `ssh-inject` option, Lago will replace it with the path to lago’s self generated ssh keys.

*Note:* If SELinux is enabled in a VM, it’s possible that `selinux-relabel` will be required after adding / modifying its files.

## 2.7.2 Relation to bootstrap

Configuration is taking place after Lago runs bootstrap. You can disable bootstrap to all VMs by passing `--skip-bootstrap` to `lago init`, or by adding `bootstrap: false` to the VM definition in the init file (the second allows to control bootstrap per VM).

Since bootstrap is injecting ssh keys to the VMs, If skipping it, it's recommended to inject the ssh keys using *virt-customize* builder otherwise, shell access to the VM will use password authentication (more details can be found in the Builders sections in this documents).

## 2.7.3 Example

In the following example, *virt-customize* builder will be called on the disk of vm01.

The changes will be:

- Injecting lago's self generated ssh keys.
- Copy `dummy_file` from the host to `/root` in vm01
- Create files `/root/file1` and `/root/file2` in vm01
- Finish with SELinux relabel of vm01.

```
domains:
  vm01:
    artifacts: [/var/log]
    bootstrap: false
    disks:
      - build:
          - virt-customize:
              ssh-inject: ''
              copy: dummy_file:/root
              touch: [/root/file1, /root/file2]
              selinux-relabel: ''
    dev: vda
    format: qcow2
    name: root
    path: $LAGO_INITFILE_PATH/lago-basic-suite-4-1-engine_root.qcow2
    template_name: el7.3-base
    template_type: qcow2
    type: template
```

## 2.8 Lago CPU Models in detail

There are several ways you can configure the CPU model Lago will use for each VM. This section tries to explain more in-depth how it will be mapped to libvirt XML.

- `vcpu`: Number of virtual CPUs.
- `cpu_model`: `<model>`: This defines an exact match of a CPU model. The generated Libvirt `<cpu>` XML will be:

```
<cpu>
  <model>Westmere</model>
  <topology cores="1" sockets="3" threads="1"/>
```

(continues on next page)

(continued from previous page)

```
<feature name="vmx" policy="require"/>
</cpu>
```

If the vendor of the host CPU and the selected model match, it will attempt to require `vmx` on Intel CPUs and `svm` on AMD CPUs, assuming the host CPU has that feature. The topology node will be generated with sockets equals to `vcpu` parameter, by default it is set to 2.

- `cpu_custom`: This allows to override entirely the CPU definition, by writing the domain CPU XML in YAML syntax, for example, for the following `LagoInitFile`:

```
domains:
  vm-el73:
    vcpu: 2
    cpu_custom:
      '@mode': custom
      '@match': exact
    model:
      '@fallback': allow
      '#text': Westmere
    feature:
      '@policy': optional
      '@name': 'vmx'
    numa:
      cell:
        -
          '@id': 0
          '@cpus': 0
          '@memory': 2048
          '@unit': 'MiB'
        -
          '@id': 1
          '@cpus': 1
          '@memory': 2048
          '@unit': 'MiB'
    ...
```

This will be the generated `<cpu>` XML:

```
<cpu mode="custom" match="exact">
  <model fallback="allow">Westmere</model>
  <feature policy="optional" name="vmx"/>
  <numa>
    <cell id="0" cpus="0" memory="2048" unit="MiB"/>
    <cell id="1" cpus="1" memory="2048" unit="MiB"/>
  </numa>
  <topology cores="1" sockets="2" threads="1"/>
</cpu>
<vcpu>2</vcpu>
```

The conversion is pretty straight-forward, `@` maps to attribute, and `#text` to text fields. If `topology` section is not defined, it will be added.

- No `cpu_custom` or `cpu_model`: Then Libvirt's `host-passthrough` will be used. For more information see: [Libvirt CPU model](#)

### 3.1 CI Process

Here is described the usual workflow of going through the CI process from starting a new branch to getting it merged and released in the [unstable repo](#).

#### 3.1.1 Starting a branch

First of all, when starting to work on a new feature or fix, you have to start a new branch (in your fork if you don't have push rights to the main repo). Make sure that your branch is up to date with the project's master:

```
git checkout -b my_fancy_feature
# in case that origin is already lagoon-project/lagoon
git reset --hard origin/master
```

Then, once you can just start working, doing commits to that branch, and pushing to the remote from time to time as a backup.

Once you are ready to run the ci tests, you can create a pull request to master branch, if you have [hub](#) installed you can do so from command line, if not use the ui:

```
$ hub pull-request
```

That will automatically trigger a test run on ci, you'll see the status of the run in the pull request page. At that point, you can keep working on your branch, probably just rebasing on master regularly and maybe amending/squashing commits so they are logically meaningful.

#### 3.1.2 A clean commit history

An example of not good pull request history:

- Added `right_now` parameter to `virt.VM.start` function

- Merged master into my\_fancy\_feature
- Added tests for the new parameter case
- Renamed right\_now parameter to sudo\_right\_now
- Merged master into my\_fancy\_feature
- Adapted test to the rename

This history can be greatly improved if you squashed a few commits:

- Added sudo\_right\_now parameter to virt.VM.start function
- Added tests for the new parameter case
- Merged master into my\_fancy\_feature
- Merged master into my\_fancy\_feature

And even more if instead of merging master, you just rebased:

- Added sudo\_right\_now parameter to virt.VM.start function
- Added tests for the new parameter case

That looks like a meaningful history :)

### 3.1.3 Rerunning the tests

While working on your branch, you might want to rerun the tests at some point, to do so, you just have to add a new comment to the pull request with one of the following as content:

- ci test please
- ci :+1:
- ci :thumbsup:

### 3.1.4 Asking for reviews

If at any point, you see that you are not getting reviews, please add the label ‘needs review’ to flag that pull request as ready for review.

### 3.1.5 Getting the pull request merged

Once the pull request has been reviewed and passes all the tests, an admin can start the merge process by adding a comment with one of the following as content:

- ci merge please
- ci :shipit:

That will trigger the merge pipeline, that will run the tests on the merge commit and deploy the artifacts to the [unstable repo](#) on success.

## 3.2 Environment setup

Here are some guidelines on how to set up your development of the lago project.



### 3.2.1 Requirements

You'll need some extra packages to get started with the code for lago, assuming you are running Fedora:

```
> sudo dnf install git mock libvirt-daemon qemu-kvm autotools
```

And you'll need also a few Python libs, which you can install from the repos or use venv or similar, for the sake of this example we will use the repos ones:

```
> sudo dnf install python-flake8 python-nose python-dulwich yapf
```

Yapf is not available on older Fedoras or CentOS, you can get it from the [official yapf repo](#) or try on [copr](#).

Now you are ready to get the code:

```
> git clone git@github.com:lago-project/lago.git
```

From now on all the commands will be based from the root of the cloned repo:

```
> cd lago
```

### 3.2.2 Style formatting

We will accept only patches that pass pep8 and that are formatted with yapf. More specifically, only patches that pass the local tests:

```
> make check-local
```

It's recommended that you setup your editor to check automatically for pep8 issues. For the yapf formatting, if you don't want to forget about it, you can install the pre-commit git hook that comes with the project code:

```
> ln -s scripts/pre-commit.style .git/pre-commit
```

Now each time that you run `git commit` it will automatically reformat the code you changed with yapf so you don't have any issues when submitting a patch.

### 3.2.3 Testing your changes

Once you do some changes, you should make sure they pass the checks, there's no need to run on each edition but before submitting a patch for review you should do it.

You can run them on your local machine, but the tests themselves will install packages and do some changes to the os, so it's really recommended that you use a vm, or as we do on the CI server, use mock chroots. If you don't want to setup mock, skip the next section.

Hopefully in a close future we can use lago for that ;)

#### 3.2.3.1 Setting up mock\_runner.sh with mock (fedora)

For now we are using a script developed by the *oVirt* devels to generate chroots and run tests inside them, it's not packaged yet, so we must get the code itself:

```
> cd ..
> git clone git://gerrit.ovirt.org/jenkins
```

As an alternative, you can just download the script and install them in your *\$PATH*:

```
> wget https://gerrit.ovirt.org/gitweb?p=jenkins.git;a=blob_plain;f=mock_configs/mock_
↪runner.sh;hb=refs/heads/master
```

We will need some extra packages:

```
> sudo dnf install mock
```

And, if not running as root (you shouldn't!) you have to add your user to the newly created mock group, and make sure the current session is in that group:

```
> sudo usermod -a -G mock $USER
> newgrp mock
> id # check that mock is listed
```

### 3.2.3.2 Running the tests inside mock

Now we have all the setup we needed, so we can go back to the lago repo and run the tests, the first time you run them, it will take a while to download all the required packages and install them in the chroot, but on consecutive runs it will reuse all the cached chroots.

The *mock\_runner.sh* script allows us to test also different distributions, any that is supported by mock, for example, to run the tests for fedora 23 you can run:

```
> ../jenkins/mock_runner.sh -p fc23
```

That will run all the *check-patch.sh* (the *-p* option) tests inside a chroot, with a minimal fedora 23 installation. It will leave any logs under the *logs* directory and any generated artifacts under *exported-artifacts*.

## 3.3 Getting started developing

Everyone is welcome to send patches to lago, but we know that not everybody knows everything, so here's a reference list of technologies and methodologies that lago uses for reference.

### 3.3.1 Python!

Lago is written in python 2.7 (for now), so you should get yourself used to basic-to-medium python constructs and technics like:

- Basic python: Built-in types, flow control, pythonisms (import this)
- Object oriented programming (OOP) in python: Magic methods, class inheritance

Some useful resources:

- Base docs: <https://docs.python.org/2.7/>
- Built-in types: <https://docs.python.org/2.7/library/stdtypes.html>
- About classes: <https://docs.python.org/2.7/reference/datamodel.html#new-style-and-classic-classes>
- The Zen of Python:

```
> python -c "import this"

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

### 3.3.2 Bash

Even though there is not much bash code, the functional tests and some support scripts use it, so better to get some basics on it. We will try to follow the same standards for it than the [oVirt project](#) has.

### 3.3.3 Libvirt + qemu/kvm

As we are using intensively libvirt and qemu/kvm, it's a good idea to get yourself familiar with the main commands and services:

- libvirt: <http://libvirt.org>
- virsh client: <http://libvirt.org/virshcmdref.html>
- qemu (qemu-img is useful to deal with vm disk images): <https://en.wikibooks.org/wiki/QEMU/Images>

Also, there's a library and a set of tools from the [libguestfs](#) project that we use to prepare templates and are very useful when debugging, make sure you play at least with virt-builder, virt-customize, virt-sparsify and guestmount.

### 3.3.4 Git + Github

We use git as code version system, and we host it on Github right now, so if you are not familiar with any of those tools, you should get started with them, specially you should be able to:

- Clone a repo from github
- Fork a repo from github
- Create/delete/move to branches (git checkout)
- Move to different points in git history (git reset)
- Create/delete tags (git tag)

- See the history (git log)
- Create/amend commits (git commit)
- Retrieve changes from the upstream repository (git fetch)
- Apply your changes on top of the retrieved ones (git rebase)
- Apply your changes as a merge commit (git merge)
- Squash/reorder existing commits (git rebase –interactive)
- Send your changes to the upstream (git push)
- Create a pull request

You can always go to [the git docs](#) though there is a lot of good literature on it too.

### 3.3.5 Unit tests with py.test

Lately we decided to use [py.test](#) for the unit tests, and all the current unit tests were migrated to it. We encourage adding unit tests to any pull requests you send.

### 3.3.6 Functional tests with bats

For the functional tests, we decided to use [bats framework](#). It's completely written in bash, and if you are modifying or adding any functionality, you should add/modify those tests accordingly. It has a couple of custom constructs, so take a look to the [bats docs](#) while reading/writing tests.

### 3.3.7 Packaging

Our preferred distribution vector is through packages. Right now we are only building for rpm-based system, so right now you can just take a peek on [how to build rpms](#). Keep in mind also that we try to move as much of the packaging logic as possible to the [python packaging system](#) itself too, worth getting used to it too.

### 3.3.8 Where to go next

You can continue [setting up your environment](#) and try running the examples in the [readme](#) to get used to lago. Once you get familiar with it, you can pick any of the [existing issues](#) and send a pull request to fix it, so you get used to the [ci process](#) we use to get stuff developed flawlessly and quickly, welcome!

## 4.1 lago package

### 4.1.1 Subpackages

#### 4.1.1.1 lago.plugins package

**exception** `lago.plugins.NoSuchPluginError`

Bases: `lago.plugins.PluginError`

`lago.plugins.PLUGIN_ENTRY_POINTS` = {'cli': 'lago.plugins.cli', 'out': 'lago.plugins.outp

Map of plugin type string -> setuptools entry point

**class** `lago.plugins.Plugin`

Bases: `object`

Base class for all the plugins

**exception** `lago.plugins.PluginError`

Bases: `exceptions.Exception`

`lago.plugins.load_plugins(namespace, instantiate=True)`

Loads all the plugins for the given namespace

#### Parameters

- **namespace** (*str*) – Namespace string, as in the setuptools entry\_points
- **instantiate** (*bool*) – If true, will instantiate the plugins too

**Returns** Returns the list of loaded plugins

**Return type** dict of str, `object`

## Submodules

### lago.plugins.cli module

#### About CLIPlugins

A CLIPlugin is a subcommand of the lagocli command, it's ment to group actions together in a logical sense, for example grouping all the actions done to templates.

To create a new subcommand for testenvcli you just have to subclass the CLIPlugin abstract class and declare it in the setuptools as an entry\_point, see this module's setup.py/setup.cfg for an example:

```
class NoopCLIplugin(CLIPlugin):
    init_args = {
        'help': 'dummy help string',
    }

    def populate_parser(self, parser):
        parser.addArgument('--dummy-flag', action='store_true')

    def do_run(self, args):
        if args.dummy_flag:
            print "Dummy flag passed to noop subcommand!"
        else:
            print "Dummy flag not passed to noop subcommand!"
```

You can also use decorators instead, an equivalent is:

```
@cli_plugin_add_argument('--dummy-flag', action='store_true')
@cli_plugin(help='dummy help string')
def my_fancy_plugin_func(dummy_flag, **kwargs):
    if dummy_flag:
        print "Dummy flag passed to noop subcommand!"
    else:
        print "Dummy flag not passed to noop subcommand!"
```

Or:

```
@cli_plugin_add_argument('--dummy-flag', action='store_true')
def my_fancy_plugin_func(dummy_flag, **kwargs):
    "dummy help string"
    if dummy_flag:
        print "Dummy flag passed to noop subcommand!"
    else:
        print "Dummy flag not passed to noop subcommand!"
```

Then you will need to add an entry\_points section in your setup.py like:

```
setup(
    ...
    entry_points={
        'lago.plugins.cli': [
            'noop=noop_module:my_fancy_plugin_func',
        ],
    },
    ...
)
```

Or in your setup.cfg like:

```
[entry_points]
lago.plugins.cli =
    noop=noop_module:my_fancy_plugin_func
```

Any of those will add a new subcommand to the lagocli command that can be run as:

```
$ lagocli noop
Dummy flag not passed to noop subcommand!
```

TODO: Allow per-plugin namespacing to get rid of the *\*\*kwargs* parameter

**class** lago.plugins.cli.CLIPlugin

Bases: *lago.plugins.Plugin*

**\_abc\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache\_version** = 35

**\_abc\_registry** = <\_weakrefset.WeakSet object>

**do\_run** (*args*)

Execute any actions given the arguments

**Parameters** *args* (*Namespace*) – with the arguments

**Returns** None

**init\_args**

Dictionary with the argument to initialize the cli parser (for example, the help argument)

**populate\_parser** (*parser*)

Add any required arguments to the parser

**Parameters** *parser* (*ArgumentParser*) – parser to add the arguments to

**Returns** None

**class** lago.plugins.cli.CLIPluginFuncWrapper (*do\_run=None, init\_args=None*)

Bases: *lago.plugins.cli.CLIPlugin*

Special class to handle decorated cli plugins, take into account that the decorated functions have some limitations on what arguments can they define actually, if you need something complicated, used the abstract class *CLIPlugin* instead.

Keep in mind that right now the decorated function must use *\*\*kwargs* as param, as it will be passed all the members of the parser, not just whatever it defined

**\_\_call\_\_** (*\*args, \*\*kwargs*)

Keep the original function interface, so it can be used elsewhere

**\_abc\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache\_version** = 35

**\_abc\_registry** = <\_weakrefset.WeakSet object>

**add\_argument** (*\*argument\_args, \*\*argument\_kwargs*)

**do\_run** (*args*)

Execute any actions given the arguments

**Parameters** `args` (*Namespace*) – with the arguments

**Returns** `None`

`init_args`

**populate\_parser** (*parser*)

Add any required arguments to the parser

**Parameters** `parser` (*ArgumentParser*) – parser to add the arguments to

**Returns** `None`

**set\_help** (*help=None*)

**set\_init\_args** (*init\_args*)

`lago.plugins.cli.cli_plugin` (*func=None, \*\*kwargs*)

Decorator that wraps the given function in a *CLIPlugin*

**Parameters**

- **func** (*callable*) – function/class to decorate
- **\*\*kwargs** – Any other arg to use when initializing the parser (like `help`, or `prefix_chars`)

**Returns** cli plugin that handles that method

**Return type** *CLIPlugin*

## Notes

It can be used as a decorator or as a decorator generator, if used as a decorator generator don't pass any parameters

## Examples

```
>>> @cli_plugin
... def test(**kwargs):
...     print 'test'
...
>>> print test.__class__
<class 'cli.CLIPluginFuncWrapper'>
```

```
>>> @cli_plugin()
... def test(**kwargs):
...     print 'test'
...
>>> print test.__class__
<class 'cli.CLIPluginFuncWrapper'>
```

```
>>> @cli_plugin(help='dummy help')
... def test(**kwargs):
...     print 'test'
...
>>> print test.__class__
<class 'cli.CLIPluginFuncWrapper'>
>>> print test.init_args['help']
'dummy help'
```



`lago.plugins.cli.cli_plugin_add_argument(*args, **kwargs)`

Decorator generator that adds an argument to the cli plugin based on the decorated function

#### Parameters

- **\*args** – Any args to be passed to `argparse.ArgumentParser.add_argument()`
- **\*kwargs** – Any keyword args to be passed to `argparse.ArgumentParser.add_argument()`

#### Returns

**Decorator that builds or extends the cliplugin for the** decorated function, adding the given argument definition

**Return type** function

#### Examples

```
>>> @cli_plugin_add_argument('-m', '--mogambo', action='store_true')
... def test(**kwargs):
...     print 'test'
...
>>> print test.__class__
<class 'cli.CLIPluginFuncWrapper'>
>>> print test._parser_args
[('-', 'mogambo'), {'action': 'store_true'}]
```

```
>>> @cli_plugin_add_argument('-m', '--mogambo', action='store_true')
... @cli_plugin_add_argument('-b', '--bogabmo', action='store_false')
... @cli_plugin
... def test(**kwargs):
...     print 'test'
...
>>> print test.__class__
<class 'cli.CLIPluginFuncWrapper'>
>>> print test._parser_args
[('-', 'bogabmo'), {'action': 'store_false'}],
[('-', 'mogambo'), {'action': 'store_true'}]
```

`lago.plugins.cli.cli_plugin_add_help(help)`

Decorator generator that adds the cli help to the cli plugin based on the decorated function

**Parameters** `help (str)` – help string for the cli plugin

#### Returns

**Decorator that builds or extends the cliplugin for the** decorated function, setting the given help

**Return type** function

#### Examples

```
>>> @cli_plugin_add_help('my help string')
... def test(**kwargs):
...     print 'test'
```

(continues on next page)

(continued from previous page)

```
...
>>> print test.__class__
<class 'cli.CLIPuginFuncWrapper'>
>>> print test.help
my help string
```

