Lago Documentation

Release 0.3

David Caro

December 17, 2015

Contents

CHAPTER 1

Getting started

Check out the awesome README!

CHAPTER 2

Releases

2.1 Release process

2.1.1 Versioning

For lago we use a similar approach to semantic versioning, that is:

```
X.Y[.Z]
```

For example:

```
0.1
1.2.123
2.0
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
```

2.1.2 RPM Versioning

The rpm versions differ from the generic version in that they have the `-release` suffix, that has the value:

- **If not in a tagged commit:** version = `X.Y.Z-1.git_hash` (if it bothers anyone, we can drop the hash easily)

- **If in a tagged commit:** version = `X.Y.Z-1`

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)

2.1.3 Repository layout

Tree schema of the repository:

```
lago
-- stable <-- subdirs for each major version to avoid accidental
|   |          non-backwards compatible ugrade
|   |
|   -- 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
    |   |           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
        -- ...
```

2.1.4 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 -> 2.0`)
- New minor version bump (`X.Y+1`, for exampyre `1.1 -> 1.2`)
- If it passed certain time since the last `X` or `Y` version bumps (`X.Y.Z+n`, for example `1.0.1 -> 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`)

2.1.5 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 3

Contents

3.1 lago package

class lago.**Prefix**(*prefix*)
Bases: object

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

_prefix
str

Path to the directory of this prefix

_paths
lago.path.Paths

Path handler class

_virt_env
lago.virt.VirtEnv

Lazily loaded virtual env handler

_metadata
dict

Lazily loaded metadata

_add_nic_to_mapping(*net*, *dom*, *nic*)
Populates the given net spec mapping entry with the nicks of the given domain

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

Return type list

_check_predefined_subnets(*conf*)
Checks if all of the nets defined in the config are inside the allowed range, throws exception if not

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

Returns None

Raises RuntimeError – If there are any subnets out of the allowed range

_config_net_topology(*conf*)
Initialize and populate all the network related elements, like reserving ips and populating network specs of the given confiiguration spec

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

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 **Tuple** – Path with the disk and metadata

Return type str, dict

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

_create_paths()
Get the path handler for this instance

Returns Path handler

Return type *lago.paths.Paths*

_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*

_get_metadata()
Retrieve the metadata info for this prefix

Returns metadata info

Return type dict

_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 None

_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

_save_metadata()
Write this prefix metadata to disk

Returns None

_use_prototype(*spec*, *conf*)
Populates the given spec with the values of it's declared prototype

Parameters

- **spec** (*dict*) – spec to update
- **conf** (*dict*) – Configuration spec containing the prototypes

Returns updated spec

Return type dict

cleanup()
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

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

initialize()
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

paths
Access the path handler for this prefix

Returns Path handler

Return type *lago.paths.Paths*

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

start()
Start this prefix

Returns None

stop()
Stop this prefix

Returns None

virt_conf(*conf*, *template_repo=None*, *template_store=None*)
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 intance
- **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.**_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.**_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

3.1.1 Submodules

3.1.2 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*)

3.1.3 lago.config module

lago.config.**_get_environ**()

lago.config.**_get_from_dir**(*path*, *key*)

lago.config.**_get_from_env**(*key*)

lago.config.**_get_from_files**(*paths*, *key*)

lago.config.**_get_providers**()

lago.config.**get**(*key*, *default=<object object>*)

3.1.4 lago.constants module

3.1.5 lago.dirlock module

lago.dirlock.**_lock_path**(*path*)

lago.dirlock.**lock**(*path*, *excl*, *key_path*)

lago.dirlock.**trylock**(*path*, *excl*, *key_path*)

lago.dirlock.**unlock**(*path*, *key_path*)

3.1.6 lago.paths module

class lago.paths.**Paths**(*prefix*)
Bases: object

_prefixed(**args*)

images(**path*)

logs()

metadata()

prefix()

prefix_lagofile()
This file represents a prefix that's initialized

ssh_id_rsa()

ssh_id_rsa_pub()

uuid()

virt(**path*)

3.1.7 lago.subnet_lease module

Module that handles the leases for the subnets of the virtual network interfaces.

