Lorax Documentation

Release 24.1

Anaconda Team

August 12, 2015

Contents

Contents:

CHAPTER 1

Introduction to Lorax

I am the Lorax. I speak for the trees [and images].

Lorax is used to build the Anaconda Installer boot.iso, it consists of a library, pylorax, a set of templates, and the lorax script. Its operation is driven by a customized set of Mako templates that lists the packages to be installed, steps to execute to remove unneeded files, and creation of the iso for all of the supported architectures.

CHAPTER 2

Before Lorax

Tree building tools such as pungi and revisor rely on 'buildinstall' in anaconda/scripts/ to produce the boot images and other such control files in the final tree. The existing buildinstall scripts written in a mix of bash and Python are unmaintainable. Lorax is an attempt to replace them with something more flexible.

EXISTING WORKFLOW:

pungi and other tools call scripts/buildinstall, which in turn call other scripts to do the image building and data generation. Here's how it currently looks:

-> buildinstall

- process command line options
- write temporary yum.conf to point to correct repo
- find anaconda release RPM
- unpack RPM, pull in those versions of upd-instroot, mk-images, maketreeinfo.py, makestamp.py, and buildinstall

-> call upd-instroot

-> call maketreeinfo.py

-> call mk-images (which figures out which mk-images.ARCH to call)

-> call makestamp.py

- clean up

PROBLEMS:

The existing workflow presents some problems with maintaining the scripts. First, almost all knowledge of what goes in to the stage 1 and stage 2 images lives in upd-instroot. The mk-images* scripts copy things from the root created by upd-instroot in order to build the stage 1 image, though it's not completely clear from reading the scripts.

NEW IDEAS:

Create a new central driver with all information living in Python modules. Configuration files will provide the knowledge previously contained in the upd-instroot and mk-images* scripts.

CHAPTER 3

lorax

The lorax script executes the templates and create the boot.iso

CHAPTER 4

livemedia-creator

Authors Brian C. Lane <bcl@redhat.com>

livemedia-creator uses Anaconda, kickstart and Lorax to create bootable media that use the same install path as a normal system installation. It can be used to make live isos, bootable (partitioned) disk images, tarfiles, and filesystem images for use with virtualization and container solutions like libvirt, docker, and OpenStack.

The general idea is to use virt-install with kickstart and an Anaconda boot.iso to install into a disk image and then use the disk image to create the bootable media.

livemedia-creator –help will describe all of the options available. At the minimum you need:

`--make-iso` to create a final bootable .iso or one of the other `--make-*` options.

`--iso` to specify the Anaconda install media to use with virt-install

`--ks` to select the kickstart file describing what to install.

To use livemedia-creator with virt-install you will need to install the following packages, as well as have libvirtd setup correctly.

- `virt-install`
- `libvirt-python`

If you are going to be using Anaconda directly, with `--no-virt` mode, make sure you have the anaconda package installed. You can use the anaconda-tui package to save a bit of space on the build system.

Conventions used in this document:

`lmc` is an abbreviation for livemedia-creator.

`builder` is the system where livemedia-creator is being run

`image` is the disk image being created by running livemedia-creator

4.1 Quickstart

Run this to create a bootable live iso:

```
sudo livemedia-creator --make-iso \
--iso=/extra/iso/boot.iso --ks=./docs/fedora-livemedia.ks
```

You can run it directly from the lorax git repo like this:

```
sudo PATH=./src/sbin/:$PATH PYTHONPATH=./src/ ./src/sbin/livemedia-creator \
--make-iso --iso=/extra/iso/boot.iso \
--ks=./docs/fedora-livemedia.ks --lorax-templates=./share/
```

If you want to watch the install you can pass `--vnc vnc` and use a vnc client to connect to localhost:0

This is usually a good idea when testing changes to the kickstart. lmc tries to monitor the logs for fatal errors, but may not catch everything.

4.2 How ISO creation works

There are 2 stages, the install stage which produces a disk or filesystem image as its output, and the boot media creation which uses the image as its input. Normally you would run both stages, but it is possible to stop after the install stage, by using `--image-only`, or to skip the install stage and use a previously created disk image by passing `--disk-image` or `--fs-image`

When creating an iso virt-install boots using the passed Anaconda installer iso and installs the system based on the kickstart. The `%post` section of the kickstart is used to customize the installed system in the same way that current spin-kickstarts do.

livemedia-creator monitors the install process for problems by watching the install logs. They are written to the current directory or to the base directory specified by the –logfile command. You can also monitor the install by passing `--vnc vnc` and using a vnc client. This is recommended when first modifying a kickstart, since there are still places where Anaconda may get stuck without the log monitor catching it.