```
>>> @cli_plugin_add_help('my help string')
... @cli_plugin()
... def test(**kwargs):
...     print 'test'
>>> print test.__class__
<class 'cli.CLIPuginFuncWrapper'>
>>> print test.help
my help string
```

## lago.plugins.output module

### About OutFormatPlugins

An OutFormatPlugin is used to format the output of the commands that extract information from the prefixes, like status.

**class** lago.plugins.output.DefaultOutFormatPlugin

Bases: *lago.plugins.output.OutFormatPlugin*

**\_abc\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache\_version** = 35

**\_abc\_registry** = <\_weakrefset.WeakSet object>

**format** (*info\_obj*, *indent*=")

Execute any actions given the arguments

**Parameters** *info\_dict* (*dict*) – information to reformat

**Returns** String representing the formatted info

**Return type** *str*

**indent\_unit** = ' '

**class** lago.plugins.output.FlatOutFormatPlugin

Bases: *lago.plugins.output.OutFormatPlugin*

**\_abc\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache\_version** = 35

**\_abc\_registry** = <\_weakrefset.WeakSet object>

**format** (*info\_dict*, *delimiter*='/')

This formatter will take a data structure that represent a tree and will print all the paths from the root to the leaves

in our case it will print each value and the keys that needed to get to it, for example:

**vm0:** net: lago memory: 1024

will be output as:

vm0/net/lago vm0/memory/1024

**Args:** `info_dict` (dict): information to reformat  
`delimiter` (str): a delimiter for the path components

**Returns:** str: String representing the formatted info

**class** `lago.plugins.output.JSONOutFormatPlugin`

Bases: `lago.plugins.output.OutFormatPlugin`

`_abc_cache` = `<_weakrefset.WeakSet object>`

`_abc_negative_cache` = `<_weakrefset.WeakSet object>`

`_abc_negative_cache_version` = 35

`_abc_registry` = `<_weakrefset.WeakSet object>`

**format** (`info_dict`)

Execute any actions given the arguments

**Parameters** `info_dict` (`dict`) – information to reformat

**Returns** String representing the formatted info

**Return type** str

**class** `lago.plugins.output.OutFormatPlugin`

Bases: `lago.plugins.Plugin`

`_abc_cache` = `<_weakrefset.WeakSet object>`

`_abc_negative_cache` = `<_weakrefset.WeakSet object>`

`_abc_negative_cache_version` = 35

`_abc_registry` = `<_weakrefset.WeakSet object>`

**format** (`info_dict`)

Execute any actions given the arguments

**Parameters** `info_dict` (`dict`) – information to reformat

**Returns** String representing the formatted info

**Return type** str

**class** `lago.plugins.output.YAMLOutFormatPlugin`

Bases: `lago.plugins.output.OutFormatPlugin`

`_abc_cache` = `<_weakrefset.WeakSet object>`

`_abc_negative_cache` = `<_weakrefset.WeakSet object>`

`_abc_negative_cache_version` = 35

`_abc_registry` = `<_weakrefset.WeakSet object>`

**format** (`info_dict`)

Execute any actions given the arguments

**Parameters** `info_dict` (`dict`) – information to reformat

**Returns** String representing the formatted info

Return type `str`

## lago.plugins.service module

### Service Plugin

This plugins are used in order to manage services in the vms

**class** `lago.plugins.service.ServicePlugin` (*vm*, *name*)

Bases: `lago.plugins.Plugin`

#### **BIN\_PATH**

Path to the binary used to manage services in the vm, will be checked for existence when trying to decide if the service is supported on the VM (see *func:is\_supported*).

**Returns** Full path to the binary inside the domain

Return type `str`

`_abc_cache = <weakrefset.WeakSet object>`

`_abc_negative_cache = <weakrefset.WeakSet object>`

`_abc_negative_cache_version = 35`

`_abc_registry = <weakrefset.WeakSet object>`

`_request_start()`

Low level implementation of the service start request, used by the *func:start* method

**Returns** True if the service succeeded to start, False otherwise

Return type `bool`

`_request_stop()`

Low level implementation of the service stop request, used by the *func:stop* method

**Returns** True if the service succeeded to stop, False otherwise

Return type `bool`

`alive()`

`exists()`

**classmethod** `is_supported` (*vm*)

`start()`

`state()`

Check the current status of the service

**Returns** Which state the service is at right now

Return type `ServiceState`

`stop()`

**class** `lago.plugins.service.ServiceState`

Bases: `enum.Enum`

**ACTIVE** = 2

**INACTIVE** = 1

**MISSING = 0**

This state corresponds to a service that is not available in the domain

## lago.plugins.vm module

### VM Plugins

There are two VM-related plugin extension points, there's the VM Type Plugin, that allows you to modify at a higher level the inner workings of the VM class (domain concept in the initfile). The other plugin extension point, the [VM Provider Plugin], that allows you to create an alternative implementation of the provisioning details for the VM, for example, using a remote libvirt instance or similar.

**exception** lago.plugins.vm.**ExtractPathError**

Bases: *lago.plugins.vm.VMError*

**exception** lago.plugins.vm.**ExtractPathNoPathError**

Bases: *lago.plugins.vm.VMError*

**exception** lago.plugins.vm.**LagoFailedToGetVMStateError**

Bases: *lago.utils.LagoException*

**exception** lago.plugins.vm.**LagoVMDoesNotExistError**

Bases: *lago.utils.LagoException*

**exception** lago.plugins.vm.**LagoVMNotRunningError** (*vm\_name*)

Bases: *lago.utils.LagoUserException*

**exception** lago.plugins.vm.**VMError**

Bases: *exceptions.Exception*

**class** lago.plugins.vm.**VMPlugin** (*env, spec*)

Bases: *lago.plugins.Plugin*

**\_abc\_cache** = *<\_weakrefset.WeakSet object>*

**\_abc\_negative\_cache** = *<\_weakrefset.WeakSet object>*

**\_abc\_negative\_cache\_version** = 35

**\_abc\_registry** = *<\_weakrefset.WeakSet object>*

**\_artifact\_paths** ()

**\_detect\_service\_provider** ()

**\_get\_service\_provider** ()

**NOTE:** Can be reduced to just one get call once we remove support for the service\_class spec entry

**Returns** class for the loaded provider for that vm\_spec None: if no provider was specified in the vm\_spec

**Return type** class

**\_get\_vm\_provider** ()

**classmethod** **\_normalize\_spec** (*spec*)

**\_scp** (*\*\*kws*)

**\_template\_metadata** ()

**all\_ips** ()

**bootstrap** (*\*args, \*\*kwargs*)

Thin method that just uses the provider

**collect\_artifacts** (*host\_path, ignore\_nopath*)

**copy\_from** (*remote\_path, local\_path, recursive=True, propagate\_fail=True*)

**copy\_to** (*local\_path, remote\_path, recursive=True*)

**cpu\_model**

**cpu\_vendor**

**create\_snapshot** (*name, \*args, \*\*kwargs*)

Thin method that just uses the provider

**disks**

**distro** ()

**export\_disks** (*standalone=True, dst\_dir=None, compress=False, collect\_only=False, with\_threads=True, \*args, \*\*kwargs*)

Thin method that just uses the provider

**extract\_paths** (*paths, \*args, \*\*kwargs*)

Thin method that just uses the provider

**extract\_paths\_dead** (*paths, \*args, \*\*kwargs*)

Thin method that just uses the provider

**groups**

Returns –

**list of str: The names of the groups to which this vm belongs** (as specified in the init file)

**guest\_agent** ()

**has\_guest\_agent** ()

**interactive\_console** (*\*args, \*\*kwargs*)

Thin method that just uses the provider

**interactive\_ssh** (*\*args, \*\*kwargs*)

**ip** ()

**ips\_in\_net** (*net\_name*)

**iscsi\_name** ()

**metadata**

**mgmt\_name**

**mgmt\_net**

**name** ()

**nets** ()

**nics** ()

**reboot** (*\*args, \*\*kwargs*)

Thin method that just uses the provider

**revert\_snapshot** (*name, \*args, \*\*kwargs*)

Thin method that just uses the provider

**root\_password** ()

**running** (\*args, \*\*kwargs)

Thin method that just uses the provider

**save** (path=None)

**service** (\*args, \*\*kwargs)

**shutdown** (\*args, \*\*kwargs)

Thin method that just uses the provider

**spec**

**ssh** (command, data=None, show\_output=True, propagate\_fail=True, tries=None)

**ssh\_reachable** (tries=None, propagate\_fail=True)

Check if the VM is reachable with ssh

#### Parameters

- **tries** (*int*) – Number of tries to try connecting to the host
- **propagate\_fail** (*bool*) – If set to true, this event will appear
- **the log and fail the outter stage. Otherwise, it will be** (*in*) –
- **discarded.** –

**Returns** True if the VM is reachable.

**Return type** *bool*

**ssh\_script** (path, show\_output=True)

**start** (\*args, \*\*kwargs)

Thin method that just uses the provider

**state** (\*args, \*\*kwargs)

Thin method that just uses the provider

**stop** (\*args, \*\*kwargs)

Thin method that just uses the provider

**vm\_type**

**wait\_for\_ssh** ()

**class** lago.plugins.vm.VMProviderPlugin (vm)

Bases: *lago.plugins.Plugin*

If you want to use a custom provider for you VMs (say, ovirt for example), you have to inherit from this class, and then define the ‘default\_vm\_provider’ in your config to be your plugin, or explicitly specify it on each domain definition in the initfile with ‘vm-provider’ key

You will have to override at least all the abstractmethods in order to write a provider plugin, even if they are just runnig *pass*.

**\_extract\_paths\_scp** (\*args, \*\*kwargs)

**bootstrap** (\*args, \*\*kwargs)

Does any actions needed to get the domain ready to be used, ran on prefix init.

**Returns** None

**create\_snapshot** (name, \*args, \*\*kwargs)

Take any actions needed to create a snapshot

**Parameters** **name** (*str*) – Name for the snapshot, will be used as key to retrieve it later

**Returns** None

**export\_disks** (*standalone, dst\_dir, compress, \*args, \*\*kwargs*)

Export ‘disks’ as a standalone image or a layered image.

**Parameters**

- **disks** (*list*) – The names of the disks to export (None means all the disks)
- **standalone** (*bool*) – If true create a copy of the layered image else create a new disk which is a combination of the current layer and the base disk.
- **dst\_dir** (*str*) – dir to place the exported images
- **compress** (*bool*) – if true, compress the exported image.

**extract\_paths** (*paths, ignore\_nopath*)

Extract the given paths from the domain

**Parameters**

- **paths** (*list of str*) – paths to extract
- **ignore\_nopath** (*boolean*) – if True will ignore none existing paths.

**Returns** None

**Raises**

- *ExtractPathNoPathError* – if a none existing path was found on the VM, and *ignore\_nopath* is True.
- *ExtractPathError* – on all other failures.

**extract\_paths\_dead** (*paths, ignore\_nopath*)

Extract the given paths from the domain, without the underlying OS awareness

**interactive\_console** ()

Run an interactive console

**Returns** result of the interactive execution

**Return type** *lago.utils.CommandStatus*

**name** ()

**reboot** (*\*args, \*\*kwargs*)

Reboot a domain

**Returns** None

**revert\_snapshot** (*name, \*args, \*\*kwargs*)

Take any actions needed to revert/restore a snapshot

**Parameters** **name** (*str*) – Name for the snapshot, same that was set on creation

**Returns** None

**running** (*\*args, \*\*kwargs*)

**Returns** True if the VM is running

**Return type** (*bool*)

**shutdown** (*\*args, \*\*kwargs*)

Shutdown a domain



**Returns** None

**start** (\*args, \*\*kwargs)  
Start a domain

**Returns** None

**state** (\*args, \*\*kwargs)  
Return the current state of the domain

**Returns** Small description of the current domain state

**Return type** str

**stop** (\*args, \*\*kwargs)  
Stop a domain

**Returns** None

lago.plugins.vm.\_resolve\_service\_class(class\_name, service\_providers)

**NOTE:** This must be removed once the service\_class spec entry is fully deprecated

Retrieves a service plugin class from the class name instead of the provider name

**Parameters**

- **class\_name** (str) – Class name of the service plugin to retrieve
- **service\_providers** (dict) – provider\_name->provider\_class of the loaded service providers

**Returns** Class of the plugin that matches that name

**Return type** class

**Raises** `lago.plugins.NoSuchPluginError` – if there was no service plugin that matched the search

lago.plugins.vm.check\_running(func)

#### 4.1.1.2 lago.providers package

##### Subpackages

##### lago.providers.libvirt package

##### Submodules

##### lago.providers.libvirt.cpu module

**class** lago.providers.libvirt.cpu.CPU(spec, host\_cpu)

Bases: object

**cpu\_xml**

**generate\_cpu\_xml** ()  
Get CPU XML

**Returns** cpu node

**Return type** lxml.etree.Element

**generate\_custom** (*cpu*, *vcpu\_num*, *fill\_topology*)

Generate custom CPU model. This method attempts to convert the dict to XML, as defined by `xmldict.unparse` method.

**Parameters**

- **cpu** (*dict*) – CPU spec
- **vcpu\_num** (*int*) – number of virtual cpus
- **fill\_topology** (*bool*) – if topology is not defined in *cpu* and *vcpu* was not set, will add CPU topology to the generated CPU.

**Returns** CPU XML node

**Return type** `lxml.etree.Element`

**Raises** `LagoInitException` – when failed to convert dict to XML

**generate\_exact** (*model*, *vcpu\_num*, *host\_cpu*)

Generate exact CPU model with nested virtualization CPU feature.

**Parameters**

- **model** (*str*) – libvirt supported CPU model
- **vcpu\_num** (*int*) – number of virtual cpus
- **host\_cpu** (`lxml.etree.Element`) – the host CPU model

**Returns** CPU XML node

**Return type** `lxml.etree.Element`

**generate\_feature** (*name*, *policy*=*'require'*)

Generate CPU feature element

**Parameters**

- **name** (*str*) – feature name
- **policy** (*str*) – libvirt feature policy

**Returns** feature XML element

**Return type** `lxml.etree.Element`

**generate\_host\_passthrough** (*vcpu\_num*)

Generate host-passthrough XML cpu node

**Parameters** **vcpu\_num** (*str*) – number of virtual CPUs

**Returns** CPU XML node

**Return type** `lxml.etree.Element`

**generate\_numa** (*vcpu\_num*)

Generate guest CPU <numa> XML child Configures 1, 2 or 4 vCPUs per cell.

**Parameters** **vcpu\_num** (*str*) – number of virtual CPUs

**Returns** numa XML element

**Return type** `lxml.etree.Element`

**generate\_topology** (*vcpu\_num*, *cores*=*1*, *threads*=*1*)

Generate CPU <topology> XML child

**Parameters**

- **vcpu\_num** (*str*) – number of virtual CPUs
- **cores** (*int*) – number of cores
- **threads** (*int*) – number of threads

**Returns** topology XML element

**Return type** lxml.etree.Element

**generate\_vcpu** (*vcpu\_num*)

Generate <vcpu> domain XML child

**Parameters** **vcpu\_num** (*str*) – number of virtual cpus

**Returns** vcpu XML element

**Return type** lxml.etree.Element

**generate\_vcpu\_xml** (*vcpu\_num*)

**Parameters** **vcpu\_num** (*int*) – number of virtual cpus

**Returns** vcpu XML node

**Return type** lxml.etree.Element

**model**

**validate** ()

Validate CPU-related VM spec are compatible

**Raises** `LagoInitException` – if both ‘cpu\_model’ and ‘cpu\_custom’ are defined.

**vcpu\_xml**

**vendor**

**class** `lago.providers.libvirt.cpu.LibvirtCPU`

Bases: `object`

Query data from /usr/share/libvirt/cpu\_map.xml

**classmethod** **get\_cpu\_props** (*family*, *arch*=‘x86’)

Get CPU info XML

**Parameters**

- **family** (*str*) – CPU family
- **arch** (*str*) – CPU arch

**Returns** CPU xml

**Return type** lxml.etree.Element

**Raises** `LagoException` – If no such CPU family exists

**classmethod** **get\_cpu\_vendor** (*family*, *arch*=‘x86’)

Get CPU vendor, if vendor is not available will return ‘generic’

**Parameters**

- **family** (*str*) – CPU family
- **arch** (*str*) – CPU arch

**Returns** CPU vendor if found otherwise ‘generic’

**Return type** `str`

**classmethod** `get_cpus_by_arch (arch)`  
Get all CPUs info by arch

**Parameters** `arch (str)` – CPU architecture

**Returns** CPUs by arch XML

**Return type** `lxml.etree.element`

**Raises** `LagoException` – If no such ARCH is found

## lago.providers.libvirt.network module

**class** `lago.providers.libvirt.network.BridgeNetwork (env, spec, compat)`  
Bases: `lago.providers.libvirt.network.Network`

`_libvirt_xml ()`

**start ()**  
Start the network, will check if the network is active `attempts` times, waiting `timeout` between each attempt.

**Parameters**

- **attempts** (`int`) – number of attempts to check the network is active
- **timeout** (`int`) – timeout for each attempt

**Returns** `None`

**Raises**

- `RuntimeError` – if network creation failed, or failed to verify it is
- `active.`

**stop ()**

**class** `lago.providers.libvirt.network.NATNetwork (env, spec, compat)`  
Bases: `lago.providers.libvirt.network.Network`

`_generate_dns_disable ()`

`_generate_dns_forward (forward_ip)`

`_generate_main_dns (records, subnet, forward_plain='no')`

`_ipv6_prefix (subnet, const='fd8f:1391:3a82:')`

`_libvirt_xml ()`

**class** `lago.providers.libvirt.network.Network (env, spec, compat)`  
Bases: `object`

`_libvirt_name ()`

`_libvirt_xml ()`

**add\_mapping** (`name, ip, save=True`)

**add\_mappings** (`mappings`)

**alive ()**

**gw ()**

**is\_management ()**

```

mapping()
mtu()
name()
resolve(name)
save()
spec
start(attempts=5, timeout=2)
    Start the network, will check if the network is active attempts times, waiting timeout between each
    attempt.

    Parameters
        • attempts (int) – number of attempts to check the network is active
        • timeout (int) – timeout for each attempt

    Returns None

    Raises
        • RuntimeError – if network creation failed, or failed to verify it is
        • active.

stop()

```

## lago.providers.libvirt.utils module

Utilities to help deal with the libvirt python bindings

```
lago.providers.libvirt.utils.DOMAIN_STATES = {<sphinx.ext.autodoc.importer._MockObject obj
    Mapping of domain statuses values to human readable strings
```

```
class lago.providers.libvirt.utils.Domain
    Bases: object
```

Class to namespace libvirt domain related helpers

```
static resolve_state(state_number)
    Get a nice description from a domain state number
```

**Parameters** *state\_number* (*list of int*) – State number as returned by libvirt.  
virDomain.state()

### Returns

**small human readable description of the domain state, unknown** if the state is not in the  
known list

### Return type

`str`

```
lago.providers.libvirt.utils.LIBVIRT_CONNECTIONS = {}
    Singleton with the cached opened libvirt connections
```

```
lago.providers.libvirt.utils.auth_callback(credentials, user_data)
```

```
lago.providers.libvirt.utils.dict_to_xml(spec, full_document=False)
    Convert dict to XML
```

### Parameters

- **spec** (*dict*) – dict to convert
- **full\_document** (*bool*) – whether to add XML headers

**Returns** XML tree

**Return type** `lxml.etree.Element`

`lago.providers.libvirt.utils.get_domain_template(distro, libvirt_ver, **kwargs)`  
Get a rendered Jinja2 domain template

**Parameters**

- **distro** (*str*) – domain distro
- **libvirt\_ver** (*int*) – libvirt version
- **kwargs** (*dict*) – args for template render

**Returns** rendered template

**Return type** `str`

`lago.providers.libvirt.utils.get_libvirt_connection(name)`

`lago.providers.libvirt.utils.get_template(basename)`  
Load a file as a string from the templates directory

**Parameters** **basename** (*str*) – filename

**Returns** string representation of the file

**Return type** `str`

`lago.providers.libvirt.utils.libvirt_callback(userdata, err)`

## **lago.providers.libvirt.vm module**

**exception** `lago.providers.libvirt.vm.LagoLocalLibvirtVMProviderException`

Bases: `lago.utils.LagoException`

**class** `lago.providers.libvirt.vm.LocalLibvirtVMProvider(vm)`

Bases: `lago.plugins.vm.VMProviderPlugin`

`_createXML(dom_xml, flags=0)`

`_create_dead_snapshot(name)`

`_create_live_snapshot(name)`

`_get_domain()`

Return the object representation of this provider VM.

**Returns** Libvirt domain object

**Return type** `libvirt.virDomain`

**Raises** `exc:~lago.plugins.vm.LagoFailedToGetVMStateError`: If the VM exist, but the query returned an error.

`_get_qemu_kvm_path()`

`_libvirt_name()`

`_libvirt_xml()`

`_load_xml()`

**\_reclaim\_disk** (*path*)

**\_reclaim\_disks** ()

**\_shutdown** (*libvirt\_cmd*, *ssh\_cmd*, *msg*)

Choose the invoking method (using libvirt or ssh) to shutdown / poweroff the domain.

If acpi is defined in the domain use libvirt, otherwise use ssh.

**Parameters**

- **libvirt\_cmd** (*function*) – Libvirt function the invoke
- **ssh\_cmd** (*list of str*) – Shell command to invoke on the domain
- **msg** (*str*) – Name of the command that should be inserted to the log message.

**Returns** None

**Raises** `RuntimeError` – If acpi is not configured an ssh isn't available

**alive** ()

**bootstrap** ()

Does any actions needed to get the domain ready to be used, ran on prefix init.

**Returns** None

**caps**

**cpu**

**cpu\_model**

VM CPU model

**Returns** CPU model

**Return type** `str`

**cpu\_vendor**

VM CPU Vendor

**Returns** CPU vendor

**Return type** `str`

**create\_snapshot** (*name*)

Take any actions needed to create a snapshot

**Parameters** **name** (*str*) – Name for the snapshot, will be used as key to retrieve it later

**Returns** None

**export\_disks** (*standalone*, *dst\_dir*, *compress*, *collect\_only=False*, *with\_threads=True*, *\*args*, *\*\*kwargs*)

Export all the disks of self.

**Parameters**

- **standalone** (*bool*) – if true, merge the base images and the layered image into a new file (Supported only in qcow2 format)
- **dst\_dir** (*str*) – dir to place the exported disks
- **compress** (*bool*) – if true, compress each disk.

- **collect\_only** (*bool*) – If true, return only a dict which maps between the name of the vm to the paths of the disks that will be exported (don't export anything).
- **with\_threads** (*bool*) – If True, export disks in parallel

**Returns** which maps between the name of the vm to the paths of the disks that will be exported

**Return type** (*dict*)

**extract\_paths** (*paths, ignore\_nopath*)

Extract the given paths from the domain

Attempt to extract all files defined in *paths* with the method defined in `extract_paths()`, if it fails, and *guestfs* is available it will try extracting the files with *guestfs*.

**Parameters**

- **paths** (*list of tuples*) – files to extract in `[(src1, dst1), (src2, dst2)...]` format.
- **ignore\_nopath** (*boolean*) – if True will ignore none existing paths.

**Returns** None

**Raises**

- `ExtractPathNoPathError` – if a none existing path was found on the VM, and *ignore\_nopath* is False.
- `ExtractPathError` – on all other failures.

**extract\_paths\_dead** (*paths, ignore\_nopath*)

Extract the given paths from the domain using *guestfs*. Using *guestfs* can have side-effects and should be used as a second option, mainly when SSH is not available.

**Parameters**

- **paths** (*list of str*) – paths to extract
- **ignore\_nopath** (*boolean*) – if True will ignore none existing paths.

**Returns** None

**Raises**

- `LagoException` – if *guestfs* is not importable.
- `ExtractPathNoPathError` – if a none existing path was found on the VM, and *ignore\_nopath* is True.
- `ExtractPathError` – on failure extracting the files.

**interactive\_console** ()

Opens an interactive console

**Returns** result of the *virsh* command execution

**Return type** *lago.utils.CommandStatus*

**libvirt\_ver**

**raw\_state** ()

Return the state of the domain in Libvirt's terms

**Retruns:** tuple of ints: The state and its reason

**Raises**

- `LagoVMDoesNotExistError` – If the VM of this provider doesn't exist.



- exc:~lago.plugins.vm.LagoFailedToGetVMStateError: If the VM exist, but the query returned an error.

**reboot** (\*args, \*\*kwargs)

Reboot a domain

**Returns** None

**revert\_snapshot** (name)

Take any actions needed to revert/restore a snapshot

**Parameters** **name** (*str*) – Name for the snapshot, same that was set on creation

**Returns** None

**running** ()

Returns: (bool): True if the VM is running

**shutdown** (\*args, \*\*kwargs)

Shutdown a domain

**Returns** None

**start** ()

Start a domain

**Returns** None

**state** ()

Return a small description of the current status of the domain

**Returns** small description of the domain status, 'down' if it's not found at all.

**Return type** *str*

**stop** ()

Stop a domain

**Returns** None

## 4.1.2 Submodules

### 4.1.3 lago.brctl module

`lago.brctl._brctl (command, *args)`

`lago.brctl._set_link (name, state)`

`lago.brctl.create (name, stp=True)`

`lago.brctl.destroy (name)`

`lago.brctl.exists (name)`

### 4.1.4 lago.build module

**class** `lago.build.Build (name, disk_path, paths)`

Bases: `object`

A Build object represents a build section in the init file. Each build section (which in turn belongs to a specific disk) should get his own Build object

In order to add support for a new build command, a new function with the name of the command should be implemented in this class. this function should accept a list of options and arguments and return a named tuple 'Command', where 'Command.name' is the name of the command and 'Command.cmd' is the a list containing the command and its args, for example: Command.name = 'virt-customize' Command.cmd = ['virt-customize', '-a', PATH\_TO\_DISK, SOME\_CMDS...]

**name**

*str* – The name of the vm this builder belongs

**disk\_path**

*str* – The path to the disk that needs to be customized

**paths**

*lago.paths.Paths* – The paths of the current prefix

**build\_cmds**

*list of str* – A list of commands that should be invoked on the disk located in disk\_path

**build()**

Run all the commands in self.build\_cmds

**Raises** *lago.build.BuildException* – If a command returned a non-zero code

**get\_cmd\_handler(cmd)**

Return an handler for cmd. The handler and the command should have the same name. See class description for more info about handlers.

**Parameters** *cmd (str)* – The name of the command

**Returns** which handles cmd

**Return type** callable

**Raises** *lago.build.BuildException* – If an handler for cmd doesn't exist

**classmethod get\_instance\_from\_build\_spec(name, disk\_path, build\_spec, paths)****Parameters**

- **name** (*str*) – The name of the vm this builder belongs
- **disk\_path** (*str*) – The path to the disk that needs to be customized
- **paths** (*lago.paths.Paths*) – The paths of the current prefix
- **build\_spec** (*dict*) – The build spec part, associated with the disk located at disk\_path, from the init file.

**Returns**

An instance of Build with a normalized build spec i.e ready to be invoked.

**normalize\_build\_spec(build\_spec)**

Convert a build spec into a list of Command tuples. After running this command, self.build\_cmds should hold all the commands that should be run on the disk in self.disk\_path.

**Parameters** *build\_spec (dict)* – The builds spec part from the init file

**static normalize\_options(options)**

Turns a mapping of 'option: arg' to a list and prefix the options. arg can be a list of arguments.

for example:

```
dict = { o1: a1, o2: , o3: [a31, a32] o4: []
```

```
}

```

will be transformed to:

```
[ prefix_option(o1), a1, prefix_option(o2), prefix_option(o3), a31, prefix_option(o3), a32 pre-
  fix_option(o4)
]
```

note that empty arguments are omitted

**Parameters** `options` (*dict*) – A mapping between options and arguments

**Returns** A normalized version of ‘options’ as mentioned above

**Return type** `lst`

**static** `prefix_option` (*option*)

Depends on the option’s length, prefix it with ‘-’ or ‘--’:param option: The option to prefix :type option: str

**Returns** prefixed option

**Return type** `str`

**virt\_customize** (*options*)

Handler for ‘virt-customize’ note: if ‘ssh-inject’ option was specified without a path to a key, the prefix’ key will be copied to the vm.

**Parameters** `options` (*lst of str*) – Options and arguments for ‘virt-customize’

**Returns** which handles cmd

**Return type** callable

**Raises** `lago.build.BuildException` – If an handler for cmd doesn’t exist

**exception** `lago.build.BuildException`

Bases: `lago.utils.LagoException`

**class** `lago.build.Command` (*name, cmd*)

Bases: `tuple`

`__getnewargs__` ()

Return self as a plain tuple. Used by copy and pickle.

`__getstate__` ()

Exclude the OrderedDict from pickling

`__repr__` ()

Return a nicely formatted representation string

`_asdict` ()

Return a new OrderedDict which maps field names to their values

`_fields` = ('name', 'cmd')

**classmethod** `_make` (*iterable, new=<built-in method \_\_new\_\_ of type object at 0x906d60>, len=<built-in function len>*)

Make a new Command object from a sequence or iterable

`_replace` (*\*\*kws*)

Return a new Command object replacing specified fields with new values

`cmd`

Alias for field number 1

**name**

Alias for field number 0

### 4.1.5 lago.cmd module

`lago.cmd.create_parser(cli_plugins, out_plugins)`

`lago.cmd.exit_handler(signum, frame)`

Catch SIGTERM and SIGHUP and call “sys.exit” which raises “SystemExit” exception. This will trigger all the cleanup code defined in ContextManagers and “finally” statements.

For more details about the arguments see “signal” documentation.

#### Parameters

- **signum** (*int*) – The signal’s number
- **frame** (*frame*) – The current stack frame, can be None

`lago.cmd.main()`

### 4.1.6 lago.config module

**class** `lago.config.ConfigLoad` (*root\_section='lago', defaults={}*)

Bases: `object`

Merges configuration parameters from 3 different sources: 1. Enviornment vairables 2. config files in .INI format 3. argparse.ArgumentParser

The assumed order(but not necessary) order of calls is: load() - load from config files and environment variables update\_parser(parser) - update from the declared argparse parser update\_args(args) - update from passed arguments to the parser

`__getitem__` (*key*)

Get a variable from the default section, good for fail-fast if key does not exists.

**Parameters** **key** (*str*) – key

**Returns** config variable

**Return type** `str`

**get** (*\*args*)

Get a variable from the default section :param \*args: dict.get() args :type \*args: args

**Returns** config variable

**Return type** `str`

**get\_ini** (*incl\_unset=False*)

Return the config dictionary in INI format :param incl\_unset: include variables with no defaults. :type incl\_unset: bool

**Returns** string of the config file in INI format

**Return type** `str`

**get\_section** (*\*args*)

get a section dictionary Args:

**Returns** section config dictionary

**Return type** `dict`

**load()**

Load all configurations from available resources, skip if empty:

1. default` dict passed to `ConfigLoad.__init__()`.
2. **Custom paths as defined in `CONFS_PATH` in `constants`.**
3. XDG standard paths.
4. Environment variables.

**Returns** dict of dicts.

**Return type** `dict`

**update\_args(args)**

Update config dictionary with parsed args, as resolved by `argparse`. Only root positional arguments that already exist will be overridden.

**Parameters** `args` (`namespace`) – args parsed by `argparse`

**update\_parser(parser)**

Update config dictionary with declared arguments in an `argparse.parser`. New variables will be created, and existing ones overridden.

**Parameters** `parser` (`argparse.ArgumentParser`) – parser to read variables from

**lago.config.\_get\_configs\_path()**

Get a list of possible configuration files, from the following sources: 1. All files that exist in `constants.CONFS_PATH`. 2. All XDG standard config files for “lago.conf”, in reversed order of importance.

**Returns** list of files

**Return type** `list(str)`

**lago.config.get\_env\_dict(root\_section)**

Read all Lago variables from the environment. The lookup format is: `LAGO_VARNAME` - will land into ‘lago’ section `LAGO__SECTION1__VARNAME` - will land into ‘section1’ section, notice the double ‘\_\_’. `LAGO__LONG_SECTION_NAME__VARNAME` - will land into ‘long\_section\_name’

**Returns** dict of section configuration dicts

**Return type** `dict`

**Examples**

```
>>> os.environ['LAGO_GLOBAL_VAR'] = 'global'
>>> os.environ['LAGO__INIT__REPO_PATH'] = '/tmp/store'
>>>
>>> config.get_env_dict()
{'init': {'repo_path': '/tmp/store'}, 'lago': {'global_var': 'global'}}
```

**4.1.7 lago.constants module**

`lago.constants.CONFS_PATH = ['/etc/lago/lago.conf']`

`CONFS_PATH` - default path to first look for configuration files.

`lago.constants.LIBEXEC_DIR = '/usr/libexec/lago/'`

`LIBEXEC_DIR` -

### 4.1.8 lago.export module

**class** `lago.export.DiskExportManager` (*dst, disk\_type, disk, do\_compress*)

Bases: `object`

DiskExportManager object is responsible on the export process of an image from the current Lago prefix.

DiskExportManger is the base class of specific DiskExportManger. Each specific DiskExportManger is responsible on the export process of an image with a specific type (e.g template, file...)

**src**

*str* – Path to the image that should be exported

**name**

*str* – The name of the exported disk

**dst**

*str* – The absolute path of the exported disk

**disk\_type**

*str* – The type of the image e.g template, file, empty...

**disk**

*dict* – Disk attributes (of the disk that should be exported) as found in `workdir/current/virt/VM-NAME`

**exported\_metadata**

*dict* – A copy of the source disk metadata, this dict should be updated with new values during the export process.

**do\_compress**

*bool* – If true, apply compression to the exported disk.

**\_abc\_cache** = `<_weakrefset.WeakSet object>`

**\_abc\_negative\_cache** = `<_weakrefset.WeakSet object>`

**\_abc\_negative\_cache\_version** = 35

**\_abc\_registry** = `<_weakrefset.WeakSet object>`

**calc\_sha** (*checksum*)

Calculate the checksum of the new exported disk, write it to a file, and update this managers 'exported\_metadata'.

**Parameters** **checksum** (*str*) – The type of the checksum

**compress** ()

Compress the new exported image, Block size was taken from virt-builder page

**copy** ()

Copy the disk using cp in order to preserves the 'sparse' structure of the file

**export** ()

This method will handle the export process and should implemented in each subclass.

**static get\_instance\_by\_type** (*dst, disk, do\_compress, \*args, \*\*kwargs*)

**Parameters**

- **dst** (*str*) – The path of the new exported disk. can contain env variables.
- **disk** (*dict*) – Disk attributes (of the disk that should be exported) as found in `workdir/current/virt/VM-NAME`
- **do\_compress** (*bool*) – If true, apply compression to the exported disk.

**Returns** An instance of a subclass of DiskExportManager which matches the disk type.

**sparse()**

Make the exported images more compact by removing unused space. Please refer to ‘virt-sparsify’ for more info.

**update\_lago\_metadata()**

**write\_lago\_metadata()**

**class** lago.export.FileExportManager(*dst, disk\_type, disk, do\_compress, \*args, \*\*kwargs*)

Bases: [lago.export.DiskExportManager](#)

FileExportManager is responsible exporting images of type file and empty.

**standalone**

*bool* – If true, create a new image which is the result of merging all the layers of src (the image that we want to export).

**src\_qemu\_info**

*dict* – Metadata on src which was generated by qemu-img.

**\_abc\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache\_version** = 35

**\_abc\_registry** = <\_weakrefset.WeakSet object>

**export()**

See DiskExportManager.export

**class** lago.export.TemplateExportManager(*dst, disk\_type, disk, do\_compress, \*args, \*\*kwargs*)

Bases: [lago.export.DiskExportManager](#)

TemplateExportManager is responsible exporting images of type template.

**See superclass**

**\_abc\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache\_version** = 35

**\_abc\_registry** = <\_weakrefset.WeakSet object>

**export()**

See DiskExportManager.export

**rebase()**

Change the backing-file entry of the exported disk. Please refer to ‘qemu-img rebase’ manual for more info.

**update\_lago\_metadata()**

**class** lago.export.VMExportManager(*disks, dst, compress, with\_threads=True, \*args, \*\*kwargs*)

Bases: [object](#)

VMExportManager object is responsible on the export process of a list of disks.

**disks**

*list of dicts* – Disks to export.

**dst**

*str* – Where to place the exported disks.

**compress**

*bool* – If True compress each exported disk.

**with\_threads**

*bool* – If True, run the export in parallel

**\*args**

*list* – Extra args, will be passed to each DiskExportManager

**\*\*kwargs**

*dict* – Extra args, will be passed to each DiskExportManager

**\_collect()**

**Returns** The disks that needed to be exported

**Return type** (generator of dicts)

**\_get\_export\_mgr()**

**Returns** Handler for each disk

**Return type** (*DiskExportManager*)

**collect\_paths()**

**Returns** The path of the disks that will be exported.

**Return type** (list of str)

**export()**

Run the export process :returns: The path of the exported disks. :rtype: (list of str)

**exported\_disks\_paths()**

**Returns** The path of the exported disks.

**Return type** (list of str)

## 4.1.9 lago.guestfs\_tools module

**exception** `lago.guestfs_tools.GuestFSError`

Bases: `lago.utils.LagoException`

`lago.guestfs_tools._copy_path(conn, guest_path, host_path)`

`lago.guestfs_tools.extract_paths(disk_path, disk_root, paths, ignore_nopath)`

Extract paths from a disk using guestfs

**Parameters**

- **disk\_path** (*str*) – path to the disk
- **disk\_root** (*str*) – root partition
- **paths** (*list of tuples*) – files to extract in `[(src1, dst1), (src2, dst2)...]` format, if `srcN` is a directory in the guest, and `dstN` does not exist on the host, it will be created. If `srcN` is a file on the guest, it will be copied exactly to `dstN`
- **ignore\_nopath** (*bool*) – If set to True, ignore paths in the guest that do not exist

**Returns** None



**Raises**

- *ExtractPathNoPathError* – if a none existing path was found on the guest, and *ignore\_nopath* is False.
- *ExtractPathError* – on all other failures.

`lago.guestfs_tools.find_rootfs(conn, disk_root)`

Find the image's device root filesystem, and return its path.

1. Use `guestfs.GuestFS.inspect_os()` method. If it returns more than one root filesystem or None, try:
2. Find an exact match of *disk\_root* from `guestfs.GuestFS.list_filesystems()`, if none is found, try:
3. Return the device that has the substring *disk\_root* contained in it, from the output of `guestfs.GuestFS.list_filesystems()`.

**Parameters**

- **conn** (*guestfs.GuestFS*) – Open GuestFS handle.
- **disk\_root** (*str*) – Root device to search for. Note that by default, if *guestfs* can deduce the filesystem, it will not be used.

**Returns** root device path

**Return type** *str*

**Raises** *GuestFSError* if no root filesystem was found

`lago.guestfs_tools.guestfs_conn_mount_ro(*args, **kws)`

Open a GuestFS handle with *disk\_path* and try mounting the root filesystem. *disk\_root* is a hint where it should be looked and will only be used if GuestFS will not be able to deduce it independently.

Note that mounting a live guest, can lead to filesystem inconsistencies, causing the mount operation to fail. As we use readonly mode, this is safe, but the operation itself can still fail. Therefore, this method will watch for mount failures and retry 5 times before throwing an exception.

**Parameters**

- **disk\_path** (*str*) – Path to the disk.
- **disk\_root** (*str*) – Hint what is the root device with the OS filesystem.
- **retries** (*int*) – Number of retries for `mount_ro()` operation. Note that on each retry a new GuestFS handle will be used.
- **wait** (*int*) – Time to wait between retries.

**Yields** *guestfs.GuestFS* – An open GuestFS handle.

**Raises** *GuestFSError* – On any *guestfs* operation error, including exceeding retries for the `mount_ro()` operation.

`lago.guestfs_tools.guestfs_conn_ro(*args, **kws)`

Open a GuestFS handle and add *disk* in read only mode.

**Parameters** **disk** (*disk path*) – Path to the disk.

**Yields** *guestfs.GuestFS* – Open GuestFS handle

**Raises** *GuestFSError* – On any *guestfs* operation failure

### 4.1.10 lago.lago\_ansible module

**class** `lago.lago_ansible.LagoAnsible` (*prefix*)

Bases: `object`

A class for handling Ansible related tasks

**prefix**

*lago.prefix.Prefix* – The prefix that this object wraps

**`_generate_entry`** (*vm*)

Generate host entry for the given VM :param vm: The VM for which the entry should be created for.

**Returns** An entry for vm

**Return type** `str`

**`get_inventory`** (*keys=None*)

Create an Ansible inventory based on python dicts and lists. The returned value is a dict in which every key represents a group and every value is a list of entries for that group.

**Parameters** **keys** (*list of str*) – Path to the keys that will be used to create groups.

**Returns** dict based Ansible inventory

**Return type** `dict`

**`get_inventory_str`** (*keys=None*)

Convert a dict generated by `ansible.LagoAnsible.get_inventory` to an INI-like file.

**Parameters** **keys** (*list of str*) – Path to the keys that will be used to create groups.

**Returns** INI-like Ansible inventory

**Return type** `str`

**`get_inventory_temp_file`** (*\*\*kws*)

Context manager which returns the inventory written on a tempfile. The tempfile will be deleted as soon as this context manger ends.

**Parameters** **keys** (*list of str*) – Path to the keys that will be used to create groups.

**Yields** `tempfile.NamedTemporaryFile` – Temp file containing the inventory

**static `get_key`** (*key, data\_structure*)

Helper method for extracting values from a nested data structure.

**Parameters**

- **key** (*str*) – The path to the vales (a series of keys and indexes separated by '/')
- **data\_structure** (*dict or list*) – The data structure from which the value will be extracted.

**Returns** The values associated with key

**Return type** `str`

### 4.1.11 lago.log\_utils module

This module defines the special logging tools that lago uses

`lago.log_utils.ALWAYS_SHOW_REG = <_sre.SRE_Pattern object>`  
 Regexp that will match the above template

`lago.log_utils.ALWAYS_SHOW_TRIGGER_MSG = 'force-show:%s'`  
 Message template that will always show the message

**class** `lago.log_utils.ColorFormatter` (*fmt=None, datefmt=None*)

Bases: `logging.Formatter`

Formatter to add colors to log records

**CRITICAL** = `'\x1b[31m'`

**CYAN** = `'\x1b[36m'`

**DEBUG** = `''`

**DEFAULT** = `'\x1b[0m'`

**ERROR** = `'\x1b[31m'`

**GREEN** = `'\x1b[32m'`

**INFO** = `'\x1b[36m'`

**NONE** = `''`

**RED** = `'\x1b[31m'`

**WARNING** = `'\x1b[33m'`

**WHITE** = `'\x1b[37m'`

**YELLOW** = `'\x1b[33m'`

**classmethod** `colored` (*color, message*)

Small function to wrap a string around a color

**Parameters**

- **color** (*str*) – name of the color to wrap the string with, must be one of the class properties
- **message** (*str*) – String to wrap with the color

**Returns** the colored string

**Return type** `str`

**format** (*record*)

Adds colors to a log record and formats it with the default

**Parameters** **record** (*logging.LogRecord*) – log record to format

**Returns** The colored and formatted record string

**Return type** `str`

**class** `lago.log_utils.ContextLock`

Bases: `object`

Context manager to thread lock a block of code

```
lago.log_utils.END_TASK_MSG = 'Success'
```

Message to be shown when a task is ended

```
lago.log_utils.END_TASK_REG = <_sre.SRE_Pattern object>
```

Regexp that will match the above template

```
lago.log_utils.END_TASK_TRIGGER_MSG = 'end task%s'
```

Message template that will trigger a task end

```
class lago.log_utils.LogTask(task, logger=<module 'logging' from
                               '/usr/lib/python2.7/logging/__init__.pyc'>, level='info', propagate_fail=True, uuid=None)
```

Bases: `object`

Context manager for a log task

### Example

```
>>> with LogTask('mytask') :
...     pass
```

```
lago.log_utils.START_TASK_MSG = ''
```

Message to be shown when a task is started

```
lago.log_utils.START_TASK_REG = <_sre.SRE_Pattern object>
```

Regexp that will match the above template

```
lago.log_utils.START_TASK_TRIGGER_MSG = 'start task%s'
```

Message template that will trigger a task

```
class lago.log_utils.Task(name, *args, **kwargs)
```

Bases: `collections.deque`

Small wrapper around deque to add the failed status and name to a task

**name**

*str* – name for this task

**failed**

*bool* – If this task has failed or not (if there was any error log shown during it's execution)

**force\_show**

*bool* – If set, will show any log records generated inside this task even if it's out of nested depth limit

**elapsed\_time()**

```
class lago.log_utils.TaskHandler(initial_depth=0, task_tree_depth=-1, buffer_size=2000,
                                  dump_level=40, level=0, formatter=<class
                                  'lago.log_utils.ColorFormatter'>)
```

Bases: `logging.StreamHandler`

This log handler will use the concept of tasks, to hide logs, and will show all the logs for the current task if there's a logged error while running that task.

It will hide any logs that belong to nested tasks that have more than `task_tree_depth` parent levels, and for the ones that are above that level, it will show only the logs that have a loglevel above `level`.

You can force showing a log record immediately if you use the `log_always()` function bypassing all the filters.

If there's a log record with log level higher than `dump_level` it will be considered a failure, and all the logs for the current task that have a log level above `level` will be shown no matter at which depth the task belongs to. Also, all the parent tasks will be tagged as error.

**formatter**

*logging.LogFormatter* – formatter to use

**initial\_depth**

*int* – Initial depth to account for, in case this handler was created in a subtask

**tasks\_by\_thread (dict of str**

OrderedDict of str: Task): List of thread names, and their currently open tasks with their latest log records

**dump\_level**

*int* – log level from which to consider a log record as error

**buffer\_size**

*int* – Size of the log record deque for each task, the bigger, the more records it can show in case of error but the more memory it will use

**task\_tree\_depth**

*int* – number of the nested level to show start/end task logs for, if -1 will show always

**level**

*int* – Log level to show logs from if the depth limit is not reached

**main\_failed**

*bool* – used to flag from a child thread that the main should fail any current task

**\_tasks\_lock**

*ContextLock* – Lock for the tasks\_by\_thread dict

**\_main\_thread\_lock**

*ContextLock* – Lock for the main\_failed bool

**TASK\_INDICATORS = ['@', '#', '\*', '-', '~']**

List of chars to show as task prefix, to ease distinguishing them

**am\_i\_main\_thread**

**\*\*Returns\*** – *bool* – if the current thread is the main thread

**close\_children\_tasks (parent\_task\_name)**

Closes all the children tasks that were open

**Parameters** *parent\_task\_name* (*str*) – Name of the parent task

**Returns** *None*

**cur\_depth\_level**

**\*\*Returns\*** – *int* – depth level for the current task

**cur\_task**

**\*\*Returns\*** – *str* – the current active task

**cur\_thread**

**\*\*Returns\*** – *str* – Name of the current thread

**emit (record)**

Handle the given record, this is the entry point from the python logging facility

**Params:** *record* (*logging.LogRecord*): log record to handle

**Returns** *None*

**get\_task\_indicator** (*task\_level=None*)

**Parameters** **task\_level** (*int* or *None*) – task depth level to get the indicator for, if *None*, will use the current tasks depth

**Returns** char to prepend to the task logs to indicate it's level

**Return type** *str*

**get\_tasks** (*thread\_name*)

**Parameters** **thread\_name** (*str*) – name of the thread to get the tasks for

**Returns**

list of task names and log records for each for the given thread

**Return type** OrderedDict of *str*, *Task*

**handle\_closed\_task** (*task\_name, record*)

Do everything needed when a task is closed

**Params:** **task\_name** (*str*): name of the task that is finishing record (*logging.LogRecord*): log record with all the info

**Returns** *None*

**handle\_error** ()

Handles an error log record that should be shown

**Returns** *None*

**handle\_new\_task** (*task\_name, record*)

Do everything needed when a task is starting

**Params:** **task\_name** (*str*): name of the task that is starting record (*logging.LogRecord*): log record with all the info

**Returns** *None*

**mark\_main\_tasks\_as\_failed** ()

Flags to the main thread that all it's tasks should fail

**Returns** *None*

**mark\_parent\_tasks\_as\_failed** (*task\_name, flush\_logs=False*)

Marks all the parent tasks as failed

**Parameters**

- **task\_name** (*str*) – Name of the child task
- **flush\_logs** (*bool*) – If *True* will discard all the logs from parent tasks

**Returns** *None*

**pretty\_emit** (*record, is\_header=False, task\_level=None*)

Wrapper around the *logging.StreamHandler* emit method to add some decoration stuff to the message

**Parameters**

- **record** (*logging.LogRecord*) – log record to emit
- **is\_header** (*bool*) – if this record is a header, usually, a start or end task message

- **task\_level** (*int*) – If passed, will take that as the current nested task level instead of calculating it from the current tasks

**Returns** None

**should\_show\_by\_depth** (*cur\_level=None*)

**Parameters** **cur\_level** (*int*) – depth level to take into account

**Returns**

True if the given depth level should show messages (not taking into account the log level)

**Return type** bool

**should\_show\_by\_level** (*record\_level, base\_level=None*)

**Parameters**

- **record\_level** (*int*) – log level of the record to check
- **base\_level** (*int or None*) – log level to check against, will use the object's *dump\_level* if None is passed

**Returns**

True if the given log record should be shown according to the log level

**Return type** bool

**tasks**

*Returns –*

OrderedDict of str, Task: list of task names and log records for each for the current thread

`lago.log_utils.end_log_task(task, logger=<module 'logging' from  
'/usr/lib/python2.7/logging/__init__.pyc'>, level='info')`

Ends a log task

**Parameters**

- **task** (*str*) – name of the log task to end
- **logger** (*logging.Logger*) – logger to use
- **level** (*str*) – log level to use

**Returns** None

`lago.log_utils.get_default_log_formatter()`

`lago.log_utils.hide_paramiko_logs()`

`lago.log_utils.hide_stevedore_logs()`

Hides the logs of stevedore, this function was added in order to support older versions of stevedore

We are using the NullHandler in order to get rid from 'No handlers could be found for logger...' msg

**Returns** None

`lago.log_utils.log_always(message)`

Wraps the given message with a tag that will make it be always logged by the task logger

**Parameters** **message** (*str*) – message to wrap with the tag

**Returns**

tagged message that will get it shown immediately by the task logger

**Return type** str

```
lago.log_utils.log_task(task, logger=<module 'logging' from
                        '/usr/lib/python2.7/logging/__init__.pyc'>,
                        level='info',
                        gate_fail=True, uuid=None)
Parameterized decorator to wrap a function in a log task
```

### Example

```
>>> @log_task('mytask')
... def do_something():
...     pass
```

```
lago.log_utils.setup_prefix_logging(logdir)
Sets up a file logger that will create a log in the given logdir (usually a lago prefix)
```

**Parameters** `logdir` (*str*) – path to create the log into, will be created if it does not exist

**Returns** None

```
lago.log_utils.start_log_task(task, logger=<module 'logging' from
                                '/usr/lib/python2.7/logging/__init__.pyc'>, level='info')
```

Starts a log task

**Parameters**

- **task** (*str*) – name of the log task to start
- **logger** (*logging.Logger*) – logger to use
- **level** (*str*) – log level to use

**Returns** None

## 4.1.12 lago.paths module