Note: Currently only /24 ranges are handled, and all of them under the 192.168.MIN_SUBNET to 192.168.MAX_SUBNET ranges

The leases are stored under *LEASE_DIR* 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)

lago.subnet_lease.**LEASE_DIR = '/var/lib/lago/subnets/'**
Path to the directory where the net leases are stored

lago.subnet_lease.**LOCK_FILE = '/var/lib/lago/subnets/leases.lock'**
Path to the net leases lock

lago.subnet_lease.**MAX_SUBNET = 209**
Upper range for the allowed subnets

lago.subnet_lease.**MIN_SUBNET = 200**
Lower range for the allowed subnets

lago.subnet_lease.**_acquire**(**args*, ***kwargs*)
Lease a free network for the given uuid path

Parameters **uuid_path** (*str*) – Path to the uuid file of a *lago.Prefix*

Returns the third element of the dotted ip of the leased network or None if no lease was available

Return type int or None

Todo

Raise exception or something instead of returning None so the caller can handle the failure case

lago.subnet_lease.**_lease_owned**(*path*, *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 ownersip of

Returns `True` if the given lease in owned by the prefix, `False` otherwise

Return type bool

lago.subnet_lease.**_lease_valid**(*path*)
Checs if the given lease still has a prefix that owns it

Parameters **path** (*str*) – Path to the lease

Returns `True` if the uuid path in the lease still exists and is the same as the one in the lease

Return type bool

lago.subnet_lease.**_locked**(*func*)
Decorator that will make sure that you have the exclusive lock for the leases

lago.subnet_lease.**_release**(**args*, ***kwargs*)
Free the lease of the given subnet index

Parameters **index** (*int*) – Third element of a dotted ip representation of the subnet, for example, for 1.2.3.4 it would be 3

Returns None

lago.subnet_lease.**_take_lease**(*path*, *uuid_path*)
Persist to the given leases path the prefix uuid that's in the uuid path passed

Parameters

- **path** (*str*) – Path to the leases file
- **uuid_path** (*str*) – Path to the prefix uuid

Returns None

lago.subnet_lease.**_validate_lease_dir_present**(*func*)
Decorator that will ensure that the lease dir exists, creating it if necessary

lago.subnet_lease.**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 the dotted ip of the gateway for the leased net

Return type str

Todo

_aquire might return None, this will throw a TypeError

lago.subnet_lease.**is_leasable_subnet**(*subnet*)
Checks if a given subnet is inside the defined provisionable range

Parameters **subnet** (*str*) – Subnet or ip in dotted decimal format

Returns True if subnet is inside the range, `False` otherwise

Return type bool

lago.subnet_lease.**release**(*subnet*)
Free the lease of the given subnet

Parameters **subnet** (*str*) – dotted ip or network to free the lease of

Returns None

3.1.8 lago.sysprep module

lago.sysprep.**_config_net_interface**(*iface*, ***kwargs*)

lago.sysprep.**_upload_file**(*local_path*, *remote_path*)

lago.sysprep.**_write_file**(*path*, *content*)

lago.sysprep.**add_ssh_key**(*key*, *with_restorecon_fix=False*)

lago.sysprep.**config_net_interface_dhcp**(*iface*, *hwaddr*)

lago.sysprep.**set_hostname**(*hostname*)

lago.sysprep.**set_iscsi_initiator_name**(*name*)

lago.sysprep.**set_root_password**(*password*)

lago.sysprep.**set_selinux_mode**(*mode*)

lago.sysprep.**sysprep**(*disk*, *mods*)

3.1.9 lago.templates module

3.1.10 lago.utils module

class lago.utils.**CommandStatus**
Bases: *lago.utils.CommandStatus*

class lago.utils.**EggTimer**(*timeout*)

elapsed()

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 @ vdsm 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*)

class lago.utils.**VectorThread**(*targets*)

join_all(*raise_exceptions=True*)

start_all()

lago.utils.**_CommandStatus**
alias of *CommandStatus*

lago.utils.**_read_nonblocking**(*f*)

lago.utils.**_ret_via_queue**(*func*, *queue*)

lago.utils.**drain_ssh_channel**(*chan*, *stdin=None*, *stdout=<open file '<stdout>', mode 'w'>*, *stderr=<open file '<stderr>', mode 'w'>*)

lago.utils.**func_vector**(*target*, *args_sequence*)

lago.utils.**interactive_ssh_channel**(*chan*, *command=None*, *stdin=<open file '<stdin>', mode 'r'>*)

lago.utils.**invoke_in_parallel**(*func*, **args_sequences*)

lago.utils.**json_dump**(*obj*, *f*)

lago.utils.**run_command**(*command*, *input_data=None*, *out_pipe=-1*, *err_pipe=-1*, *env=None*, ***kwargs*)

lago.utils.**service_is_enabled**(*name*)

lago.utils.**setup_logging**(*logdir*)