The output from this process is a partitioned disk image. kpartx can be used to mount and examine it when there is a problem with the install. It can also be booted using kvm.

When creating an iso the disk image's / partition is copied into a formatted disk image which is then used as the input to lorax for creation of the final media.

The final image is created by lorax, using the templates in /usr/share/lorax/ or the directory specified by `--lorax-templates`

Currently the standard lorax templates are used to make a bootable iso, but it should be possible to modify them to output other results. They are written using the Mako template system which is very flexible.

4.3 Kickstarts

The docs/ directory includes several example kickstarts, one to create a live desktop iso using GNOME, and another to create a minimal disk image. When creating your own kickstarts you should start with the minimal example, it includes several needed packages that are not always included by dependencies.

Or you can use existing spin kickstarts to create live media with a few changes. Here are the steps I used to convert the Fedora XFCE spin.

1. Flatten the xfce kickstart using ksflatten
2. Add zerombr so you don't get the disk init dialog
3. Add clearpart –all
4. Add swap partition
5. bootloader target
6. Add shutdown to the kickstart

7. Add network –bootproto=dhcp –activate to activate the network This works for F16 builds but for F15 and before you need to pass something on the cmdline that activate the network, like sshd:

   ```
   livemedia-creator --kernel-args="sshd"
   ```

8. Add a root password:

```
rootpw rootme
network --bootproto=dhcp --activate
zerombr
clearpart --all
bootloader --location=mbr
part swap --size=512
shutdown
```

9. In the livesys script section of the %post remove the root password. This really depends on how the spin wants to work. You could add the live user that you create to the %wheel group so that sudo works if you wanted to.

   ```
   passwd -d root > /dev/null
   ```

10. Remove /etc/fstab in %post, dracut handles mounting the rootfs

    ```
    cat /dev/null > /dev/fstab
    ```

    Do this only for live iso's, the filesystem will be mounted read only if there is no /etc/fstab

11. Don't delete initramfs files from /boot in %post
12. Have dracut-config-generic, grub-efi, memtest86+ and syslinux in the package list.
13. Omit dracut-config-rescue from the %package list: `-dracut-config-rescue`

One drawback to using virt-install is that it pulls the packages from the repo each time you run it. To speed things up you either need a local mirror of the packages, or you can use a caching proxy. When using a proxy you pass it to livemedia-creator like this:

```
--proxy=http://proxy.yourdomain.com:3128
```

You also need to use a specific mirror instead of mirrormanager so that the packages will get cached, so your kickstart url would look like:

```
url --url="http://dl.fedoraproject.org/pub/fedora/linux/development/17/x86_64/os/"
```

You can also add an update repo, but don't name it updates. Add –proxy to it as well.

4.4 Anaconda image install (no-virt)

You can create images without using virt-install by passing `--no-virt` on the cmdline. This will use Anaconda's directory install feature to handle the install. There are a couple of things to keep in mind when doing this:

1. It will be most reliable when building images for the same release that the host is running. Because Anaconda has expectations about the system it is running under you may encounter strange bugs if you try to build newer or older releases.
2. Make sure selinux is set to permissive or disabled. It won't install correctly with selinux set to enforcing yet.
3. It may totally trash your host. So far I haven't had this happen, but the possibility exists that a bug in Anaconda could result in it operating on real devices. I recommend running it in a virt or on a system that you can afford to lose all data from.

The logs from anaconda will be placed in an ./anaconda/ directory in either the current directory or in the directory used for –logfile

Example cmdline:

```
sudo livemedia-creator --make-iso --no-virt --ks=./fedora-livemedia.ks
```

4.5 AMI Images

Amazon EC2 images can be created by using the –make-ami switch and an appropriate kickstart file. All of the work to customize the image is handled by the kickstart. The example currently included was modified from the cloud-kickstarts version so that it would work with livemedia-creator.

Example cmdline:

```
sudo livemedia-creator --make-ami --iso=/path/to/boot.iso
--ks=./docs/fedora-livemedia-ec2.ks
```

This will produce an ami-root.img file in the working directory.

At this time I have not tested the image with EC2. Feedback would be welcome.

4.6 Appliance Creation

livemedia-creator can now replace appliance-tools by using the –make-appliance switch. This will create the partitioned disk image and an XML file that can be used with virt-image to setup a virtual system.