```
class lago.paths.Paths(prefix_path)
```

Bases: `object`

A Paths object contains methods for getting the paths to the directories and files which composes the prefix.

**`_prefix_path`**

*str* – Path to the directory of the prefix

**`images`** (*\*path*)

**`logs`** ()

**`metadata`** ()

**`prefix_lagofile`** ()

This file represents a prefix that's initialized

**`prefix_path`** ()

**`prefixed`** (*\*args*)

**`scripts`** (*\*args*)

**`ssh_id_rsa`** ()

**`ssh_id_rsa_pub`** ()

**`uuid`** ()



**virt** (\*path)

### 4.1.13 lago.prefix module

**class** lago.prefix.**Prefix** (prefix)

Bases: `object`

A prefix is a directory that will contain all the data needed to setup the environment.

**\_paths**

*lago.path.Paths* – Paths handler class

**\_virt\_env**

*lago.virt.VirtEnv* – Lazily loaded virtual env handler

**\_metadata**

*dict* – Lazily loaded metadata

**VIRT\_ENV\_CLASS**

alias of *lago.virt.VirtEnv*

**\_add\_dns\_records** (conf, mgmts)

Add DNS records dict('dns\_records') to `conf` for each management network. Add DNS forwarder IP('dns\_forward') for each none management network.

**Parameters**

- **conf** (*spec*) – spec
- **mgmts** (*list*) – management networks names

**Returns** None

**\_add\_mgmt\_to\_domains** (conf, mgmts)

Add management network key('mgmt\_net') to each domain. Note this assumes `conf` was validated.

**Parameters**

- **conf** (*dict*) – spec
- **mgmts** (*list*) – list of management networks names

**\_add\_nic\_to\_mapping** (net, dom, nic)

Populates the given net spec mapping entry with the nics of the given domain, by the following rules:

- If `net` is management, 'domain\_name': `nic_ip`
- For each interface: 'domain\_name-eth#': `nic_ip`, where `#` is the

index of the nic in the *domain* definition. \* For each interface: 'domain\_name-net\_name-#': `nic_ip`, where `#` is a running number of interfaces from that network. \* For each interface: 'domain\_name-net\_name', which has an identical IP to 'domain\_name-net\_name-0'

**Parameters**

- **net** (*dict*) – Network spec to populate
- **dom** (*dict*) – libvirt domain specification
- **nic** (*str*) – Name of the interface to add to the net mapping from the domain

**Returns** None

**`_allocate_ips_to_nics`** (*conf*)

For all the nics of all the domains in the conf that have dynamic ip, allocate one and addit to the network mapping

**Parameters** **conf** (*dict*) – Configuration spec to extract the domains from

**Returns** None

**`_allocate_subnets`** (*conf*)

Allocate all the subnets needed by the given configuration spec

**Parameters** **conf** (*dict*) – Configuration spec where to get the nets definitions from

**Returns** allocated subnets and modified conf

**Return type** `tuple(list, dict)`

**`_config_net_topology`** (*conf*)

Initialize and populate all the network related elements, like reserving ips and populating network specs of the given configuration spec

**Parameters** **conf** (*dict*) – Configuration spec to initialize

**Returns** None

**`_copy_delpoy_scripts`** (*scripts*)

Copy the given deploy scripts to the scripts dir in the prefix

**Parameters** **scripts** (*list of str*) – list of paths of the scripts to copy to the prefix

**Returns**

**list with the paths to the copied scripts, with a** prefixed with \$LAGO\_PREFIX\_PATH so the full path is not hardcoded

**Return type** list of str

**`_copy_deploy_scripts_for_hosts`** (*domains*)

Copy the deploy scripts for all the domains into the prefix scripts dir

**Parameters** **domains** (*dict*) – spec with the domains info as when loaded from the initfile

**Returns** None

**`_create_disk`** (*name, spec, template\_repo=None, template\_store=None*)

Creates a disc with the given name from the given repo or store

**Parameters**

- **name** (*str*) – Name of the domain to create the disk for
- **spec** (*dict*) – Specification of the disk to create
- **template\_repo** (*TemplateRepository or None*) – template repo instance to use
- **template\_store** (*TemplateStore or None*) – template store instance to use

**Returns** Path to the disk and disk metadata

**Return type** `Tuple(str, dict)`

**Raises** `RuntimeError` – If the type of the disk is not supported or failed to create the disk

**`_create_disks`** (*domain\_name, disks\_specs, template\_repo, template\_store*)

**`_create_link_to_parent`** (*base, link\_name*)

**`_create_ssh_keys()`**  
 Generate a pair of ssh keys for this prefix

**Returns** None

**Raises** `RuntimeError` – if it fails to create the keys

**`_create_virt_env()`**  
 Create a new virt env from this prefix

**Returns** virt env created from this prefix

**Return type** `lago.virt.VirtEnv`

**`_deploy_host(host)`**

**`static _generate_disk_name(host_name, disk_name, disk_format)`**

**`_generate_disk_path(disk_name)`**

**`_get_net(conf, dom_name, nic)`**

**`_get_scripts(host_metadata)`**  
 Temporary method to retrieve the host scripts

**Parameters** `host_metadata(dict)` – host metadata to retrieve the scripts for

**Returns** deploy scripts for the host, empty if none found

**Return type** `list`

**`_handle_empty_disk(host_name, disk_spec)`**

**`_handle_file_disk(disk_spec)`**

**`_handle_lago_template(disk_path, template_spec, template_store, template_repo)`**

**`_handle_ova_image(domain_spec)`**

**`_handle_qcow_template(disk_path, template_spec, template_store=None, template_repo=None)`**

**`_handle_template(host_name, template_spec, template_store=None, template_repo=None)`**

**`static _init_net_specs(conf)`**  
 Given a configuration specification, initializes all the net definitions in it so they can be used comfortably

**Parameters** `conf(dict)` – Configuration specification

**Returns** the adapted new conf

**Return type** `dict`

**`_ova_to_spec(filename)`**  
 Retrieve the given ova and makes a template of it. Creates a disk from network provided ova. Calculates the needed memory from the ovf. The disk will be cached in the template repo

**Parameters** `filename(str)` – the url to retrieve the data from

**Returns** list with the disk specification int: VM memory, None if none defined int: Number of virtual cpus, None if none defined

**Return type** `list of dict`

**Raises**

- `RuntimeError` – If the ova format is not supported

- `TypeError` – If the memory units in the ova are not supported (currently only ‘MegaBytes’)

`_prepare_domain_image` (*domain\_spec, prototypes, template\_repo, template\_store*)

`_prepare_domains_images` (*conf, template\_repo, template\_store*)

`_register_preallocated_ips` (*conf*)

Parse all the domains in the given conf and preallocate all their ips into the networks mappings, raising exception on duplicated ips or ips out of the allowed ranges

See also:

`lago.subnet_lease`

**Parameters** `conf` (*dict*) – Configuration spec to parse

**Returns** None

**Raises** `RuntimeError` – if there are any duplicated ips or any ip out of the allowed range

`_retrieve_disk_url` (*disk\_url, disk\_dst\_path=None*)

`static _run_qemu` (*qemu\_cmd, disk\_path*)

`_save_metadata` ()

Write this prefix metadata to disk

**Returns** None

`_select_mgmt_networks` (*conf*)

Select management networks. If no management network is found, it will mark the first network found by sorted the network lists. Also adding default DNS domain, if none is set.

**Parameters** `conf` (*spec*) – spec

`_set_mtu_to_nics` (*conf*)

For all the nics of all the domains in the conf that have MTU set, save the MTU on the NIC definition.

**Parameters** `conf` (*dict*) – Configuration spec to extract the domains from

**Returns** None

`_set_scripts` (*host\_metadata, scripts*)

Temporary method to set the host scripts

**Parameters** `host_metadata` (*dict*) – host metadata to set scripts in

**Returns** the updated metadata

**Return type** `dict`

`_use_prototype` (*spec, prototypes*)

Populates the given spec with the values of its declared prototype

**Parameters**

- `spec` (*dict*) – spec to update
- `prototypes` (*dict*) – Configuration spec containing the prototypes

**Returns** updated spec

**Return type** `dict`

**`_validate_netconfig`** (*conf*)

Validate network configuration

**Parameters** **`conf`** (*dict*) – spec

**Returns** None

**Raises**

- *LagoInitException* – If a VM has more than
- one management network configured, or a network which is not
- management has DNS attributes, or a VM is configured with a
- none-existence NIC, or a VM has no management network.

**`build`** (*conf*)

**`cleanup`** (*\*\*kwargs*)

Stops any running entities in the prefix and uninitializes it, usually you want to do this if you are going to remove the prefix afterwards

**Returns** None

**`collect_artifacts`** (*\*\*kwargs*)

**`create_snapshots`** (*name*)

Creates one snapshot on all the domains with the given name

**Parameters** **`name`** (*str*) – Name of the snapshots to create

**Returns** None

**`deploy`** (*\*\*kwargs*)

**`destroy`** ()

Destroy this prefix, running any cleanups and removing any files inside it.

**`export_vms`** (*\*\*kwargs*)

Export vm images disks and init file. The exported images and init file can be used to recreate the environment.

**Parameters**

- **`vms_names`** (*list of str*) – Names of the vms to export, if None export all the vms in the env (default=None)
- **`standalone`** (*bool*) – If false, export a layered image (default=False)
- **`export_dir`** (*str*) – Dir to place the exported images and init file
- **`compress`** (*bool*) – If True compress the images with xz (default=False)
- **`init_file_name`** (*str*) – The name of the exported init file (default='LagoInitfile')
- **`out_format`** (*lago.plugins.output.OutFormatPlugin*) – The type of the exported init file (the default is yaml)
- **`collect_only`** (*bool*) – If True, return only a mapping from vm name to the disks that will be exported. (default=False)
- **`with_threads`** (*bool*) – If True, run the export in parallel (default=True)

**Returns** Unless `collect_only == True`, a mapping between vms' disks.

**fetch\_url** (*url*)

Retrieves the given url to the prefix

**Parameters** **url** (*str*) – Url to retrieve

**Returns** path to the downloaded file

**Return type** *str*

**get\_nets** (*\*\*kwargs*)

Retrieve info on all the nets from all the domains

**Returns** dictionary with net\_name -> net list

**Return type** dict of str->list(*str*)

**get\_paths** (*\*\*kwargs*)

Get the Paths object of this prefix. The Paths object contains the paths to the internal directories and files of this prefix.

**Returns** The Paths object of this prefix

**Return type** *lago.paths.Paths*

**get\_snapshots** ()

Retrieve info on all the snapshots from all the domains

**Returns** list(*str*): dictionary with vm\_name -> snapshot list

**Return type** dict of str

**get\_vms** (*\*\*kwargs*)

Retrieve info on all the vms

**Returns** dictionary with vm\_name -> vm list

**Return type** dict of str->list(*str*)

**initialize** (*\*\*kwargs*)

Initialize this prefix, this includes creating the destination path, and creating the uuid for the prefix, for any other actions see *Prefix.virt\_conf()*

Will safely roll back if any of those steps fail

**Returns** None

**Raises** *RuntimeError* – If it fails to create the prefix dir

**classmethod is\_prefix** (*path*)

Check if a path is a valid prefix

**Parameters** **path** (*str*) – path to be checked

**Returns** True if the given path is a prefix

**Return type** *bool*

**metadata**

Retrieve the metadata info for this prefix

**Returns** metadata info

**Return type** *dict*

**paths**

**resolve\_parent** (*disk\_path*, *template\_store*, *template\_repo*)

Given a virtual disk, checks if it has a backing file, if so check if the backing file is in the store, if not download it from the provided *template\_repo*.

After verifying that the backing-file is in the store, create a symlink to that file and locate it near the layered image.

**Parameters**

- **disk\_path** (*str*) – path to the layered disk
- **template\_repo** (*TemplateRepository* or *None*) – template repo instance to use
- **template\_store** (*TemplateStore* or *None*) – template store instance to use

**classmethod resolve\_prefix\_path** (*start\_path=None*)

Look for an existing prefix in the given path, in a *path/.lago* dir, or in a *.lago* dir under any of it's parent directories

**Parameters** **start\_path** (*str*) – path to start the search from, if *None* passed, it will use the current dir

**Returns** path to the found prefix

**Return type** *str*

**Raises** *RuntimeError* – if no prefix was found

**revert\_snapshots** (*name*)

Revert all the snapshots with the given name from all the domains

**Parameters** **name** (*str*) – Name of the snapshots to revert

**Returns** *None*

**save** ()

Save this prefix to persistent storage

**Returns** *None*

**shutdown** (*\*\*kwargs*)

Shutdown this prefix

**Parameters**

- **vm\_names** (*list of str*) – List of the vms to shutdown
- **reboot** (*bool*) – If true, reboot the requested vms

**Returns** *None*

**start** (*\*\*kwargs*)

Start this prefix

**Parameters** **vm\_names** (*list of str*) – List of the vms to start

**Returns** *None*

**stop** (*\*\*kwargs*)

Stop this prefix

**Parameters** **vm\_names** (*list of str*) – List of the vms to stop

**Returns** *None*

**virt\_conf** (*conf*, *template\_repo=None*, *template\_store=None*, *do\_bootstrap=True*, *do\_build=True*)

Initializes all the virt infrastructure of the prefix, creating the domains disks, doing any network leases and creating all the virt related files and dirs inside this prefix.

**Parameters**

- **conf** (*dict*) – Configuration spec
- **template\_repo** (*TemplateRepository*) – template repository instance
- **template\_store** (*TemplateStore*) – template store instance
- **do\_bootstrap** (*bool*) – If true run virt-sysprep on the images
- **do\_build** (*bool*) – If true run build commands on the images, see lago.build.py for more info.

**Returns** None

**virt\_conf\_from\_stream** (*conf\_fd*, *template\_repo=None*, *template\_store=None*, *do\_bootstrap=True*, *do\_build=True*)

Initializes all the virt infrastructure of the prefix, creating the domains disks, doing any network leases and creating all the virt related files and dirs inside this prefix.

**Parameters**

- **conf\_fd** (*File*) – File like object to read the config from
- **template\_repo** (*TemplateRepository*) – template repository instance
- **template\_store** (*TemplateStore*) – template store instance

**Returns** None

**virt\_env**

Getter for this instance's virt env, creates it if needed

**Returns** virt env instance used by this prefix

**Return type** *lago.virt.VirtEnv*

**lago.prefix.\_create\_ip** (*subnet*, *index*)

Given a subnet or an ip and an index returns the ip with that lower index from the subnet (255.255.255.0 mask only subnets)

**Parameters**

- **subnet** (*str*) – Strign containing the three first elements of the decimal representation of a subnet (X.Y.Z) or a full ip (X.Y.Z.A)
- **index** (*int* or *str*) – Last element of a decimal ip representation, for example, 123 for the ip 1.2.3.123

**Returns** The dotted decimal representation of the ip

**Return type** *str*

**lago.prefix.\_ip\_in\_subnet** (*subnet*, *ip*)

Checks if an ip is included in a subnet.

---

**Note:** only 255.255.255.0 masks allowed

---

**Parameters**



- **subnet** (*str*) – Strign containing the three first elements of the decimal representation of a subnet (X.Y.Z) or a full ip (X.Y.Z.A)
- **ip** (*str* or *int*) – Decimal ip representation

**Returns** True if ip is in subnet, False otherwise

**Return type** bool

#### 4.1.14 lago.sdk module

**class** `lago.sdk.SDK` (*workdir*, *prefix*)

Bases: `object`

The SDK can be initialized in 3 ways:

1. (Preferred) - by calling `sdk.init()`.
2. By loading an existing workdir from the disk, with `load_env()`.
3. By passing already created workdir and prefix objects.

**ansible\_inventory** (*keys*=[*'vm-type'*, *'groups'*, *'vm-provider'*])

Get an Ansible inventory as a string, *keys* should be list on which to group the hosts by. You can use any key defined in LagoInitFile.

Examples of possible *keys*:

*keys*=[*'disks/0/metadata/arch'*], would group the hosts by the architecture.

*keys*=[*'disks/0/metadata/distro'*, *'disks/0/metadata/arch'*], would create groups by architecture and also by distro.

*keys*=[*'groups'*] - would group hosts by the groups defined for each VM in the LagoInitFile, i.e.:

domains:

**vm-01:** ... groups: web-server ..

**vm-02:** groups: db-server

**Parameters** *keys* (*list of str*) – Path to the keys that will be used to create groups.

**Returns** INI-like Ansible inventory

**Return type** str

**ansible\_inventory\_temp\_file** (*keys*=[*'vm-type'*, *'groups'*, *'vm-provider'*])

Context manager which returns Ansible inventory written on a tempfile. This is the same as `ansible_inventory()`, only the inventory file is written to a tempfile.

**Parameters** *keys* (*list of str*) – Path to the keys that will be used to create groups.

**Yields** `tempfile.NamedTemporaryFile` – Temp file containing the inventory

**destroy** ()

Destroy the environment, this will terminate all resources, and remove entirely the Lago working directory.

`lago.sdk.add_stream_logger` (*level*=10, *name*=None)

Add a stream logger. This can be used for printing all SDK calls to stdout while working in an interactive session. Note this is a logger for the entire module, which will apply to all environments started in the same session. If you need a specific logger pass a `logfile` to `init()`

#### Parameters

- **level** (*int*) – logging log level
- **name** (*str*) – logger name, will default to the root logger.

**Returns** None

`lago.sdk.init (config, workdir=None, logfile=None, loglevel=20, **kwargs)`  
Initialize the Lago environment

#### Parameters

- **config** (*str*) – Path to LagoInitFile
- **workdir** (*str*) – Path to initialize the workdir, defaults to “\$PWD/.lago”
- **\*\*kwargs** (*dict*) – Pass arguments to `do_init()`
- **logfile** (*str*) – A path to setup a log file.
- **loglevel** (*int*) – logging log level.

**Returns** Initialized Lago environment

**Return type** *SDK*

**Raises** *LagoException* – If initialization failed

`lago.sdk.load_env (workdir, logfile=None, loglevel=20)`  
Load an existing Lago environment

**Parameters** **workdir** (*str*) – Path to the workdir directory, as created by

:param *init()* or created by the CLI.: :param logfile: A Path to setup a log file. :type logfile: str :param loglevel: logging log level. :type loglevel: int

**Returns** Initialized Lago environment

**Return type** *SDK*

**Raises** *LagoException* – If loading the environment failed.

### 4.1.15 lago.sdk\_utils module

**class** `lago.sdk_utils.SDKMethod (name)`

Bases: *object*

Metadata to store inside the decorated function

`lago.sdk_utils.expose (func)`

Decorator to be used with *SDKWrapper*. This decorator indicates that the wrapped method or class should be exposed in the proxied object.

**Parameters** **func** (*types.FunctionType/types.MethodType*) – function to decorate

**Returns** None

`lago.sdk_utils.getattr_sdk (attr, name)`

Filter SDK attributes

#### Parameters

- **attr** (*attribute*) – Attribute as returned by `getattr()`.
- **name** (*str*) – Attribute name.

**Returns** *attr* if passed.

`lago.sdk_utils.setup_sdk_logging(logfile=None, loglevel=20)`

Setup a NullHandler to the root logger. If `logfile` is passed, additionally add a `FileHandler` in `loglevel` level.

**Parameters**

- **logfile** (*str*) – A path to setup a log file.
- **loglevel** (*int*) – logging log level.

**Returns** None

#### 4.1.16 lago.service module

**class** `lago.service.SysVInitService` (*vm, name*)

Bases: `lago.plugins.service.ServicePlugin`

**BIN\_PATH** =  `'/sbin/service '`

**\_abc\_cache** = `<_weakrefset.WeakSet object>`

**\_abc\_negative\_cache** = `<_weakrefset.WeakSet object>`

**\_abc\_negative\_cache\_version** = `35`

**\_abc\_registry** = `<_weakrefset.WeakSet object>`

**\_request\_start** ()

Low level implementation of the service start request, used by the *func: start* method

**Returns** True if the service succeeded to start, False otherwise

**Return type** `bool`

**\_request\_stop** ()

Low level implementation of the service stop request, used by the *func: stop* method

**Returns** True if the service succeeded to stop, False otherwise

**Return type** `bool`

**state** ()

Check the current status of the service

**Returns** Which state the service is at right now

**Return type** `ServiceState`

**class** `lago.service.SystemdContainerService` (*vm, name*)

Bases: `lago.plugins.service.ServicePlugin`

**BIN\_PATH** =  `'/usr/bin/docker '`

**HOST\_BIN\_PATH** =  `'/usr/bin/systemctl '`

**\_abc\_cache** = `<_weakrefset.WeakSet object>`

**\_abc\_negative\_cache** = `<_weakrefset.WeakSet object>`

**\_abc\_negative\_cache\_version** = `35`

**\_abc\_registry** = `<_weakrefset.WeakSet object>`

**\_request\_start** ()

Low level implementation of the service start request, used by the *func: start* method

**Returns** True if the service succeeded to start, False otherwise

**Return type** `bool`

`_request_stop()`

Low level implementation of the service stop request, used by the *func:stop* method

**Returns** True if the service succeeded to stop, False otherwise

**Return type** `bool`

`state()`

Check the current status of the service

**Returns** Which state the service is at right now

**Return type** `ServiceState`

**class** `lago.service.SystemdService` (*vm, name*)

Bases: `lago.plugins.service.ServicePlugin`

`BIN_PATH = '/usr/bin/systemctl'`

`_abc_cache = <weakrefset.WeakSet object>`

`_abc_negative_cache = <weakrefset.WeakSet object>`

`_abc_negative_cache_version = 35`

`_abc_registry = <weakrefset.WeakSet object>`

`_request_start()`

Low level implementation of the service start request, used by the *func:start* method

**Returns** True if the service succeeded to start, False otherwise

**Return type** `bool`

`_request_stop()`

Low level implementation of the service stop request, used by the *func:stop* method

**Returns** True if the service succeeded to stop, False otherwise

**Return type** `bool`

`state()`

Check the current status of the service

**Returns** Which state the service is at right now

**Return type** `ServiceState`

## 4.1.17 lago.ssh module

**exception** `lago.ssh.LagoSSTimeoutException`

Bases: `lago.utils.LagoException`

`lago.ssh._gen_ssh_command_id()`

`lago.ssh.drain_ssh_channel(chan, stdin=None, stdout=<open file '<stdout>', mode 'w'>, stderr=<open file '<stderr>', mode 'w'>)`

`lago.ssh.get_ssh_client(ip_addr, ssh_key=None, host_name=None, ssh_tries=None, propagate_fail=True, username='root', password='123456')`

Get a connected SSH client

**Parameters**

- **ip\_addr** (*str*) – IP address of the endpoint
- **ssh\_key** (*str* or *list of str*) – Path to a file which contains the private key
- **hostname** (*str*) – The hostname of the endpoint
- **ssh\_tries** (*int*) – The number of attempts to connect to the endpoint
- **propagate\_fail** (*bool*) – If set to true, this event will be in the log and fail the outer stage. Otherwise, it will be discarded.
- **username** (*str*) – The username to authenticate with
- **password** (*str*) – Used for password authentication or for private key decryption

**Raises** *LagoSshTimeoutException* – If the client failed to connect after “ssh\_tries”

```
lago.ssh.interactive_ssh(ip_addr, command=None, host_name=None, ssh_key=None, user-
                        name='root', password='123456')
lago.ssh.interactive_ssh_channel(chan, command=None, stdin=<open file '<stdin>', mode
                                'r'>)
lago.ssh.ssh(ip_addr, command, host_name=None, data=None, show_output=True, propa-
             gate_fail=True, tries=None, ssh_key=None, username='root', password='123456')
lago.ssh.ssh_script(ip_addr, path, host_name=None, show_output=True, ssh_key=None, user-
                   name='root', password='123456')
lago.ssh.wait_for_ssh(ip_addr, host_name=None, connect_timeout=600, ssh_key=None, user-
                    name='root', password='123456')
```

**4.1.18 lago.subnet\_lease module**

```
exception lago.subnet_lease.LagoSubnetLeaseBadPermissionsException(store_path,
                                                                    prv_msg)
```

Bases: *lago.subnet\_lease.LagoSubnetLeaseException*

```
exception lago.subnet_lease.LagoSubnetLeaseException(msg, prv_msg=None)
```

Bases: *lago.utils.LagoException*

```
exception lago.subnet_lease.LagoSubnetLeaseLockException(store_path)
```

Bases: *lago.subnet\_lease.LagoSubnetLeaseException*

```
exception lago.subnet_lease.LagoSubnetLeaseMalformedAddrException(required_subnet)
```

Bases: *lago.subnet\_lease.LagoSubnetLeaseException*

```
exception lago.subnet_lease.LagoSubnetLeaseOutOfRangeException(required_subnet,
                                                                    store_range)
```

Bases: *lago.subnet\_lease.LagoSubnetLeaseException*

```
exception lago.subnet_lease.LagoSubnetLeaseStoreFullException(store_range)
```

Bases: *lago.subnet\_lease.LagoSubnetLeaseException*