3.1.11 lago.virt module

class lago.virt.**BridgeNetwork**(*env*, *spec*)
Bases: *lago.virt.Network*

_libvirt_xml()

start()

stop()

class lago.virt.**NATNetwork**(*env*, *spec*)
Bases: *lago.virt.Network*

_libvirt_xml()

class lago.virt.**Network**(*env*, *spec*)
Bases: object

_libvirt_name()

add_mapping(*name*, *ip*, *save=True*)

add_mappings(*mappings*)

alive()

gw()

is_management()

name()

resolve(*name*)

save()

start()

stop()

class lago.virt.**ServiceState**

ACTIVE = 2

INACTIVE = 1

MISSING = 0

class lago.virt.**VM**(*env*, *spec*)
Bases: object

VM properties: * name * cpus * memory * disks * metadata * network/mac addr

_check_alive(*func*)

_create_dead_snapshot(*name*)

_create_live_snapshot(*name*)

_detect_service_manager()

_get_ssh_client(**args*, ***kwargs*)

_libvirt_name()

_libvirt_xml()

classmethod _normalize_spec(*spec*)

_open_ssh_client()

_reclaim_disk(*path*)

_reclaim_disks()

_sftp(**args*, ***kwds*)

_template_metadata()

alive()

bootstrap()

copy_from(*remote_path*, *local_path*)

copy_to(*local_path*, *remote_path*)

create_snapshot(*name*)

distro()

extract_paths(*paths*)

guest_agent()

interactive_ssh(**args*, ***kwargs*)

ip()

iscsi_name()

metadata

name()

nics()

revert_snapshot(*name*)

root_password()

save(*path=None*)

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

ssh(*command*, *data=None*, *show_output=True*)

ssh_script(*path*, *show_output=True*)

start()

stop()

virt_env()

vnc_port(**args*, ***kwargs*)

wait_for_ssh(*connect_retries=50*)

class lago.virt.**VirtEnv**(*prefix*, *vm_specs*, *net_specs*)
Bases: object

Env properties: * prefix * vms * net

- libvirt_con

_create_net(*net_spec*)

_create_vm(*vm_spec*)

bootstrap()

create_snapshots(*name*)

classmethod from_prefix(*prefix*)

get_net(*name=None*)

get_nets()

get_vm(*name*)

get_vms()

libvirt_con

prefixed_name(*unprefixed_name*)

revert_snapshots(*name*)

save()

start()

stop()

virt_path(**args*)

class lago.virt.**_Service**(*vm*, *name*)

alive()

exists()

classmethod is_supported(*vm*)

start()

stop()

class lago.virt.**_SysVInitService**(*vm*, *name*)
Bases: *lago.virt._Service*

BIN_PATH = '/sbin/service'

_request_start()

_request_stop()

state()

class lago.virt.**_SystemdContainerService**(*vm*, *name*)
Bases: *lago.virt._Service*

BIN_PATH = '/usr/bin/docker'

HOST_BIN_PATH = '/usr/bin/systemctl'

_request_start()

_request_stop()

state()

class lago.virt.**_SystemdService**(*vm*, *name*)
Bases: *lago.virt._Service*

BIN_PATH = '/usr/bin/systemctl'

_request_start()

_request_stop()

state()

lago.virt.**_gen_ssh_command_id**()

lago.virt.**_guestfs_copy_path**(*g*, *guest_path*, *host_path*)

lago.virt.**_ip_to_mac**(*ip*)

lago.virt.**_path_to_xml**(*basename*)

3.2 ovirtlago package

3.2.1 Submodules

3.2.2 ovirtlago.constants module

3.2.3 ovirtlago.merge_repos module

3.2.4 ovirtlago.paths module

3.2.5 ovirtlago.repoverify module

3.2.6 ovirtlago.testlib module

3.2.7 ovirtlago.utils module

3.2.8 ovirtlago.virt module

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

l

Symbols

P

R

S

T

U

V

W