The XML is generated using the Mako template from /usr/share/lorax/appliance/libvirt.xml You can use a different template by passing `--app-template <template path>`

Documentation on the Mako template system can be found at the Mako site

The name of the final output XML is appliance.xml, this can be changed with `--app-file <file path>`

The following variables are passed to the template:

`disks` A list of disk_info about each disk. Each entry has the following attributes:

- `name` base name of the disk image file
- `format` "raw"
- `checksum_type` "sha256"
- `checksum` sha256 checksum of the disk image

`name` Name of appliance, from –app-name argument

`arch` Architecture

`memory` Memory in KB (from `--ram`)

`vcpus` from `--vcpus`

`networks` list of networks from the kickstart or []

`title` from `--title`

`project` from `--project`

`releasever` from `--releasever`

The created image can be imported into libvirt using:

```
virt-image appliance.xml
```

You can also create qcow2 appliance images using `--qcow2`, for example:

```
sudo livemedia-creator --make-appliance --iso=/path/to/boot.iso --ks=./docs/fedora-minimal.ks \
--qcow2 --app-file=minimal-test.xml --image-name=minimal-test.img
```

4.7 Filesystem Image Creation

livemedia-creator can be used to create un-partitined filesystem images using the `--make-fsimage` option. As of version 21.8 this works with both virt-install and no-virt modes of operation. Previously it was only available with no-virt.

Kickstarts should have a single / partition with no extra mountpoints.

```
livemedia-creator --make-fsimage --iso=/path/to/boot.iso
--ks=./docs/fedora-minimal.ks
```

You can name the output image with `--image-name` and set a label on the filesystem with `--fs-label`

4.8 TAR File Creation

The `--make-tar` command can be used to create a tar of the root filesystem. By default it is compressed using xz, but this can be changed using the `--compression` and `--compress-arg` options. This option works with both virt and no-virt install methods.

As with `--make-fsimage` the kickstart should be limited to a single / partition.

For example:

```
livemedia-creator --make-tar --iso=/path/to/boot.iso --ks=./docs/fedora-minimal.ks \
--image-name=fedora-root.tar.xz
```

4.9 Live Image for PXE Boot

The `--make-pxe-live` command will produce squashfs image containing live root filesystem that can be used for pxe boot. Directory with results will contain the live image, kernel image, initrd image and template of pxe configuration for the images.

4.10 Atomic Live Image for PXE Boot

The `--make-ostree-live` command will produce the same result as `--make-pxe-live` for installations of Atomic Host. Example kickstart for such an installation using Atomic installer iso with local repo included in the image can be found in docs/rhel-atomic-pxe-live.ks.

4.11 Using Mock to Create Images

As of lorax version 22.2 you can use livemedia-creator and anaconda version 22.15 inside of a mock chroot with –make-iso and –make-fsimage. Note that this requires bind mounting the host's /dev/ directory into the mock, which could be dangerous since it includes the host's drives. You can work around this by /dev/loopX nodes before running livemedia-creator. This example does not do that.

On the host system:

1. yum install -y mock
2. Add a user to the mock group to use for running mock. eg. builder
3. Edit the /etc/mock/site-defaults.cfg file to change:

   `config_opts['internal_dev_setup'] = False`

   The loop devices are needed for the installation, so it needs to mount the host's /dev/ inside the mock.

   This is fairly dangerous. I would recommend using a dedicated build host and making sure you have backups just in case something goes wrong and it modifies the host system. You can avoid this if you setup the /dev/loopX device nodes yourself.

4. Create a new /etc/mock/ config file based on the rawhide one, or modify the existing one so that the following options are setup:

```
config_opts['chroot_setup_cmd'] = 'install @buildsys-build anaconda-tui lorax'

# NOTE that this actually needs to be set in site-defaults.cfg
config_opts['internal_dev_setup'] = False

# Mount the relevant host paths inside the mock /dev/
config_opts['plugin_conf']['bind_mount_enable'] = True
config_opts['plugin_conf']['bind_mount_opts']['dirs'].append(('/dev','/dev/'))
config_opts['plugin_conf']['bind_mount_opts']['dirs'].append(('/dev/pts','/dev/pts/'))
config_opts['plugin_conf']['bind_mount_opts']['dirs'].append(('/dev/shm','/dev/shm/'))

# build results go into /home/builder/results/
config_opts['plugin_conf']['bind_mount_opts']['dirs'].append(('/home/builder/results','/results/
```

The following steps are run as the builder user who is a member of the mock group.