```
exception lago.subnet_lease.LagoSubnetLeaseTakenException(required_subnet,
                                                            lease_taken_by)
```

Bases: *lago.subnet\_lease.LagoSubnetLeaseException*

```
class lago.subnet_lease.Lease(store_path, subnet)
```

Bases: *object*

Lease object is an abstraction of a lease file.

**`_store_path`**  
*str* – Path to the lease’s store.

**`_subnet`**  
*str* – The subnet that this lease represents

**`_path`**  
*str* – The path to the lease file

**`_has_env`** (*uuid\_path=None, uuid=None*)

**`_realise_lease_path`** ()

**`exist`**

**`has_env`**

**`metadata`**

**`path`**

**`subnet`**

**`to_ip_network`** ()

**`uuid`**

**`uuid_path`**

**`valid`**

```
class lago.subnet_lease.SubnetStore (path=None, min_third_octet=200,
                                     max_third_octet=255)
```

Bases: `object`

SubnetStore object represents a store of subnets used by lago for network bridges.

---

**Note:** Currently only /24 ranges are handled, and all of them under the 192.168.\_min\_third\_octet to 192.168.\_max\_third\_octet ranges.

---

The leases are stored under the store’s directory (which is specified with the *path* argument) as json files with the form:

```
[
  "/path/to/prefix/uuid/file",
  "uuid_hash",
]
```

Where the *uuid\_hash* is the 32 char uuid of the prefix (the contents of the uuid file at the time of doing the lease).

The helper class *Lease* is used to abstract the interaction with the lease files in the store (each file will be represented with a Lease object).

Cleanup of stale leases is done in a lazy manner during a request for a lease. The store will remove at most 1 stale lease in each request (see SubnetStore.\_lease\_valid for more info).

**`_path`**  
*str* – Path to the store, if not specified defaults to the value of *lease\_dir* in the config

**`_cidr`**  
*int* – Number of bits dedicated for the network address. Has a fixed value of 24.

**`_subnet_template`**  
*str* – A template for creating ip address. Has a fixed value of *192.168.{}.0*

---

**`_min_third_octet`**  
*int* – The minimum value of the subnets’ last octet.

**`_max_third_octet`**  
*int* – The maximum value of the subnets’ last octet.

**`_min_subnet`**  
*netaddr.IPNetwork* – The lowest subnet in the range of the store.

**`_max_subnet`**  
*netaddr.IPNetwork* – The highest subnet in the range of the store.

**`_acquire`** (*uuid\_path*)  
 Lease a free network for the given uuid path

**Parameters** **`uuid_path`** (*str*) – Path to the uuid file of a `lago.Prefix`

**Returns** Which represents the selected subnet

**Return type** `netaddr.IPNetwork`

**Raises** `LagoSubnetLeaseException` – If the store is full

**`_acquire_given_subnet`** (*uuid\_path*, *subnet*)  
 Try to create a lease for subnet

**Parameters**

- **`uuid_path`** (*str*) – Path to the uuid file of a `lago.Prefix`
- **`subnet`** (*str*) – dotted ipv4 subnet (for example ``192.168.200.0``)

**Returns** Which represents the selected subnet

**Return type** `netaddr.IPNetwork`

**Raises** `LagoSubnetLeaseException` – If the requested subnet is not in the range of this store or its already been taken

**`_create_lock`** ()

**`_lease_owned`** (*lease*, *current\_uuid\_path*)  
 Checks if the given lease is owned by the prefix whose uuid is in the given path

---

**Note:** The prefix must be also in the same path it was when it took the lease

---

**Parameters**

- **`path`** (*str*) – Path to the lease
- **`current_uuid_path`** (*str*) – Path to the uuid to check ownership of

**Returns**

**True if the given lease in owned by the prefix,** `False` otherwise

**Return type** `bool`

**`_lease_valid`** (*lease*)  
 Check if the given lease exist and still has a prefix that owns it. If the lease exist but its prefix isn’t, remove the lease from this store.

**Parameters** **`lease`** (`lago.subnet_lease.Lease`) – Object representation of the lease

**Returns**

If the lease and its prefix exists, return the path to the uuid of the prefix, else return None.

**Return type** `str` or None

**`_release`** (*lease*)

Free the given lease

**Parameters** **lease** (`lago.subnet_lease.Lease`) – The lease to free

**`_take_lease`** (*lease*, *uuid\_path*, *safe=True*)

Persist the given lease to the store and make the prefix in *uuid\_path* his owner

**Parameters**

- **lease** (`lago.subnet_lease.Lease`) – Object representation of the lease
- **uuid\_path** (*str*) – Path to the prefix uuid
- **safe** (*bool*) – If true (the default), validate the the lease isn't taken.

**Raises** `LagoSubnetLeaseException` – If *safe* == True and the lease is already taken.

**`_validate_lease_dir`** ()

Validate that the directory used by this store exist, otherwise create it.

**`acquire`** (*uuid\_path*, *subnet=None*)

Lease a free subnet for the given uuid path. If subnet is given, try to lease that subnet, otherwise try to lease a free subnet.

**Parameters**

- **uuid\_path** (*str*) – Path to the uuid file of a `lago.Prefix`
- **subnet** (*str*) – A subnet to lease.

**Returns** An object which represents the subnet.

**Return type** `netaddr.IPAddress`

**Raises**

- `LagoSubnetLeaseException` – 1. If this store is full 2. If the requested subnet is already taken.
- `LagoSubnetLeaseLockException` – If the lock to *self.path* can't be acquired.

**`create_lease_object_from_idx`** (*idx*)

Create a lease from *self.\_subnet\_template* and put *idx* as its third octet.

**Parameters** **idx** (*str*) – The value of the third octet

**Returns** Lease object which represents the requested subnet.

**Return type** `Lease`

**Raises**

- `LagoSubnetLeaseOutOfRangeException` – If the resultant subnet is malformed or out of the range of the store.

**`create_lease_object_from_subnet`** (*subnet*)

Create a lease from ip in a dotted decimal format, (for example *192.168.200.0/24*). the *\_cidr* will be added if not exist in *subnet*.



**Parameters** `subnet` (*str*) – The value of the third octet

**Returns** Lease object which represents the requested subnet.

**Return type** *Lease*

**Raises**

- *LagoSubnetLeaseOutOfRangeException* – If the resultant subnet is
- malformed or out of the range of the store.

**get\_allowed\_range()**

**Returns**

The range of the store (with lowest and highest subnets as the bounds).

**Return type** *str*

**is\_leasable\_subnet** (*subnet*)

Checks if a given subnet is inside the defined provision-able range

**Parameters** `subnet` (*str*) – Ip in dotted decimal format with `_cidr` notation (for example `192.168.200.0/24`)

**Returns**

True if subnet can be parsed into IPNetwork object and is inside the range, False otherwise

**Return type** *bool*

**Raises** `netaddr.AddrFormatError` – If subnet can not be parsed into an ip.

**list\_leases** (*uuid=None*)

List current subnet leases

**Parameters** `uuid` (*str*) – Filter the leases by uuid

**Returns** class:~Lease: current leases

**Return type** list of

**path**

**release** (*subnets*)

Free the lease of the given subnets

**Parameters** `subnets` (*list of str or netaddr.IPAddress*) – dotted ipv4 subnet in CIDR notation (for example ``192.168.200.0/24``) or IPAddress object.

**Raises**

- *LagoSubnetLeaseException* – If subnet is a str and can't be parsed
- *LagoSubnetLeaseLockException* – If the lock to self.path can't be acquired.

#### 4.1.19 lago.sysprep module

`lago.sysprep.__guestfs_version` (*default={'major': 1L, 'minor': 20L}*)

`lago.sysprep.__render_template` (*distro, loader, \*\*kwargs*)

`lago.sysprep.sysprep` (*disk, distro, loader=None, backend='direct', \*\*kwargs*)

Run virt-sysprep on the disk, commands are built from the distro specific template and arguments passed in kwargs. If no template is available it will default to `sysprep-base.j2`.

**Parameters**

- **disk** (*str*) – path to disk
- **distro** (*str*) – distro to render template for
- **loader** (*jinja2.BaseLoader*) – Jinja2 template loader, if not passed, will search Lago's package.
- **backend** (*str*) – libguestfs backend to use
- **\*\*kwargs** (*dict*) – environment variables for Jinja2 template

**Returns** None**Raises** `RuntimeError` – On virt-sysprep none 0 exit code.

## 4.1.20 lago.templates module

This module contains any disk template related classes and functions, including the repository store manager classes and template providers, some useful definitions:

- **Template repositories:** Repository where to fetch templates from, as an http server
- **Template store:** Local store to cache templates
- **Template:** Uninitialized disk image to use as base for other disk images
- **Template version:** Specific version of a template, to allow getting updates without having to change the template name everywhere

**class** `lago.templates.FileSystemTemplateProvider` (*root*)

Handles file type templates, that is, getting a disk template from the host's filesystem

**\_prefixed** (*\*path*)

Join all the given paths prefixed with this provider's base root path

**Parameters** **\*path** (*str*) – sections of the path to join, passed as positional arguments**Returns** Joined paths prepended with the provider root path**Return type** `str`**download\_image** (*handle, dest*)

Copies over the handl to the destination

**Parameters**

- **handle** (*str*) – path to copy over
- **dest** (*str*) – path to copy to

**Returns** None**get\_hash** (*handle*)Returns the associated hash for the given handle, the hash file must exist (`handle + '.hash'`).**Parameters** **handle** (*str*) – Path to the template to get the hash from**Returns** Hash for the given handle**Return type** `str`**get\_metadata** (*handle*)Returns the associated metadata info for the given handle, the metadata file must exist (`handle + '.metadata'`).

**Parameters** `handle` (*str*) – Path to the template to get the metadata from

**Returns** Metadata for the given handle

**Return type** `dict`

**class** `lago.templates.HttpTemplateProvider` (*baseurl*)

This provider allows the usage of http urls for templates

**download\_image** (*handle*, *dest*)

Downloads the image from the http server

**Parameters**

- **handle** (*str*) – url from the *self.baseurl* to the remote template
- **dest** (*str*) – Path to store the downloaded url to, must be a file path

**Returns** `None`

**static** `extract_image_xz` ()

**get\_hash** (*handle*)

Get the associated hash for the given handle, the hash file must exist (*handle* + '.hash').

**Parameters** `handle` (*str*) – Path to the template to get the hash from

**Returns** Hash for the given handle

**Return type** `str`

**get\_metadata** (*handle*)

Returns the associated metadata info for the given handle, the metadata file must exist (*handle* + '.metadata'). If the given handle has an `.xz` extension, it will get removed when calculating the handle metadata path

**Parameters** `handle` (*str*) – Path to the template to get the metadata from

**Returns** Metadata for the given handle

**Return type** `dict`

**open\_url** (*url*, *suffix*='', *dest*=None)

Opens the given url, trying the compressed version first. The compressed version url is generated adding the `.xz` extension to the *url* and adding the given suffix **after** that `.xz` extension. If *dest* passed, it will download the data to that path if able

**Parameters**

- **url** (*str*) – relative url from the *self.baseurl* to retrieve
- **suffix** (*str*) – optional suffix to append to the url after adding the compressed extension to the path
- **dest** (*str* or *None*) – Path to save the data to

**Returns**

**response object to read from (lazy read), closed** if no *dest* passed

**Return type** `urllib.addinfourl`

**Raises** `RuntimeError` – if the url gave http error when retrieving it

**class** `lago.templates.Template` (*name*, *versions*)

Disk image template class

**name**  
*str* – Name of this template

**\_versions** (*dict* (*str* *TemplateVersion*)): versions for this template

**get\_latest\_version** ()  
Retrieves the latest version for this template, the latest being the one with the newest timestamp

**Returns** *TemplateVersion*

**get\_version** (*ver\_name=None*)  
Get the given version for this template, or the latest

**Parameters** *ver\_name* (*str* or *None*) – Version to retrieve, *None* for the latest

**Returns**  
The version matching the given name or the latest one

**Return type** *TemplateVersion*

**class** *lago.templates.TemplateRepository* (*dom*)  
A template repository is a single source for templates, that uses different providers to actually retrieve them. That means for example that the ‘ovirt’ template repository, could support the ‘http’ and a theoretical ‘gluster’ template providers.

**\_dom**  
*dict* – Specification of the template

**\_providers**  
*dict* – Providers instances for any source in the spec

**\_get\_provider** (*spec*)  
Get the provider for the given template spec

**Parameters** *spec* (*dict*) – Template spec

**Returns** A provider instance for that spec

**Return type** *HttpTemplateProvider* or *FileSystemTemplateProvider*

**classmethod** **from\_url** (*path*)  
Instantiate a *TemplateRepository* instance from the data in a file or url

**Parameters** *path* (*str*) – Path or url to the json file to load

**Returns** A new instance

**Return type** *TemplateRepository*

**get\_by\_name** (*name*)  
Retrieve a template by it’s name

**Parameters** *name* (*str*) – Name of the template to retrieve

**Raises** *KeyError* – if no template is found

**name**  
Getter for the template repo name

**Returns** the name of this template repo

**Return type** *str*

**class** lago.templates.TemplateStore(*path*)

Local cache to store templates

The store uses various files to keep track of the templates cached, access and versions. An example template store looks like:

```
$ tree /var/lib/lago/store/
/var/lib/lago/store/
├── in_office_repo:centos6_engine:v2.tmp
├── in_office_repo:centos7_engine:v5.tmp
├── in_office_repo:fedora22_host:v2.tmp
├── phx_repo:centos6_engine:v2
├── phx_repo:centos6_engine:v2.hash
├── phx_repo:centos6_engine:v2.metadata
├── phx_repo:centos6_engine:v2.users
├── phx_repo:centos7_engine:v4.tmp
├── phx_repo:centos7_host:v4.tmp
└── phx_repo:storage-nfs:v1.tmp
```

There you can see the files:

- **\*.tmp** Temporary file created while downloading the template from the repository (depends on the provider)
- **\${repo\_name}:\${template\_name}:\${template\_version}** This file is the actual disk image template
- **\*.hash** Cached hash for the template disk image
- **\*.metadata** Metadata for this template image in json format, usually this includes the *distro* and *root-password*

**\_\_contains\_\_**(*temp\_ver*)

Checks if a given version is in this store

**Parameters** **temp\_ver** (*TemplateVersion*) – Version to look for

**Returns** `True` if the version is in this store

**Return type** `bool`

**\_prefixed**(*\*path*)

Join the given paths and prepend this stores path

**Parameters** **\*path** (*str*) – list of paths to join, as positional arguments

**Returns** all the paths joined and prepended with the store path

**Return type** `str`

**download**(*temp\_ver, store\_metadata=True*)

Retrieve the given template version

**Parameters**

- **temp\_ver** (*TemplateVersion*) – template version to retrieve
- **store\_metadata** (*bool*) – If set to `False`, will not refresh the local metadata with the retrieved one

**Returns** `None`

**get\_path**(*temp\_ver*)

Get the path of the given version in this store

**Parameters** **TemplateVersion** (*temp\_ver*) – version to look for

**Returns** The path to the template version inside the store

**Return type** `str`

**Raises** `RuntimeError` – if the template is not in the store

**get\_stored\_hash** (*temp\_ver*)

Retrieves the hash for the given template version from the store

**Parameters** **temp\_ver** (`TemplateVersion`) – template version to retrieve the hash for

**Returns** hash of the given template version

**Return type** `str`

**get\_stored\_metadata** (*temp\_ver*)

Retrieves the metadata for the given template version from the store

**Parameters** **temp\_ver** (`TemplateVersion`) – template version to retrieve the metadata for

**Returns** the metadata of the given template version

**Return type** `dict`

**class** `lago.templates.TemplateVersion` (*name, source, handle, timestamp*)

Each template can have multiple versions, each of those is actually a different disk template file representation, under the same base name.

**download** (*destination*)

Retrieves this template to the destination file

**Parameters** **destination** (`str`) – file path to write this template to

**Returns** `None`

**get\_hash** ()

Returns the associated hash for this template version

**Returns** Hash for this version

**Return type** `str`

**get\_metadata** ()

Returns the associated metadata info for this template version

**Returns** Metadata for this version

**Return type** `dict`

**timestamp** ()

Getter for the timestamp

`lago.templates._PROVIDERS = {'file': <class lago.templates.FileSystemTemplateProvider at ...>}`  
Registry for template providers

`lago.templates._locked` (*func*)

Decorator that ensures that the decorated function has the lock of the repo while running, meant to decorate only bound functions for classes that have `lock_path` method.

`lago.templates.find_repo_by_name` (*name, repo\_dir=None*)

Searches the given repo name inside the `repo_dir` (will use the config value 'template\_repos' if no repo dir passed), will rise an exception if not found

**Parameters**

- **name** (`str`) – Name of the repo to search

- **repo\_dir** (*str*) – Directory where to search the repo

**Returns** path to the repo

**Return type** *str*

**Raises** *RuntimeError* – if not found

#### 4.1.21 lago.utils module

**class** *lago.utils.CommandStatus*

Bases: *lago.utils.CommandStatus*

**class** *lago.utils.EggTimer* (*timeout*)

**elapsed** ()

**class** *lago.utils.ExceptionTimer* (*timeout*)

Bases: *object*

**start** ()

**stop** ()

**exception** *lago.utils.LagoException*

Bases: *exceptions.Exception*

**exception** *lago.utils.LagoInitException*

Bases: *lago.utils.LagoException*

**exception** *lago.utils.LagoUserException*

Bases: *lago.utils.LagoException*

**class** *lago.utils.LockFile* (*path*, *timeout=None*, *\*\*kwargs*)

Bases: *object*

Context manager that creates a lock around a directory, with optional timeout in the acquire operation

##### Parameters

- **path** (*str*) – path to the dir to lock
- **timeout** (*int*) – timeout in seconds to wait while acquiring the lock
- **\*\*kwargs** (*dict*) – Any other param to pass to *lockfile.LockFile*

**\_\_enter\_\_** ()

Start the lock with timeout if needed in the acquire operation

**Raises** *TimerException* – if the timeout is reached before acquiring the lock

**class** *lago.utils.RollbackContext* (*\*args*)

Bases: *object*

A context manager for recording and playing rollback. The first exception will be remembered and re-raised after rollback

Sample usage: > with RollbackContext() as rollback: > step1() > rollback.prependDefer(lambda: undo step1) > def undoStep2(arg): pass > step2() > rollback.prependDefer(undoStep2, arg)

More examples see tests/utilsTests.py @ vdsms code

```
__exit__(exc_type, exc_value, traceback)
    If this function doesn't return True (or raises a different exception), python re-raises the original exception
    once this function is finished.

clear()

defer(func, *args, **kwargs)

prependDefer(func, *args, **kwargs)

exception lago.utils.TimerException
    Bases: exceptions.Exception

    Exception to throw when a timeout is reached

class lago.utils.VectorThread(targets)

    join_all(raise_exceptions=True)

    start_all()

lago.utils._CommandStatus
    alias of lago.utils.CommandStatus

lago.utils._add_subparser_to_cp(cp, section, actions, incl_unset)

lago.utils._ret_via_queue(func, queue)

lago.utils._run_command(command, input_data=None, stdin=None, out_pipe=-1, err_pipe=-1,
                        env=None, uuid=None, **kwargs)

    Runs a command
```

#### Parameters

- **command** (*list of str*) – args of the command to execute, including the command itself as `command[0]` as `['ls', '-l']`
- **input\_data** (*str*) – If passed, will feed that data to the subprocess through `stdin`
- **out\_pipe** (*int or file*) – File descriptor as passed to `:ref:subprocess.Popen` to use as `stdout`
- **stdin** (*int or file*) – File descriptor as passed to `:ref:subprocess.Popen` to use as `stdin`
- **err\_pipe** (*int or file*) – File descriptor as passed to `:ref:subprocess.Popen` to use as `stderr`
- **of str** (*env (dict)* – *str*): If set, will use the given dict as `env` for the subprocess
- **uuid** (*uuid*) – If set the command will be logged with the given `uuid` converted to string, otherwise, a `uuid v4` will be generated.
- **\*\*kwargs** – Any other keyword args passed will be passed to the `:ref:subprocess.Popen` call

**Returns** result of the interactive execution

**Return type** `lago.utils.CommandStatus`

```
lago.utils.add_timestamp_suffix(base_string)

lago.utils.parse_to_ini(parser, root_section='lago', incl_unset=False)

lago.utils.compress(input_file, block_size, fail_on_error=True)

lago.utils.cp(input_file, output_file, fail_on_error=True)
```



`lago.utils.deepcopy` (*original\_obj*)

Creates a deep copy of an object with no crossed referenced lists or dicts, useful when loading from yaml as anchors generate those cross-referenced dicts and lists

**Parameters** `original_obj` (*object*) – Object to deep copy

**Returns** deep copy of the object

**Return type** `object`

`lago.utils.filter_spec` (*spec, paths, wildcard='\*', separator='/'*)

Remove keys from a spec file. For example, with the following path: domains//disks//metadata all the metadata dicts from all domains disks will be removed.

**Parameters**

- `spec` (*dict*) – spec to remove keys from
- `paths` (*list*) – list of paths to the keys that should be removed
- `wildcard` (*str*) – wildcard character
- `separator` (*str*) – path separator

**Returns** None

**Raises** `utils.LagoUserException` – If a malformed path was detected

`lago.utils.func_vector` (*target, args\_sequence*)

`lago.utils.get_hash` (*file\_path, checksum='sha1'*)

Generate a hash for the given file

**Parameters**

- `file_path` (*str*) – Path to the file to generate the hash for
- `checksum` (*str*) – hash to apply, one of the supported by hashlib, for example sha1 or sha512

**Returns** hash for that file

**Return type** `str`

`lago.utils.get_qemu_info` (*path, backing\_chain=False, fail\_on\_error=True*)

Get info on a given qemu disk

**Parameters**

- `path` (*str*) – Path to the required disk
- `backing_chain` (*bool*) – if true, include also info about
- `image predecessors.` (*the*) –

**Returns** if `backing_chain == True` then a list of dicts else a dict

**Return type** `object`

`lago.utils.in_prefix` (*prefix\_class, workdir\_class*)

`lago.utils.invoke_different_funcs_in_parallel` (*\*funcs*)

`lago.utils.invoke_in_parallel` (*func, \*args\_sequences*)

`lago.utils.ipv4_to_mac` (*ip*)

`lago.utils.json_dump` (*obj, f*)

`lago.utils.load_virt_stream(virt_fd)`

Loads the given conf stream into a dict, trying different formats if needed

**Parameters** `virt_fd` (*str*) – file like object with the virt config to load

**Returns** Loaded virt config

**Return type** `dict`

`lago.utils.qemu_rebase(target, backing_file, safe=True, fail_on_error=True)`

changes the backing file of 'source' to 'backing\_file' If backing\_file is specified as "" (the empty string), then the image is rebased onto no backing file (i.e. it will exist independently of any backing file). (Taken from qemu-img man page)

**Parameters**

- **target** (*str*) – Path to the source disk
- **backing\_file** (*str*) – path to the base disk
- **safe** (*bool*) – if false, allow unsafe rebase (check qemu-img docs for more info)

`lago.utils.read_nonblocking(file_descriptor)`

`lago.utils.rotate_dir(base_dir)`

`lago.utils.run_command(command, input_data=None, out_pipe=-1, err_pipe=-1, env=None, **kwargs)`

Runs a command non-interactively

**Parameters**

- **command** (*list of str*) – args of the command to execute, including the command itself as command[0] as ['ls', '-l']
- **input\_data** (*str*) – If passed, will feed that data to the subprocess through stdin
- **out\_pipe** (*int or file*) – File descriptor as passed to :ref:subprocess.Popen to use as stdout
- **err\_pipe** (*int or file*) – File descriptor as passed to :ref:subprocess.Popen to use as stderr
- **of str** (*env (dict)* – str): If set, will use the given dict as env for the subprocess
- **\*\*kwargs** – Any other keyword args passed will be passed to the :ref:subprocess.Popen call

**Returns** result of the interactive execution

**Return type** `lago.utils.CommandStatus`

`lago.utils.run_command_with_validation(cmd, fail_on_error=True, msg='An error has occurred')`

`lago.utils.run_interactive_command(command, env=None, **kwargs)`

Runs a command interactively, reusing the current stdin, stdout and stderr

**Parameters**

- **command** (*list of str*) – args of the command to execute, including the command itself as command[0] as ['ls', '-l']
- **of str** (*env (dict)* – str): If set, will use the given dict as env for the subprocess
- **\*\*kwargs** – Any other keyword args passed will be passed to the :ref:subprocess.Popen call

**Returns** result of the interactive execution

**Return type** `lago.utils.CommandStatus`

`lago.utils.service_is_enabled(name)`

`lago.utils.sparse(input_file, input_format, fail_on_error=True)`

`lago.utils.ver_cmp(ver1, ver2)`

Compare lago versions

**Parameters**

- **ver1** (`str`) – version string
- **ver2** (`str`) – version string

**Returns** Return negative if ver1<ver2, zero if ver1==ver2, positive if ver1>ver2.

`lago.utils.with_logging(func)`

#### 4.1.22 lago.validation module

`lago.validation.check_import(module_name)`

Search if a module exists, and it is possible to try importing it

**Parameters** **module\_name** (`str`) – module to import

**Returns** True if the package is found

**Return type** `bool`

#### 4.1.23 lago.virt module

**class** `lago.virt.VirtEnv(prefix, vm_specs, net_specs)`

Bases: `object`

Env properties: \* prefix \* vms \* net

**\_create\_net** (`net_spec, compat`)

**\_create\_vm** (`vm_spec`)

**\_get\_stop\_shutdown\_common\_args** (`vm_names`)

Get the common arguments for stop and shutdown commands

**Parameters** **vm\_names** (`list of str`) – The names of the requested vms

**Returns**

**list of plugins.vm.VMProviderPlugin:** vms objects that should be stopped

list of virt.Network: net objects that should be stopped str: log message

**Raises** `utils.LagoUserException` – If a vm name doesn't exist

**\_get\_unused\_nets** (`vms_to_stop`)

Return a list of nets that used only by the vms in vms\_to\_stop

**Parameters** **vms\_to\_stop** (`list of str`) – The names of the requested vms

**Returns**

**list of virt.Network: net objects that used only by** vms in vms\_to\_stop

**Raises** `utils.LagoUserException` – If a vm name doesn't exist

**bootstrap()**

**create\_snapshots** (*\*\*kwargs*)

**export\_vms** (*vms\_names, standalone, dst\_dir, compress, init\_file\_name, out\_format, collect\_only=False, with\_threads=True*)

**classmethod from\_prefix** (*prefix*)

**generate\_init** (*dst, out\_format, vms\_to\_include, filters=None*)

Generate an init file which represents this env and can be used with the images created by self.export\_vms

**Parameters**

- **dst** (*str*) – path and name of the new init file
- **out\_format** (`plugins.output.OutFormatPlugin`) – formatter for the output (the default is yaml)
- **filters** (*list*) – list of paths to keys that should be removed from the init file
- **(list of (vms\_to\_include) – class:lago.plugins.vm.VMPlugin):** list of vms to include in the init file

**Returns** None

**get\_compat()**

Get compatibility level for this environment - which is the Lago version used to create this environment

**get\_env\_spec** (*filters=None*)

Get the spec of the current env. The spec will hold the info about all the domains and networks associated with this env.

**Parameters** **filters** (*list*) – list of paths to keys that should be removed from the init file

**Returns** the spec of the current env

**Return type** `dict`

**get\_net** (*name=None*)

**get\_nets()**

**get\_snapshots** (*domains=None*)

Get the list of snapshots for each domain

**Parameters**

- **domanins** (*list of str*) – list of the domains to get the snapshots
- **all will be returned if none or empty list passed** (*for,*) –

**Returns** with the domain names and the list of snapshots for each

**Return type** `dict of str -> list(str)`

**get\_vm** (*name*)

**get\_vms** (*vm\_names=None*)

Returns the vm objects associated with vm\_names if vm\_names is None, return all the vms in the prefix

**Parameters** **vm\_names** (*list of str*) – The names of the requested vms

**Returns** dict: Which contains the requested vm objects indexed by name

**Raises** `utils.LagoUserException` – If a vm name doesn't exist

**prefixed\_name** (*unprefixed\_name*, *max\_length=0*)

Returns a uuid prefixed identifier

**Parameters**

- **unprefixed\_name** (*str*) – Name to add a prefix to
- **max\_length** (*int*) – maximum length of the resultant prefixed name, will adapt the given name and the length of the uuid to fit it

**Returns** prefixed identifier for the given unprefixed name

**Return type** `str`

**revert\_snapshots** (*\*\*kwargs*)

**save** (*\*\*kwargs*)

**shutdown** (*vm\_names*, *reboot=False*)

**start** (*vm\_names=None*)

**stop** (*vm\_names=None*)

**virt\_path** (*\*args*)

`lago.virt._gen_ssh_command_id()`

`lago.virt._guestfs_copy_path(g, guest_path, host_path)`

`lago.virt._path_to_xml(basename)`

#### 4.1.24 lago.vm module

**class** `lago.vm.DefaultVM` (*env*, *spec*)

Bases: `lago.plugins.vm.VMPlugin`

`_abc_cache = <_weakrefset.WeakSet object>`

`_abc_negative_cache = <_weakrefset.WeakSet object>`

`_abc_negative_cache_version = 35`

`_abc_registry = <_weakrefset.WeakSet object>`

**class** `lago.vm.SSHVMPProvider` (*vm*)

Bases: `lago.plugins.vm.VMProviderPlugin`

**bootstrap** (*\*args*, *\*\*kwargs*)

Does any actions needed to get the domain ready to be used, ran on prefix init.

**Returns** None

**create\_snapshot** (*name*, *\*args*, *\*\*kwargs*)

Take any actions needed to create a snapshot

**Parameters** **name** (*str*) – Name for the snapshot, will be used as key to retrieve it later

**Returns** None

**defined** (*\*args*, *\*\*kwargs*)

**revert\_snapshot** (*name*, \**args*, \*\**kwargs*)

Take any actions needed to revert/restore a snapshot

**Parameters** **name** (*str*) – Name for the snapshot, same that was set on creation

**Returns** None

**running** (\**args*, \*\**kwargs*)

Returns: (bool): True if the VM is running

**start** (\**args*, \*\**kwargs*)

Start a domain

**Returns** None

**state** (\**args*, \*\**kwargs*)

Return the current state of the domain

**Returns** Small description of the current domain state

**Return type** *str*

**stop** (\**args*, \*\**kwargs*)

Stop a domain

**Returns** None

## 4.1.25 lago.workdir module

A workdir is the base directory where lago will store all the files it needs and that are unique (not shared between workdirs).

It's basic structure is a directory with one soft link and multiple directories, one per prefix. Where the link points to the default prefix to use.

**exception** `lago.workdir.MalformedWorkdir`

Bases: `lago.workdir.WorkdirError`

**exception** `lago.workdir.PrefixAlreadyExists`

Bases: `lago.workdir.WorkdirError`

**exception** `lago.workdir.PrefixNotFound`

Bases: `lago.workdir.WorkdirError`

**class** `lago.workdir.Workdir` (*path*, *prefix\_class*=<class 'lago.prefix.Prefix'>)

Bases: `object`

This class represents a base workdir, where you can store multiple prefixes

**Properties:** `path(str)`: Path to the workdir prefixes(dict of str->self.prefix\_class): dict with the prefixes in the workdir, by name `current(str)`: Name of the current prefix `prefix_class(type)`: Class to use when creating prefixes

**\_set\_current** (*new\_current*)

Change the current default prefix, for internal usage

**Parameters** **new\_current** (*str*) – Name of the new current prefix, it must already exist

**Returns** None

**Raises** `PrefixNotFound` – if the given prefix name does not exist in the workdir

**\_update\_current** ()

Makes sure that a current is set

**add\_prefix** (\*args, \*\*kwargs)

Adds a new prefix to the workdir.

**Parameters**

- **name** (*str*) – Name of the new prefix to add
- **\*args** – args to pass along to the prefix constructor
- **\*\*kwargs** – kwargs to pass along to the prefix constructor

**Returns** The newly created prefix

**Raises** *PrefixAlreadyExists* – if the prefix name already exists in the workdir

**cleanup** ()

Attempt to set a new current symlink if it is broken. If no other prefixes exist and the workdir is empty, try to delete the entire workdir.

**Raises** *MalformedWorkdir* – if no prefixes were found, but the workdir is not empty.

**destroy** (\*args, \*\*kwargs)

Destroy all the given prefixes and remove any left files if no more prefixes are left

**Parameters**

- **prefix\_names** (*list of str*) – list of prefix names to destroy, if None
- **passed** (*default*) –

**get\_prefix** (\*args, \*\*kwargs)

Retrieve a prefix, resolving the current one if needed

**Parameters** **name** (*str*) – name of the prefix to retrieve, or current to get the current one

**Returns** instance of the prefix with the given name

**Return type** self.prefix\_class

**initialize** (prefix\_name='default', \*args, \*\*kwargs)

Initializes a workdir by adding a new prefix to the workdir.

**Parameters**

- **prefix\_name** (*str*) – Name of the new prefix to add
- **\*args** – args to pass along to the prefix constructor
- **\*\*kwargs** – kwargs to pass along to the prefix constructor

**Returns** The newly created prefix

**Raises** *PrefixAlreadyExists* – if the prefix name already exists in the workdir

**static is\_possible\_workdir** (path)

A quick method to suggest if the path is a possible workdir. This does not guarantee that the workdir is not malformed, only that by simple heuristics it might be one. For a full check use *is\_workdir* ().

**Parameters** **path** (*str*) – Path

**Returns** True if path might be a work dir.

**Return type** bool

**classmethod is\_workdir** (path)

Check if the given path is a workdir

**Parameters** **path** (*str*) – Path to check

**Returns** True if the given path is a workdir

**Return type** `bool`

**join** (\*args)

Gets a joined path prefixed with the workdir path

**Parameters** \*args (*str*) – path sections to join

**Returns** Joined path prefixed with the workdir path

**Return type** `str`

**load** ()

Loads the prefixes that are available in the workdir

**Returns** None

**Raises** *MalformedWorkdir* – if the workdir is malformed

**classmethod resolve\_workdir\_path** (start\_path='.')

Look for an existing workdir in the given path, in a path/.lago dir, or in a .lago dir under any of its parent directories

**Parameters** start\_path (*str*) – path to start the search from, if None passed, it will use the current dir

**Returns** path to the found prefix

**Return type** `str`

**Raises** *LagoUserException* – if no prefix was found

**set\_current** (\*args, \*\*kwargs)

Change the current default prefix

**Parameters** new\_current (*str*) – Name of the new current prefix, it must already exist

**Returns** None

**Raises** *PrefixNotFound* – if the given prefix name does not exist in the workdir

**exception** lago.workdir.**WorkdirError**

Bases: `exceptions.RuntimeError`

Base exception for workdir errors, catch this one to catch any workdir error

**lago.workdir.workdir\_loaded** (*func*)

Decorator to make sure that the workdir is loaded when calling the decorated function



## 5.1 Release process

### 5.1.1 Versioning

For Iago we use a similar approach to semantic versioning, that is:

`X.Y.Z`

For example:

`0.1.0  
1.2.123  
2.0.0  
2.0.1`

Where:

- Z changes for each patch (number of patches since X.Y tag)
- Y changes from time to time, with milestones (arbitrary bump), only for backwards compatible changes
- X changes if it's a non-backwards compatible change or arbitrarily (we don't like Y getting too high, or big milestone reached, ...)

The source tree has tags with the X.Y versions, that's where the packaging process gets them from.

On each X or Y change a new tag is created.

For now we have only one branch (master) and we will try to keep it that way as long as possible, if at some point we have to support old versions, then we will create a branch for each X version in the form:

`vX`

For example:

```
v0
v1
```

There's a helper script to resolve the current version, based on the last tag and the compatibility breaking commits since then, to get the version for the current repo run:

```
$ scripts/version_manager.py . version
```

## 5.1.2 RPM Versioning

The rpm versions differ from the generic version in that they have the `-1` suffix, where the `-1` is the release for that rpm (usually will never change, only when repackaging without any code change, something that is not so easy for us but if there's any external packagers is helpful for them)

## 5.1.3 Repository layout

Tree schema of the repository:

```
lago
├── stable <-- subdirs for each major version to avoid accidental
│               non-backwards compatible upgrade
│   ├── 0.0 <-- Contains any 0.* release for lago
│   │   ├── ChangeLog_0.0.txt
│   │   ├── rpm
│   │   │   ├── el6
│   │   │   ├── el7
│   │   │   ├── fc22
│   │   │   └── fc23
│   │   └── sources
│   ├── 1.0
│   │   ├── ChangeLog_1.0.txt
│   │   ├── rpm
│   │   │   ├── el6
│   │   │   ├── el7
│   │   │   ├── fc22
│   │   │   └── fc23
│   │   └── sources
│   └── 2.0
│       ├── ChangeLog_2.0.txt
│       ├── rpm
│       │   ├── el6
│       │   ├── el7
│       │   ├── fc22
│       │   └── fc23
│       └── sources
└── unstable <-- Multiple subdirs are needed only if branching
    ├── 0.0 <-- Contains 0.* builds that might or might not have
    │       been released
    │   ├── latest <--- keeps the latest build from merged, static
    │   │           url
    │   ├── snapshot-lago_0.0_pipeline_1
    │   └── snapshot-lago_0.0_pipeline_2
    │       ^ contains the rpms created on the pipeline build
```

(continues on next page)

(continued from previous page)

```

    number 2 for the 0.0 version, this is needed to
    ease the automated testing of the rpms
  ... <-- this is cleaned up from time to time to avoid
         using too much space

1.0
├── latest
├── snapshot-lago_1.0_pipeline_1
├── snapshot-lago_pipeline_2
├── ...
2.0
├── latest
├── snapshot-lago_2.0_pipeline_1
├── snapshot-lago_2.0_pipeline_2
├── ...

```

### 5.1.4 Promotion of artifacts to stable, aka. releasing

The goal is to have an automated set of tests, that check in depth lago, and run them in a timely fashion, and if passed, deploy to stable. As right now that's not yet possible, so for now the tests and deploy is done manually.

The promotion of the artifacts is done in these cases:

- New major version bump ( $X+1.0$ , for example  $1.0 \rightarrow 2.0$ )
- New minor version bump ( $X.Y+1$ , for example  $1.1 \rightarrow 1.2$ )
- If it passed certain time since the last  $X$  or  $Y$  version bumps ( $X.Y.Z+n$ , for example  $1.0.1 \rightarrow 1.0.2$ )
- If there are blocking/important bugfixes ( $X.Y.Z+n$ )
- If there are important new features ( $X.Y+1$  or  $X.Y.Z+n$ )

### 5.1.5 How to mark a major version

Whenever there's a commit that breaks the backwards compatibility, you should add to it the pseudo-header:

```
Sem-Ver: api-breaking
```

And that will force a major version bump for any package built from it, that is done so in the moment when you submit the commit in Gerrit, the packages that are build from it have the correct version.

After that, make sure that you tag that commit too, so it will be easy to look for it in the future.

### 5.1.6 The release procedure on the maintainer side

1. Select the snapshot repo you want to release
2. **Test the rpms, for now we only have the tests from projects that use it:**
  - Run all the **ovirt tests** on it, make sure it does not break anything, if there are issues -> [open bug](#)
  - Run **vdsm functional tests**, make sure it does not break anything, if there are issues -> [open bug](#)
3. **On non-major version bump  $X.Y+1$  or  $X.Y.Z+n$**

- **Create a changelog** since the base of the tag and keep it aside

4. **On Major version bump X+1 . 0**

- **Create a changelog since the previous . 0 tag (X . 0) and keep** it aside

5. Deploy the rpms from snapshot to dest repo and copy the ChangeLog from the tarball to ChangeLog\_X . 0 . txt in the base of the stable/X . 0/ dir
6. Send email to [lago-devel](#) with the announcement and the changelog since the previous tag that you kept aside, feel free to change the body to your liking:

```
Subject: [day-month-year] New lago release - X.Y.Z

Hi everyone! There's a new lago release with version X.Y.Z ready for you to
upgrade!

Here are the changes:
    <CHANGELOG HERE>

Enjoy!
```

## CHAPTER 6

---

### Changelog

---

Here you can find the [full changelog](#) for this version



## CHAPTER 7

---

### Indices and tables

---

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





### I

- [lago](#), 23
- [lago.brctl](#), 43
- [lago.build](#), 43
- [lago.cmd](#), 46
- [lago.config](#), 46
- [lago.constants](#), 47
- [lago.export](#), 48
- [lago.guestfs\\_tools](#), 50
- [lago.lago\\_ansible](#), 52
- [lago.log\\_utils](#), 53
- [lago.paths](#), 58
- [lago.plugins](#), 23
- [lago.plugins.cli](#), 24
- [lago.plugins.output](#), 28
- [lago.plugins.service](#), 30
- [lago.plugins.vm](#), 31
- [lago.prefix](#), 59
- [lago.providers](#), 35
- [lago.providers.libvirt](#), 35
- [lago.providers.libvirt.cpu](#), 35
- [lago.providers.libvirt.network](#), 38
- [lago.providers.libvirt.utils](#), 39
- [lago.providers.libvirt.vm](#), 40
- [lago.sdk](#), 67
- [lago.sdk\\_utils](#), 68
- [lago.service](#), 69
- [lago.ssh](#), 70
- [lago.subnet\\_lease](#), 71
- [lago.sysprep](#), 75
- [lago.templates](#), 76
- [lago.utils](#), 81
- [lago.validation](#), 85
- [lago.virt](#), 85
- [lago.vm](#), 87
- [lago.workdir](#), 88



## Symbols

- `__CommandStatus` (in module `lago.utils`), 82
- `__PROVIDERS` (in module `lago.templates`), 80
- `__call__()` (`lago.plugins.cli.CLIPluginFuncWrapper` method), 25
- `__contains__()` (`lago.templates.TemplateStore` method), 79
- `__enter__()` (`lago.utils.LockFile` method), 81
- `__exit__()` (`lago.utils.RollbackContext` method), 81
- `__getitem__()` (`lago.config.ConfigLoad` method), 46
- `__getnewargs__()` (`lago.build.Command` method), 45
- `__getstate__()` (`lago.build.Command` method), 45
- `__repr__()` (`lago.build.Command` method), 45
- `_abc_cache` (`lago.export.DiskExportManager` attribute), 48
- `_abc_cache` (`lago.export.FileExportManager` attribute), 49
- `_abc_cache` (`lago.export.TemplateExportManager` attribute), 49
- `_abc_cache` (`lago.plugins.cli.CLIPlugin` attribute), 25
- `_abc_cache` (`lago.plugins.cli.CLIPluginFuncWrapper` attribute), 25
- `_abc_cache` (`lago.plugins.output.DefaultOutFormatPlugin` attribute), 28
- `_abc_cache` (`lago.plugins.output.FlatOutFormatPlugin` attribute), 28
- `_abc_cache` (`lago.plugins.output.JSONOutFormatPlugin` attribute), 29
- `_abc_cache` (`lago.plugins.output.OutFormatPlugin` attribute), 29
- `_abc_cache` (`lago.plugins.output.YAMLOutFormatPlugin` attribute), 29
- `_abc_cache` (`lago.plugins.service.ServicePlugin` attribute), 30
- `_abc_cache` (`lago.plugins.vm.VMPlugin` attribute), 31
- `_abc_cache` (`lago.service.SysVInitService` attribute), 69
- `_abc_cache` (`lago.service.SystemdContainerService` attribute), 69
- `_abc_cache` (`lago.service.SystemdService` attribute), 70
- `_abc_cache` (`lago.vm.DefaultVM` attribute), 87
- `_abc_negative_cache` (`lago.export.DiskExportManager` attribute), 48
- `_abc_negative_cache` (`lago.export.FileExportManager` attribute), 49
- `_abc_negative_cache` (`lago.export.TemplateExportManager` attribute), 49
- `_abc_negative_cache` (`lago.plugins.cli.CLIPlugin` attribute), 25
- `_abc_negative_cache` (`lago.plugins.cli.CLIPluginFuncWrapper` attribute), 25
- `_abc_negative_cache` (`lago.plugins.output.DefaultOutFormatPlugin` attribute), 28
- `_abc_negative_cache` (`lago.plugins.output.FlatOutFormatPlugin` attribute), 28
- `_abc_negative_cache` (`lago.plugins.output.JSONOutFormatPlugin` attribute), 29
- `_abc_negative_cache` (`lago.plugins.output.OutFormatPlugin` attribute), 29
- `_abc_negative_cache` (`lago.plugins.output.YAMLOutFormatPlugin` attribute), 29
- `_abc_negative_cache` (`lago.plugins.service.ServicePlugin` attribute), 30
- `_abc_negative_cache` (`lago.plugins.vm.VMPlugin` attribute), 31
- `_abc_negative_cache` (`lago.service.SysVInitService` attribute), 69
- `_abc_negative_cache` (`lago.service.SystemdContainerService` attribute), 69
- `_abc_negative_cache` (`lago.service.SystemdService` attribute), 70
- `_abc_negative_cache` (`lago.vm.DefaultVM` attribute), 87
- `_abc_negative_cache_version` (`lago.export.DiskExportManager` attribute), 48
- `_abc_negative_cache_version` (`lago.export.FileExportManager` attribute), 49
- `_abc_negative_cache_version` (`lago.export.TemplateExportManager` attribute), at-

tribute), 49

`_abc_negative_cache_version` (lago.plugins.cli.CLIPlugin attribute), 25

`_abc_negative_cache_version` (lago.plugins.cli.CLIPluginFuncWrapper attribute), 25

`_abc_negative_cache_version` (lago.plugins.output.DefaultOutFormatPlugin attribute), 28

`_abc_negative_cache_version` (lago.plugins.output.FlatOutFormatPlugin attribute), 28

`_abc_negative_cache_version` (lago.plugins.output.JSONOutFormatPlugin attribute), 29

`_abc_negative_cache_version` (lago.plugins.output.OutFormatPlugin attribute), 29

`_abc_negative_cache_version` (lago.plugins.output.YAMLOutFormatPlugin attribute), 29

`_abc_negative_cache_version` (lago.plugins.service.ServicePlugin attribute), 30

`_abc_negative_cache_version` (lago.plugins.vm.VMPlugin attribute), 31

`_abc_negative_cache_version` (lago.service.SysVInitService attribute), 69

`_abc_negative_cache_version` (lago.service.SystemdContainerService attribute), 69

`_abc_negative_cache_version` (lago.service.SystemdService attribute), 70

`_abc_negative_cache_version` (lago.vm.DefaultVM attribute), 87

`_abc_registry` (lago.export.DiskExportManager attribute), 48

`_abc_registry` (lago.export.FileExportManager attribute), 49

`_abc_registry` (lago.export.TemplateExportManager attribute), 49

`_abc_registry` (lago.plugins.cli.CLIPlugin attribute), 25

`_abc_registry` (lago.plugins.cli.CLIPluginFuncWrapper attribute), 25

`_abc_registry` (lago.plugins.output.DefaultOutFormatPlugin attribute), 28

`_abc_registry` (lago.plugins.output.FlatOutFormatPlugin attribute), 28

`_abc_registry` (lago.plugins.output.JSONOutFormatPlugin attribute), 29

`_abc_registry` (lago.plugins.output.OutFormatPlugin attribute), 29

`_abc_registry` (lago.plugins.output.YAMLOutFormatPlugin attribute), 29

`_abc_registry` (lago.service.ServicePlugin attribute), 30

`_abc_registry` (lago.service.SysVInitService attribute), 69

`_abc_registry` (lago.service.SystemdContainerService attribute), 69

`_abc_registry` (lago.service.SystemdService attribute), 70

`_abc_registry` (lago.vm.DefaultVM attribute), 87

`_acquire()` (lago.subnet\_lease.SubnetStore method), 73

`_acquire_given_subnet()` (lago.subnet\_lease.SubnetStore method), 73

`_add_dns_records()` (lago.prefix.Prefix method), 59

`_add_mgmt_to_domains()` (lago.prefix.Prefix method), 59

`_add_nic_to_mapping()` (lago.prefix.Prefix method), 59

`_add_subparser_to_cp()` (in module lago.utils), 82

`_allocate_ips_to_nics()` (lago.prefix.Prefix method), 59

`_allocate_subnets()` (lago.prefix.Prefix method), 60

`_artifact_paths()` (lago.plugins.vm.VMPlugin method), 31

`_asdict()` (lago.build.Command method), 45

`_brctl()` (in module lago.brctl), 43

`_cidr` (lago.subnet\_lease.SubnetStore attribute), 72

`_collect()` (lago.export.VMExportManager method), 50

`_config_net_topology()` (lago.prefix.Prefix method), 60

`_copy_delpoy_scripts()` (lago.prefix.Prefix method), 60

`_copy_deploy_scripts_for_hosts()` (lago.prefix.Prefix method), 60

`_copy_path()` (in module lago.guestfs\_tools), 50

`_createXML()` (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 40

`_create_dead_snapshot()` (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 40

`_create_disk()` (lago.prefix.Prefix method), 60

`_create_disks()` (lago.prefix.Prefix method), 60

`_create_ip()` (in module lago.prefix), 66

`_create_link_to_parent()` (lago.prefix.Prefix method), 60

`_create_live_snapshot()` (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 40

`_create_lock()` (lago.subnet\_lease.SubnetStore method), 73

`_create_net()` (lago.virt.VirtEnv method), 85

`_create_ssh_keys()` (lago.prefix.Prefix method), 60

`_create_virt_env()` (lago.prefix.Prefix method), 61

`_create_vm()` (lago.virt.VirtEnv method), 85

`_deploy_host()` (lago.prefix.Prefix method), 61

`_detect_service_provider()` (lago.plugins.vm.VMPlugin method), 31

`_dom` (lago.templates.TemplateRepository attribute), 78

`_extract_paths_scp()` (lago.plugins.vm.VMProviderPlugin method), 33

`_fields` (lago.build.Command attribute), 45

[\\_gen\\_ssh\\_command\\_id\(\) \(in module lago.ssh\), 70](#)  
[\\_gen\\_ssh\\_command\\_id\(\) \(in module lago.virt\), 87](#)  
[\\_generate\\_disk\\_name\(\) \(lago.prefix.Prefix static method\), 61](#)  
[\\_generate\\_disk\\_path\(\) \(lago.prefix.Prefix method\), 61](#)  
[\\_generate\\_dns\\_disable\(\) \(lago.providers.libvirt.network.NATNetwork method\), 38](#)  
[\\_generate\\_dns\\_forward\(\) \(lago.providers.libvirt.network.NATNetwork method\), 38](#)  
[\\_generate\\_entry\(\) \(lago.lago\\_ansible.LagoAnsible method\), 52](#)  
[\\_generate\\_main\\_dns\(\) \(lago.providers.libvirt.network.NATNetwork method\), 38](#)  
[\\_get\\_configs\\_path\(\) \(in module lago.config\), 47](#)  
[\\_get\\_domain\(\) \(lago.providers.libvirt.vm.LocalLibvirtVMProvider method\), 40](#)  
[\\_get\\_export\\_mgr\(\) \(lago.export.VMExportManager method\), 50](#)  
[\\_get\\_net\(\) \(lago.prefix.Prefix method\), 61](#)  
[\\_get\\_provider\(\) \(lago.templates.TemplateRepository method\), 78](#)  
[\\_get\\_qemu\\_kvm\\_path\(\) \(lago.providers.libvirt.vm.LocalLibvirtVMProvider method\), 40](#)  
[\\_get\\_scripts\(\) \(lago.prefix.Prefix method\), 61](#)  
[\\_get\\_service\\_provider\(\) \(lago.plugins.vm.VMPlugin method\), 31](#)  
[\\_get\\_stop\\_shutdown\\_common\\_args\(\) \(lago.virt.VirtEnv method\), 85](#)  
[\\_get\\_unused\\_nets\(\) \(lago.virt.VirtEnv method\), 85](#)  
[\\_get\\_vm\\_provider\(\) \(lago.plugins.vm.VMPlugin method\), 31](#)  
[\\_guestfs\\_copy\\_path\(\) \(in module lago.virt\), 87](#)  
[\\_guestfs\\_version\(\) \(in module lago.sysprep\), 75](#)  
[\\_handle\\_empty\\_disk\(\) \(lago.prefix.Prefix method\), 61](#)  
[\\_handle\\_file\\_disk\(\) \(lago.prefix.Prefix method\), 61](#)  
[\\_handle\\_lago\\_template\(\) \(lago.prefix.Prefix method\), 61](#)  
[\\_handle\\_ova\\_image\(\) \(lago.prefix.Prefix method\), 61](#)  
[\\_handle\\_qcow\\_template\(\) \(lago.prefix.Prefix method\), 61](#)  
[\\_handle\\_template\(\) \(lago.prefix.Prefix method\), 61](#)  
[\\_has\\_env\(\) \(lago.subnet\\_lease.Lease method\), 72](#)  
[\\_init\\_net\\_specs\(\) \(lago.prefix.Prefix static method\), 61](#)  
[\\_ip\\_in\\_subnet\(\) \(in module lago.prefix\), 66](#)  
[\\_ipv6\\_prefix\(\) \(lago.providers.libvirt.network.NATNetwork method\), 38](#)  
[\\_lease\\_owned\(\) \(lago.subnet\\_lease.SubnetStore method\), 73](#)  
[\\_lease\\_valid\(\) \(lago.subnet\\_lease.SubnetStore method\), 73](#)  
[\\_libvirt\\_name\(\) \(lago.providers.libvirt.network.Network method\), 38](#)  
[\\_libvirt\\_name\(\) \(lago.providers.libvirt.vm.LocalLibvirtVMProvider method\), 40](#)  
[\\_libvirt\\_xml\(\) \(lago.providers.libvirt.network.BridgeNetwork method\), 38](#)  
[\\_libvirt\\_xml\(\) \(lago.providers.libvirt.network.NATNetwork method\), 38](#)  
[\\_libvirt\\_xml\(\) \(lago.providers.libvirt.network.Network method\), 38](#)  
[\\_libvirt\\_xml\(\) \(lago.providers.libvirt.vm.LocalLibvirtVMProvider method\), 40](#)  
[\\_load\\_xml\(\) \(lago.providers.libvirt.vm.LocalLibvirtVMProvider method\), 40](#)  
[\\_locked\(\) \(in module lago.templates\), 80](#)  
[\\_main\\_thread\\_lock \(lago.log\\_utils.TaskHandler attribute\), 55](#)  
[\\_mutex\(\) \(lago.build.Command class method\), 45](#)  
[\\_max\\_subnet \(lago.subnet\\_lease.SubnetStore attribute\), 73](#)  
[\\_min\\_third\\_octet \(lago.subnet\\_lease.SubnetStore attribute\), 73](#)  
[\\_metadata \(lago.prefix.Prefix attribute\), 59](#)  
[\\_min\\_subnet \(lago.subnet\\_lease.SubnetStore attribute\), 73](#)  
[\\_min\\_third\\_octet \(lago.subnet\\_lease.SubnetStore attribute\), 73](#)  
[\\_nova\\_to\\_spec\(\) \(lago.plugins.vm.VMPlugin class method\), 31](#)  
[\\_ova\\_to\\_spec\(\) \(lago.prefix.Prefix method\), 61](#)  
[\\_path \(lago.subnet\\_lease.Lease attribute\), 72](#)  
[\\_path \(lago.subnet\\_lease.SubnetStore attribute\), 72](#)  
[\\_path\\_to\\_xml\(\) \(in module lago.virt\), 87](#)  
[\\_paths \(lago.prefix.Prefix attribute\), 59](#)  
[\\_prefix\\_path \(lago.paths.Paths attribute\), 58](#)  
[\\_prefixed\(\) \(lago.templates.FileSystemTemplateProvider method\), 76](#)  
[\\_prefixed\(\) \(lago.templates.TemplateStore method\), 79](#)  
[\\_prepare\\_domain\\_image\(\) \(lago.prefix.Prefix method\), 62](#)  
[\\_prepare\\_domains\\_images\(\) \(lago.prefix.Prefix method\), 62](#)  
[\\_providers \(lago.templates.TemplateRepository attribute\), 78](#)  
[\\_realise\\_lease\\_path\(\) \(lago.subnet\\_lease.Lease method\), 72](#)  
[\\_reclaim\\_disk\(\) \(lago.providers.libvirt.vm.LocalLibvirtVMProvider method\), 40](#)  
[\\_reclaim\\_disks\(\) \(lago.providers.libvirt.vm.LocalLibvirtVMProvider method\), 41](#)  
[\\_register\\_preallocated\\_ips\(\) \(lago.prefix.Prefix method\), 62](#)  
[\\_release\(\) \(lago.subnet\\_lease.SubnetStore method\), 74](#)  
[\\_render\\_template\(\) \(in module lago.sysprep\), 75](#)  
[\\_replace\(\) \(lago.build.Command method\), 45](#)  
[\\_request\\_start\(\) \(lago.plugins.service.ServicePlugin method\), 30](#)  
[\\_request\\_start\(\) \(lago.service.SysVInitService method\), 69](#)

`_request_start()` (lago.service.SystemdContainerService method), 69  
`_request_start()` (lago.service.SystemdService method), 70  
`_request_stop()` (lago.plugins.service.ServicePlugin method), 30  
`_request_stop()` (lago.service.SysVInitService method), 69  
`_request_stop()` (lago.service.SystemdContainerService method), 70  
`_request_stop()` (lago.service.SystemdService method), 70  
`_resolve_service_class()` (in module lago.plugins.vm), 35  
`_ret_via_queue()` (in module lago.utils), 82  
`_retrieve_disk_url()` (lago.prefix.Prefix method), 62  
`_run_command()` (in module lago.utils), 82  
`_run_qemu()` (lago.prefix.Prefix static method), 62  
`_save_metadata()` (lago.prefix.Prefix method), 62  
`_scp()` (lago.plugins.vm.VMPlugin method), 31  
`_select_mgmt_networks()` (lago.prefix.Prefix method), 62  
`_set_current()` (lago.workdir.Workdir method), 88  
`_set_link()` (in module lago.brctl), 43  
`_set_mtu_to_nics()` (lago.prefix.Prefix method), 62  
`_set_scripts()` (lago.prefix.Prefix method), 62  
`_shutdown()` (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 41  
`_store_path` (lago.subnet\_lease.Lease attribute), 71  
`_subnet` (lago.subnet\_lease.Lease attribute), 72  
`_subnet_template` (lago.subnet\_lease.SubnetStore attribute), 72  
`_take_lease()` (lago.subnet\_lease.SubnetStore method), 74  
`_tasks_lock` (lago.log\_utils.TaskHandler attribute), 55  
`_template_metadata()` (lago.plugins.vm.VMPlugin method), 31  
`_update_current()` (lago.workdir.Workdir method), 88  
`_use_prototype()` (lago.prefix.Prefix method), 62  
`_validate_lease_dir()` (lago.subnet\_lease.SubnetStore method), 74  
`_validate_netconfig()` (lago.prefix.Prefix method), 62  
`_virt_env` (lago.prefix.Prefix attribute), 59

**A**

`acquire()` (lago.subnet\_lease.SubnetStore method), 74  
`ACTIVE` (lago.plugins.service.ServiceState attribute), 30  
`add_argument()` (lago.plugins.cli.CLIPuginFuncWrapper method), 25  
`add_mapping()` (lago.providers.libvirt.network.Network method), 38  
`add_mappings()` (lago.providers.libvirt.network.Network method), 38  
`add_prefix()` (lago.workdir.Workdir method), 88  
`add_stream_logger()` (in module lago.sdk), 67  
`add_timestamp_suffix()` (in module lago.utils), 82  
`alive()` (lago.plugins.service.ServicePlugin method), 30  
`alive()` (lago.providers.libvirt.network.Network method), 38  
`alive()` (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 41  
`all_ips()` (lago.plugins.vm.VMPlugin method), 31  
`ALWAYS_SHOW_REG` (in module lago.log\_utils), 53  
`ALWAYS_SHOW_TRIGGER_MSG` (in module lago.log\_utils), 53  
`am_i_main_thread` (lago.log\_utils.TaskHandler attribute), 55  
`ansible_inventory()` (lago.sdk.SDK method), 67  
`ansible_inventory_temp_file()` (lago.sdk.SDK method), 67  
`argparse_to_ini()` (in module lago.utils), 82  
`auth_callback()` (in module lago.providers.libvirt.utils), 39

## B

`BIN_PATH` (lago.plugins.service.ServicePlugin attribute), 30  
`BIN_PATH` (lago.service.SystemdContainerService attribute), 69  
`BIN_PATH` (lago.service.SystemdService attribute), 70  
`BIN_PATH` (lago.service.SysVInitService attribute), 69  
`bootstrap()` (lago.plugins.vm.VMPlugin method), 31  
`bootstrap()` (lago.plugins.vm.VMProviderPlugin method), 33  
`bootstrap()` (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 41  
`bootstrap()` (lago.virt.VirtEnv method), 86  
`bootstrap()` (lago.vm.SSHVMProvider method), 87  
`BridgeNetwork` (class in lago.providers.libvirt.network), 38  
`buffer_size` (lago.log\_utils.TaskHandler attribute), 55  
`Build` (class in lago.build), 43  
`build()` (lago.build.Build method), 44  
`build()` (lago.prefix.Prefix method), 63  
`build_cmds` (lago.build.Build attribute), 44  
`BuildException`, 45

## C

`calc_sha()` (lago.export.DiskExportManager method), 48  
`caps` (lago.providers.libvirt.vm.LocalLibvirtVMProvider attribute), 41  
`check_import()` (in module lago.validation), 85  
`check_running()` (in module lago.plugins.vm), 35  
`cleanup()` (lago.prefix.Prefix method), 63  
`cleanup()` (lago.workdir.Workdir method), 89  
`clear()` (lago.utils.RollbackContext method), 82  
`cli_plugin()` (in module lago.plugins.cli), 26  
`cli_plugin_add_argument()` (in module lago.plugins.cli), 26  
`cli_plugin_add_help()` (in module lago.plugins.cli), 27

- CLIPlugin (class in lago.plugins.cli), 25
  - CLIPluginFuncWrapper (class in lago.plugins.cli), 25
  - close\_children\_tasks() (lago.log\_utils.TaskHandler method), 55
  - cmd (lago.build.Command attribute), 45
  - collect\_artifacts() (lago.plugins.vm.VMPlugin method), 32
  - collect\_artifacts() (lago.prefix.Prefix method), 63
  - collect\_paths() (lago.export.VMExportManager method), 50
  - colored() (lago.log\_utils.ColorFormatter class method), 53
  - ColorFormatter (class in lago.log\_utils), 53
  - Command (class in lago.build), 45
  - CommandStatus (class in lago.utils), 81
  - compress (lago.export.VMExportManager attribute), 50
  - compress() (in module lago.utils), 82
  - compress() (lago.export.DiskExportManager method), 48
  - ConfigLoad (class in lago.config), 46
  - CONFS\_PATH (in module lago.constants), 47
  - ContextLock (class in lago.log\_utils), 53
  - copy() (lago.export.DiskExportManager method), 48
  - copy\_from() (lago.plugins.vm.VMPlugin method), 32
  - copy\_to() (lago.plugins.vm.VMPlugin method), 32
  - cp() (in module lago.utils), 82
  - CPU (class in lago.providers.libvirt.cpu), 35
  - cpu (lago.providers.libvirt.vm.LocalLibvirtVMProvider attribute), 41
  - cpu\_model (lago.plugins.vm.VMPlugin attribute), 32
  - cpu\_model (lago.providers.libvirt.vm.LocalLibvirtVMProvider attribute), 41
  - cpu\_vendor (lago.plugins.vm.VMPlugin attribute), 32
  - cpu\_vendor (lago.providers.libvirt.vm.LocalLibvirtVMProvider attribute), 41
  - cpu\_xml (lago.providers.libvirt.cpu.CPU attribute), 35
  - create() (in module lago.brctl), 43
  - create\_lease\_object\_from\_idx() (lago.subnet\_lease.SubnetStore method), 74
  - create\_lease\_object\_from\_subnet() (lago.subnet\_lease.SubnetStore method), 74
  - create\_parser() (in module lago.cmd), 46
  - create\_snapshot() (lago.plugins.vm.VMPlugin method), 32
  - create\_snapshot() (lago.plugins.vm.VMProviderPlugin method), 33
  - create\_snapshot() (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 41
  - create\_snapshot() (lago.vm.SSHVMProvider method), 87
  - create\_snapshots() (lago.prefix.Prefix method), 63
  - create\_snapshots() (lago.virt.VirtEnv method), 86
  - CRITICAL (lago.log\_utils.ColorFormatter attribute), 53
  - cur\_depth\_level (lago.log\_utils.TaskHandler attribute), 55
  - cur\_task (lago.log\_utils.TaskHandler attribute), 55
  - cur\_thread (lago.log\_utils.TaskHandler attribute), 55
  - CYAN (lago.log\_utils.ColorFormatter attribute), 53
- ## D
- DEBUG (lago.log\_utils.ColorFormatter attribute), 53
  - deepcopy() (in module lago.utils), 83
  - DEFAULT (lago.log\_utils.ColorFormatter attribute), 53
  - DefaultOutFormatPlugin (class in lago.plugins.output), 28
  - DefaultVM (class in lago.vm), 87
  - defer() (lago.utils.RollbackContext method), 82
  - defined() (lago.vm.SSHVMProvider method), 87
  - deploy() (lago.prefix.Prefix method), 63
  - destroy() (in module lago.brctl), 43
  - destroy() (lago.prefix.Prefix method), 63
  - destroy() (lago.sdk.SDK method), 67
  - destroy() (lago.workdir.Workdir method), 89
  - dict\_to\_xml() (in module lago.providers.libvirt.utils), 39
  - disk (lago.export.DiskExportManager attribute), 48
  - disk\_path (lago.build.Build attribute), 44
  - disk\_type (lago.export.DiskExportManager attribute), 48
  - DiskExportManager (class in lago.export), 48
  - disks (lago.export.VMExportManager attribute), 49
  - disks (lago.plugins.vm.VMPlugin attribute), 32
  - distro() (lago.plugins.vm.VMPlugin method), 32
  - do\_compress (lago.export.DiskExportManager attribute), 48
  - do\_run() (lago.plugins.cli.CLIPlugin method), 25
  - do\_run() (lago.plugins.cli.CLIPluginFuncWrapper method), 25
  - Domain (class in lago.providers.libvirt.utils), 39
  - DOMAIN\_STATES (in module lago.providers.libvirt.utils), 39
  - download() (lago.templates.TemplateStore method), 79
  - download() (lago.templates.TemplateVersion method), 80
  - download\_image() (lago.templates.FileSystemTemplateProvider method), 76
  - download\_image() (lago.templates.HttpTemplateProvider method), 77
  - drain\_ssh\_channel() (in module lago.ssh), 70
  - dst (lago.export.DiskExportManager attribute), 48
  - dst (lago.export.VMExportManager attribute), 49
  - dump\_level (lago.log\_utils.TaskHandler attribute), 55
- ## E
- EggTimer (class in lago.utils), 81
  - elapsed() (lago.utils.EggTimer method), 81
  - elapsed\_time() (lago.log\_utils.Task method), 54
  - emit() (lago.log\_utils.TaskHandler method), 55
  - end\_log\_task() (in module lago.log\_utils), 57
  - END\_TASK\_MSG (in module lago.log\_utils), 53



END\_TASK\_REG (in module lago.log\_utils), 54  
 END\_TASK\_TRIGGER\_MSG (in module lago.log\_utils), 54  
 ERROR (lago.log\_utils.ColorFormatter attribute), 53  
 ExceptionTimer (class in lago.utils), 81  
 exist (lago.subnet\_lease.Lease attribute), 72  
 exists() (in module lago.brctl), 43  
 exists() (lago.plugins.service.ServicePlugin method), 30  
 exit\_handler() (in module lago.cmd), 46  
 export() (lago.export.DiskExportManager method), 48  
 export() (lago.export.FileExportManager method), 49  
 export() (lago.export.TemplateExportManager method), 49  
 export() (lago.export.VMExportManager method), 50  
 export\_disks() (lago.plugins.vm.VMPlugin method), 32  
 export\_disks() (lago.plugins.vm.VMProviderPlugin method), 34  
 export\_disks() (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 41  
 export\_vms() (lago.prefix.Prefix method), 63  
 export\_vms() (lago.virt.VirtEnv method), 86  
 exported\_disks\_paths() (lago.export.VMExportManager method), 50  
 exported\_metadata (lago.export.DiskExportManager attribute), 48  
 expose() (in module lago.sdk\_utils), 68  
 extract\_image\_xz() (lago.templates.HttpTemplateProvider static method), 77  
 extract\_paths() (in module lago.guestfs\_tools), 50  
 extract\_paths() (lago.plugins.vm.VMPlugin method), 32  
 extract\_paths() (lago.plugins.vm.VMProviderPlugin method), 34  
 extract\_paths() (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 42  
 extract\_paths\_dead() (lago.plugins.vm.VMPlugin method), 32  
 extract\_paths\_dead() (lago.plugins.vm.VMProviderPlugin method), 34  
 extract\_paths\_dead() (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 42  
 ExtractPathError, 31  
 ExtractPathNoPathError, 31

## F

failed (lago.log\_utils.Task attribute), 54  
 fetch\_url() (lago.prefix.Prefix method), 63  
 FileExportManager (class in lago.export), 49  
 FileSystemTemplateProvider (class in lago.templates), 76  
 filter\_spec() (in module lago.utils), 83  
 find\_repo\_by\_name() (in module lago.templates), 80  
 find\_rootfs() (in module lago.guestfs\_tools), 51  
 FlatOutFormatPlugin (class in lago.plugins.output), 28  
 force\_show (lago.log\_utils.Task attribute), 54  
 format() (lago.log\_utils.ColorFormatter method), 53

format() (lago.plugins.output.DefaultOutFormatPlugin method), 28  
 format() (lago.plugins.output.FlatOutFormatPlugin method), 28  
 format() (lago.plugins.output.JSONOutFormatPlugin method), 29  
 format() (lago.plugins.output.OutFormatPlugin method), 29  
 format() (lago.plugins.output.YAMLOutFormatPlugin method), 29  
 formatter (lago.log\_utils.TaskHandler attribute), 55  
 from\_prefix() (lago.virt.VirtEnv class method), 86  
 from\_url() (lago.templates.TemplateRepository class method), 78  
 func\_vector() (in module lago.utils), 83

## G

generate\_cpu\_xml() (lago.providers.libvirt.cpu.CPU method), 35  
 generate\_custom() (lago.providers.libvirt.cpu.CPU method), 35  
 generate\_exact() (lago.providers.libvirt.cpu.CPU method), 36  
 generate\_feature() (lago.providers.libvirt.cpu.CPU method), 36  
 generate\_host\_passthrough() (lago.providers.libvirt.cpu.CPU method), 36  
 generate\_init() (lago.virt.VirtEnv method), 86  
 generate\_numa() (lago.providers.libvirt.cpu.CPU method), 36  
 generate\_topology() (lago.providers.libvirt.cpu.CPU method), 36  
 generate\_vcpu() (lago.providers.libvirt.cpu.CPU method), 37  
 generate\_vcpu\_xml() (lago.providers.libvirt.cpu.CPU method), 37  
 get() (lago.config.ConfigLoad method), 46  
 get\_allowed\_range() (lago.subnet\_lease.SubnetStore method), 75  
 get\_by\_name() (lago.templates.TemplateRepository method), 78  
 get\_cmd\_handler() (lago.build.Build method), 44  
 get\_compat() (lago.virt.VirtEnv method), 86  
 get\_cpu\_props() (lago.providers.libvirt.cpu.LibvirtCPU class method), 37  
 get\_cpu\_vendor() (lago.providers.libvirt.cpu.LibvirtCPU class method), 37  
 get\_cpus\_by\_arch() (lago.providers.libvirt.cpu.LibvirtCPU class method), 37  
 get\_default\_log\_formatter() (in module lago.log\_utils), 57  
 get\_domain\_template() (in module lago.providers.libvirt.utils), 40



- `get_env_dict()` (in module `lago.config`), 47
  - `get_env_spec()` (`lago.virt.VirtEnv` method), 86
  - `get_hash()` (in module `lago.utils`), 83
  - `get_hash()` (`lago.templates.FileSystemTemplateProvider` method), 76
  - `get_hash()` (`lago.templates.HttpTemplateProvider` method), 77
  - `get_hash()` (`lago.templates.TemplateVersion` method), 80
  - `get_ini()` (`lago.config.ConfigLoad` method), 46
  - `get_instance_by_type()` (`lago.export.DiskExportManager` static method), 48
  - `get_instance_from_build_spec()` (`lago.build.Build` class method), 44
  - `get_inventory()` (`lago.lago_ansible.LagoAnsible` method), 52
  - `get_inventory_str()` (`lago.lago_ansible.LagoAnsible` method), 52
  - `get_inventory_temp_file()` (`lago.lago_ansible.LagoAnsible` method), 52
  - `get_key()` (`lago.lago_ansible.LagoAnsible` static method), 52
  - `get_latest_version()` (`lago.templates.Template` method), 78
  - `get_libvirt_connection()` (in module `lago.providers.libvirt.utils`), 40
  - `get_metadata()` (`lago.templates.FileSystemTemplateProvider` method), 76
  - `get_metadata()` (`lago.templates.HttpTemplateProvider` method), 77
  - `get_metadata()` (`lago.templates.TemplateVersion` method), 80
  - `get_net()` (`lago.virt.VirtEnv` method), 86
  - `get_nets()` (`lago.prefix.Prefix` method), 64
  - `get_nets()` (`lago.virt.VirtEnv` method), 86
  - `get_path()` (`lago.templates.TemplateStore` method), 79
  - `get_paths()` (`lago.prefix.Prefix` method), 64
  - `get_prefix()` (`lago.workdir.Workdir` method), 89
  - `get_qemu_info()` (in module `lago.utils`), 83
  - `get_section()` (`lago.config.ConfigLoad` method), 46
  - `get_snapshots()` (`lago.prefix.Prefix` method), 64
  - `get_snapshots()` (`lago.virt.VirtEnv` method), 86
  - `get_ssh_client()` (in module `lago.ssh`), 70
  - `get_stored_hash()` (`lago.templates.TemplateStore` method), 80
  - `get_stored_metadata()` (`lago.templates.TemplateStore` method), 80
  - `get_task_indicator()` (`lago.log_utils.TaskHandler` method), 55
  - `get_tasks()` (`lago.log_utils.TaskHandler` method), 56
  - `get_template()` (in module `lago.providers.libvirt.utils`), 40
  - `get_version()` (`lago.templates.Template` method), 78
  - `get_vm()` (`lago.virt.VirtEnv` method), 86
  - `get_vms()` (`lago.prefix.Prefix` method), 64
  - `get_vms()` (`lago.virt.VirtEnv` method), 86
  - `getattr_sdk()` (in module `lago.sdk_utils`), 68
  - `GREEN` (`lago.log_utils.ColorFormatter` attribute), 53
  - `groups` (`lago.plugins.vm.VMPlugin` attribute), 32
  - `guest_agent()` (`lago.plugins.vm.VMPlugin` method), 32
  - `guestfs_conn_mount_ro()` (in module `lago.guestfs_tools`), 51
  - `guestfs_conn_ro()` (in module `lago.guestfs_tools`), 51
  - `GuestFSError`, 50
  - `gw()` (`lago.providers.libvirt.network.Network` method), 38
- ## H
- `handle_closed_task()` (`lago.log_utils.TaskHandler` method), 56
  - `handle_error()` (`lago.log_utils.TaskHandler` method), 56
  - `handle_new_task()` (`lago.log_utils.TaskHandler` method), 56
  - `has_env` (`lago.subnet_lease.Lease` attribute), 72
  - `has_guest_agent()` (`lago.plugins.vm.VMPlugin` method), 32
  - `hide_paramiko_logs()` (in module `lago.log_utils`), 57
  - `hide_stevedore_logs()` (in module `lago.log_utils`), 57
  - `HOST_BIN_PATH` (`lago.service.SystemdContainerService` attribute), 69
  - `HttpTemplateProvider` (class in `lago.templates`), 77
- ## I
- `images()` (`lago.paths.Paths` method), 58
  - `in_prefix()` (in module `lago.utils`), 83
  - `INACTIVE` (`lago.plugins.service.ServiceState` attribute), 30
  - `indent_unit` (`lago.plugins.output.DefaultOutFormatPlugin` attribute), 28
  - `INFO` (`lago.log_utils.ColorFormatter` attribute), 53
  - `init()` (in module `lago.sdk`), 68
  - `init_args` (`lago.plugins.cli.CLIPlugin` attribute), 25
  - `init_args` (`lago.plugins.cli.CLIPluginFuncWrapper` attribute), 26
  - `initial_depth` (`lago.log_utils.TaskHandler` attribute), 55
  - `initialize()` (`lago.prefix.Prefix` method), 64
  - `initialize()` (`lago.workdir.Workdir` method), 89
  - `interactive_console()` (`lago.plugins.vm.VMPlugin` method), 32
  - `interactive_console()` (`lago.plugins.vm.VMProviderPlugin` method), 34
  - `interactive_console()` (`lago.providers.libvirt.vm.LocalLibvirtVMProvider` method), 42
  - `interactive_ssh()` (in module `lago.ssh`), 71
  - `interactive_ssh()` (`lago.plugins.vm.VMPlugin` method), 32
  - `interactive_ssh_channel()` (in module `lago.ssh`), 71
  - `invoke_different_funcs_in_parallel()` (in module `lago.utils`), 83

[invoke\\_in\\_parallel\(\)](#) (in module [lago.utils](#)), 83  
[ip\(\)](#) ([lago.plugins.vm.VMPlugin](#) method), 32  
[ips\\_in\\_net\(\)](#) ([lago.plugins.vm.VMPlugin](#) method), 32  
[ipv4\\_to\\_mac\(\)](#) (in module [lago.utils](#)), 83  
[is\\_leasable\\_subnet\(\)](#) ([lago.subnet\\_lease.SubnetStore](#) method), 75  
[is\\_management\(\)](#) ([lago.providers.libvirt.network.Network](#) method), 38  
[is\\_possible\\_workdir\(\)](#) ([lago.workdir.Workdir](#) static method), 89  
[is\\_prefix\(\)](#) ([lago.prefix.Prefix](#) class method), 64  
[is\\_supported\(\)](#) ([lago.plugins.service.ServicePlugin](#) class method), 30  
[is\\_workdir\(\)](#) ([lago.workdir.Workdir](#) class method), 89  
[iscsi\\_name\(\)](#) ([lago.plugins.vm.VMPlugin](#) method), 32

## J

[join\(\)](#) ([lago.workdir.Workdir](#) method), 90  
[join\\_all\(\)](#) ([lago.utils.VectorThread](#) method), 82  
[json\\_dump\(\)](#) (in module [lago.utils](#)), 83  
[JSONOutFormatPlugin](#) (class in [lago.plugins.output](#)), 29

## L

[lago](#) (module), 23  
[lago.brctl](#) (module), 43  
[lago.build](#) (module), 43  
[lago.cmd](#) (module), 46  
[lago.config](#) (module), 46  
[lago.constants](#) (module), 47  
[lago.export](#) (module), 48  
[lago.guestfs\\_tools](#) (module), 50  
[lago.lago\\_ansible](#) (module), 52  
[lago.log\\_utils](#) (module), 53  
[lago.paths](#) (module), 58  
[lago.plugins](#) (module), 23  
[lago.plugins.cli](#) (module), 24  
[lago.plugins.output](#) (module), 28  
[lago.plugins.service](#) (module), 30  
[lago.plugins.vm](#) (module), 31  
[lago.prefix](#) (module), 59  
[lago.providers](#) (module), 35  
[lago.providers.libvirt](#) (module), 35  
[lago.providers.libvirt.cpu](#) (module), 35  
[lago.providers.libvirt.network](#) (module), 38  
[lago.providers.libvirt.utils](#) (module), 39  
[lago.providers.libvirt.vm](#) (module), 40  
[lago.sdk](#) (module), 67  
[lago.sdk\\_utils](#) (module), 68  
[lago.service](#) (module), 69  
[lago.ssh](#) (module), 70  
[lago.subnet\\_lease](#) (module), 71  
[lago.sysprep](#) (module), 75  
[lago.templates](#) (module), 76  
[lago.utils](#) (module), 81

[lago.validation](#) (module), 85  
[lago.virt](#) (module), 85  
[lago.vm](#) (module), 87  
[lago.workdir](#) (module), 88  
[LagoAnsible](#) (class in [lago.lago\\_ansible](#)), 52  
[LagoException](#), 81  
[LagoFailedToGetVMStateError](#), 31  
[LagoInitException](#), 81  
[LagoLocalLibvirtVMProviderException](#), 40  
[LagoSSTimeoutException](#), 70  
[LagoSubnetLeaseBadPermissionsException](#), 71  
[LagoSubnetLeaseException](#), 71  
[LagoSubnetLeaseLockException](#), 71  
[LagoSubnetLeaseMalformedAddrException](#), 71  
[LagoSubnetLeaseOutOfRangeException](#), 71  
[LagoSubnetLeaseStoreFullException](#), 71  
[LagoSubnetLeaseTakenException](#), 71  
[LagoUserException](#), 81  
[LagoVMDoesNotExistError](#), 31  
[LagoVMNotRunningError](#), 31  
[Lease](#) (class in [lago.subnet\\_lease](#)), 71  
[level](#) ([lago.log\\_utils.TaskHandler](#) attribute), 55  
[LIBEXEC\\_DIR](#) (in module [lago.constants](#)), 47  
[libvirt\\_callback\(\)](#) (in module [lago.providers.libvirt.utils](#)), 40  
[LIBVIRT\\_CONNECTIONS](#) (in module [lago.providers.libvirt.utils](#)), 39  
[libvirt\\_ver](#) ([lago.providers.libvirt.vm.LocalLibvirtVMProvider](#) attribute), 42  
[LibvirtCPU](#) (class in [lago.providers.libvirt.cpu](#)), 37  
[list\\_leases\(\)](#) ([lago.subnet\\_lease.SubnetStore](#) method), 75  
[load\(\)](#) ([lago.config.ConfigLoad](#) method), 47  
[load\(\)](#) ([lago.workdir.Workdir](#) method), 90  
[load\\_env\(\)](#) (in module [lago.sdk](#)), 68  
[load\\_plugins\(\)](#) (in module [lago.plugins](#)), 23  
[load\\_virt\\_stream\(\)](#) (in module [lago.utils](#)), 83  
[LocalLibvirtVMProvider](#) (class in [lago.providers.libvirt.vm](#)), 40  
[LockFile](#) (class in [lago.utils](#)), 81  
[log\\_always\(\)](#) (in module [lago.log\\_utils](#)), 57  
[log\\_task\(\)](#) (in module [lago.log\\_utils](#)), 58  
[logs\(\)](#) ([lago.paths.Paths](#) method), 58  
[LogTask](#) (class in [lago.log\\_utils](#)), 54

## M

[main\(\)](#) (in module [lago.cmd](#)), 46  
[main\\_failed](#) ([lago.log\\_utils.TaskHandler](#) attribute), 55  
[MalformedWorkdir](#), 88  
[mapping\(\)](#) ([lago.providers.libvirt.network.Network](#) method), 38  
[mark\\_main\\_tasks\\_as\\_failed\(\)](#) ([lago.log\\_utils.TaskHandler](#) method), 56  
[mark\\_parent\\_tasks\\_as\\_failed\(\)](#) ([lago.log\\_utils.TaskHandler](#) method), 56

metadata (lago.plugins.vm.VMPlugin attribute), 32  
 metadata (lago.prefix.Prefix attribute), 64  
 metadata (lago.subnet\_lease.Lease attribute), 72  
 metadata() (lago.paths.Paths method), 58  
 mgmt\_name (lago.plugins.vm.VMPlugin attribute), 32  
 mgmt\_net (lago.plugins.vm.VMPlugin attribute), 32  
 MISSING (lago.plugins.service.ServiceState attribute), 30  
 model (lago.providers.libvirt.cpu.CPU attribute), 37  
 mtu() (lago.providers.libvirt.network.Network method), 39

## N

name (lago.build.Build attribute), 44  
 name (lago.build.Command attribute), 45  
 name (lago.export.DiskExportManager attribute), 48  
 name (lago.log\_utils.Task attribute), 54  
 name (lago.templates.Template attribute), 77  
 name (lago.templates.TemplateRepository attribute), 78  
 name() (lago.plugins.vm.VMPlugin method), 32  
 name() (lago.plugins.vm.VMProviderPlugin method), 34  
 name() (lago.providers.libvirt.network.Network method), 39  
 NATNetwork (class in lago.providers.libvirt.network), 38  
 nets() (lago.plugins.vm.VMPlugin method), 32  
 Network (class in lago.providers.libvirt.network), 38  
 nics() (lago.plugins.vm.VMPlugin method), 32  
 NONE (lago.log\_utils.ColorFormatter attribute), 53  
 normalize\_build\_spec() (lago.build.Build method), 44  
 normalize\_options() (lago.build.Build static method), 44  
 NoSuchPluginError, 23

## O

open\_url() (lago.templates.HttpTemplateProvider method), 77  
 OutFormatPlugin (class in lago.plugins.output), 29

## P

path (lago.subnet\_lease.Lease attribute), 72  
 path (lago.subnet\_lease.SubnetStore attribute), 75  
 Paths (class in lago.paths), 58  
 paths (lago.build.Build attribute), 44  
 paths (lago.prefix.Prefix attribute), 64  
 Plugin (class in lago.plugins), 23  
 PLUGIN\_ENTRY\_POINTS (in module lago.plugins), 23  
 PluginError, 23  
 populate\_parser() (lago.plugins.cli.CLIPugin method), 25  
 populate\_parser() (lago.plugins.cli.CLIPuginFuncWrapper method), 26  
 Prefix (class in lago.prefix), 59  
 prefix (lago.lago.ansible.LagoAnsible attribute), 52  
 prefix\_lagofile() (lago.paths.Paths method), 58  
 prefix\_option() (lago.build.Build static method), 45

prefix\_path() (lago.paths.Paths method), 58  
 PrefixAlreadyExists, 88  
 prefixed() (lago.paths.Paths method), 58  
 prefixed\_name() (lago.virt.VirtEnv method), 87  
 PrefixNotFound, 88  
 prependDefer() (lago.utils.RollbackContext method), 82  
 pretty\_emit() (lago.log\_utils.TaskHandler method), 56

## Q

qemu\_rebase() (in module lago.utils), 84

## R

raw\_state() (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 42  
 read\_nonblocking() (in module lago.utils), 84  
 rebase() (lago.export.TemplateExportManager method), 49  
 reboot() (lago.plugins.vm.VMPlugin method), 32  
 reboot() (lago.plugins.vm.VMProviderPlugin method), 34  
 reboot() (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 43  
 RED (lago.log\_utils.ColorFormatter attribute), 53  
 release() (lago.subnet\_lease.SubnetStore method), 75  
 resolve() (lago.providers.libvirt.network.Network method), 39  
 resolve\_parent() (lago.prefix.Prefix method), 64  
 resolve\_prefix\_path() (lago.prefix.Prefix class method), 65  
 resolve\_state() (lago.providers.libvirt.utils.Domain static method), 39  
 resolve\_workdir\_path() (lago.workdir.Workdir class method), 90  
 revert\_snapshot() (lago.plugins.vm.VMPlugin method), 32  
 revert\_snapshot() (lago.plugins.vm.VMProviderPlugin method), 34  
 revert\_snapshot() (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 43  
 revert\_snapshot() (lago.virt.SSHVMProvider method), 87  
 revert\_snapshots() (lago.prefix.Prefix method), 65  
 revert\_snapshots() (lago.virt.VirtEnv method), 87  
 RollbackContext (class in lago.utils), 81  
 root\_password() (lago.plugins.vm.VMPlugin method), 32  
 rotate\_dir() (in module lago.utils), 84  
 run\_command() (in module lago.utils), 84  
 run\_command\_with\_validation() (in module lago.utils), 84  
 run\_interactive\_command() (in module lago.utils), 84  
 running() (lago.plugins.vm.VMPlugin method), 33  
 running() (lago.plugins.vm.VMProviderPlugin method), 34  
 running() (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 43

running() (lago.vm.SSHVMProvider method), 88

## S

save() (lago.plugins.vm.VMPlugin method), 33

save() (lago.prefix.Prefix method), 65

save() (lago.providers.libvirt.network.Network method), 39

save() (lago.virt.VirtEnv method), 87

scripts() (lago.paths.Paths method), 58

SDK (class in lago.sdk), 67

SDKMethod (class in lago.sdk\_utils), 68

service() (lago.plugins.vm.VMPlugin method), 33

service\_is\_enabled() (in module lago.utils), 85

ServicePlugin (class in lago.plugins.service), 30

ServiceState (class in lago.plugins.service), 30

set\_current() (lago.workdir.Workdir method), 90

set\_help() (lago.plugins.cli.CLIPuginFuncWrapper method), 26

set\_init\_args() (lago.plugins.cli.CLIPuginFuncWrapper method), 26

setup\_prefix\_logging() (in module lago.log\_utils), 58

setup\_sdk\_logging() (in module lago.sdk\_utils), 69

should\_show\_by\_depth() (lago.log\_utils.TaskHandler method), 57

should\_show\_by\_level() (lago.log\_utils.TaskHandler method), 57

shutdown() (lago.plugins.vm.VMPlugin method), 33

shutdown() (lago.plugins.vm.VMProviderPlugin method), 34

shutdown() (lago.prefix.Prefix method), 65

shutdown() (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 43

shutdown() (lago.virt.VirtEnv method), 87

sparse() (in module lago.utils), 85

sparse() (lago.export.DiskExportManager method), 49

spec (lago.plugins.vm.VMPlugin attribute), 33

spec (lago.providers.libvirt.network.Network attribute), 39

src (lago.export.DiskExportManager attribute), 48

src\_qemu\_info (lago.export.FileExportManager attribute), 49

ssh() (in module lago.ssh), 71

ssh() (lago.plugins.vm.VMPlugin method), 33

ssh\_id\_rsa() (lago.paths.Paths method), 58

ssh\_id\_rsa\_pub() (lago.paths.Paths method), 58

ssh\_reachable() (lago.plugins.vm.VMPlugin method), 33

ssh\_script() (in module lago.ssh), 71

ssh\_script() (lago.plugins.vm.VMPlugin method), 33

SSHVMProvider (class in lago.vm), 87

standalone (lago.export.FileExportManager attribute), 49

start() (lago.plugins.service.ServicePlugin method), 30

start() (lago.plugins.vm.VMPlugin method), 33

start() (lago.plugins.vm.VMProviderPlugin method), 35

start() (lago.prefix.Prefix method), 65

start() (lago.providers.libvirt.network.BridgeNetwork method), 38

start() (lago.providers.libvirt.network.Network method), 39

start() (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 43

start() (lago.utils.ExceptionTimer method), 81

start() (lago.virt.VirtEnv method), 87

start() (lago.vm.SSHVMProvider method), 88

start\_all() (lago.utils.VectorThread method), 82

start\_log\_task() (in module lago.log\_utils), 58

START\_TASK\_MSG (in module lago.log\_utils), 54

START\_TASK\_REG (in module lago.log\_utils), 54

START\_TASK\_TRIGGER\_MSG (in module lago.log\_utils), 54

state() (lago.plugins.service.ServicePlugin method), 30

state() (lago.plugins.vm.VMPlugin method), 33

state() (lago.plugins.vm.VMProviderPlugin method), 35

state() (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 43

state() (lago.service.SystemdContainerService method), 70

state() (lago.service.SystemdService method), 70

state() (lago.service.SysVInitService method), 69

state() (lago.vm.SSHVMProvider method), 88

stop() (lago.plugins.service.ServicePlugin method), 30

stop() (lago.plugins.vm.VMPlugin method), 33

stop() (lago.plugins.vm.VMProviderPlugin method), 35

stop() (lago.prefix.Prefix method), 65

stop() (lago.providers.libvirt.network.BridgeNetwork method), 38

stop() (lago.providers.libvirt.network.Network method), 39

stop() (lago.providers.libvirt.vm.LocalLibvirtVMProvider method), 43

stop() (lago.utils.ExceptionTimer method), 81

stop() (lago.virt.VirtEnv method), 87

stop() (lago.vm.SSHVMProvider method), 88

subnet (lago.subnet\_lease.Lease attribute), 72

SubnetStore (class in lago.subnet\_lease), 72

sysprep() (in module lago.sysprep), 75

SystemdContainerService (class in lago.service), 69

SystemdService (class in lago.service), 70

SysVInitService (class in lago.service), 69

## T

Task (class in lago.log\_utils), 54

TASK\_INDICATORS (lago.log\_utils.TaskHandler attribute), 55

task\_tree\_depth (lago.log\_utils.TaskHandler attribute), 55

TaskHandler (class in lago.log\_utils), 54

tasks (lago.log\_utils.TaskHandler attribute), 57

Template (class in lago.templates), 77

TemplateExportManager (class in lago.export), 49

TemplateRepository (class in lago.templates), 78  
 TemplateStore (class in lago.templates), 78  
 TemplateVersion (class in lago.templates), 80  
 TimerException, 82  
 timestamp() (lago.templates.TemplateVersion method), 80  
 to\_ip\_network() (lago.subnet\_lease.Lease method), 72

## U

update\_args() (lago.config.ConfigLoad method), 47  
 update\_lago\_metadata() (lago.export.DiskExportManager method), 49  
 update\_lago\_metadata() (lago.export.TemplateExportManager method), 49  
 update\_parser() (lago.config.ConfigLoad method), 47  
 uuid (lago.subnet\_lease.Lease attribute), 72  
 uuid() (lago.paths.Paths method), 58  
 uuid\_path (lago.subnet\_lease.Lease attribute), 72

## V

valid (lago.subnet\_lease.Lease attribute), 72  
 validate() (lago.providers.libvirt.cpu.CPU method), 37  
 vcpu\_xml (lago.providers.libvirt.cpu.CPU attribute), 37  
 VectorThread (class in lago.utils), 82  
 vendor (lago.providers.libvirt.cpu.CPU attribute), 37  
 ver\_cmp() (in module lago.utils), 85  
 virt() (lago.paths.Paths method), 58  
 virt\_conf() (lago.prefix.Prefix method), 65  
 virt\_conf\_from\_stream() (lago.prefix.Prefix method), 66  
 virt\_customize() (lago.build.Build method), 45  
 virt\_env (lago.prefix.Prefix attribute), 66  
 VIRT\_ENV\_CLASS (lago.prefix.Prefix attribute), 59  
 virt\_path() (lago.virt.VirtEnv method), 87  
 VirtEnv (class in lago.virt), 85  
 vm\_type (lago.plugins.vm.VMPlugin attribute), 33  
 VMError, 31  
 VMExportManager (class in lago.export), 49  
 VMPlugin (class in lago.plugins.vm), 31  
 VMProviderPlugin (class in lago.plugins.vm), 33

## W

wait\_for\_ssh() (in module lago.ssh), 71  
 wait\_for\_ssh() (lago.plugins.vm.VMPlugin method), 33  
 WARNING (lago.log\_utils.ColorFormatter attribute), 53  
 WHITE (lago.log\_utils.ColorFormatter attribute), 53  
 with\_logging() (in module lago.utils), 85  
 with\_threads (lago.export.VMExportManager attribute), 50  
 Workdir (class in lago.workdir), 88  
 workdir\_loaded() (in module lago.workdir), 90  
 WorkdirError, 90  
 write\_lago\_metadata() (lago.export.DiskExportManager method), 49

## Y

YAMLOutFormatPlugin (class in lago.plugins.output), 29  
 YELLOW (lago.log\_utils.ColorFormatter attribute), 53