5. Make a directory for results matching the bind mount above `mkdir ~/results/`
6. Copy the example kickstarts `cp /usr/share/docs/lorax/*ks .`
7. Make sure tar and dracut-network are in the %packages section and that the `url` points to the correct repo
8. Init the mock `mock -r fedora-rawhide-x86_64 --init`
9. Copy the kickstart inside the mock `mock -r fedora-rawhide-x86_64 --copyin ./fedora-minimal.ks /root/`
10. Make a minimal iso:

```
mock -r fedora-rawhide-x86_64 --chroot -- livemedia-creator --no-virt \
--resultdir=/results/try-1 --logfile=/results/logs/try-1/try-1.log \
--make-iso --ks /root/fedora-minimal.ks
```

Results will be in ./results/try-1 and logs under /results/logs/try-1/ including anaconda logs and livemedia-creator logs. The new iso will be located at ~/results/try-1/images/boot.iso, and the ~/results/try-1/ directory tree will also contain the vmlinuz, initrd, etc.

4.12 OpenStack Image Creation

OpenStack supports partitioned disk images so `--make-disk` can be used to create images for importing into glance, OpenStack's image storage component. You need to have access to an OpenStack provider that allows image uploads,

or setup your own using the instructions from the *RDO Project <https://www.rdoproject.org/Quickstart>*.

The example kickstart, fedora-openstack.ks, is only slightly different than the fedora-minimal.ks one. It adds the cloud-init and cloud-utils-growpart packages. OpenStack supports setting up the image using cloud-init, and cloud-utils-growpart will grow the image to fit the instance's disk size.

Create a qcow2 image using the kickstart like this:

```
sudo livemedia-creator --make-disk --iso=/path/to/boot.iso
--ks=/path/to/fedora-openstack.ks --qcow2
```

Note: On the RHEL7 version of lmc `--qcow2` isn't supported. You can only create a bare partitioned disk image.

Import the resulting disk image into the OpenStack system, either via the web UI, or glance on the cmdline:

```
glance image-create --name "fedora-openstack" --is-public true --disk-format qcow2 \
--container-format bare --file ./fedora-openstack.qcow2
```

If qcow2 wasn't used then `--disk-format` should be set to raw.

4.13 Docker Image Creation

Use lmc to create a tarfile as described in the *TAR File Creation* section, but substitute the fedora-docker.ks example kickstart which removes the requirement for core files and the kernel.

You can then import the tarfile into docker like this (as root):

```
cat /var/tmp/fedora-root.tar.xz | docker import - fedora-root
```

And then run bash inside of it:

```
sudo docker run -i -t fedora-root /bin/bash
```

4.14 Debugging problems

Sometimes an installation will get stuck. When using virt-install the logs will be written to ./virt-install.log and most of the time any problems that happen will be near the end of the file. lmc tries to detect common errors and will cancel the installation when they happen. But not everything can be caught. When creating a new kickstart it is helpful to use the `--vnc vnc` command so that you can monitor the installation as it happens, and if it gets stuck without lmc detecting the problem you can switch to tty1 and examine the system directly.

If it does get stuck the best way to cancel is to use virsh to destroy the domain.

1. Use `sudo virsh list` to show the name of the virt. It will start with LiveOS and contain a UUID.
2. Run `sudo virsh destroy <name>` to destroy the domain.
3. Wait 20 seconds or so for lmc to detect that the domain vanished. It should handle cleanup.

If lmc didn't handle the cleanup for some reason you can do this: 1. `sudo virsh undefine <name>` 2. `sudo umount /tmp/tmpXXXX` to unmount the iso from its mountpoint. 3. `sudo rm -rf /tmp/tmpXXXX` 4. `sudo rm /var/tmp/diskXXXXX` to remove the disk image.

The logs from the virt-install run are stored in virt-install.log, logs from livemedia-creator are in livemedia.log and program.log

You can add `--image-only` to skip the .iso creation and examine the resulting disk image. Or you can pass `--keep-image` to keep it around after the iso has been created.

Cleaning up aborted `--no-virt` installs can sometimes be accomplished by running the `anaconda-cleanup` script. As of Fedora 18 anaconda is multi-threaded and it can sometimes become stuck and refuse to exit. When this happens you can usually clean up by first killing the anaconda process then running `anaconda-cleanup`.

4.15 Hacking

Development on this will take place as part of the lorax project, and on the anaconda-devel-list mailing list, and on github

Feedback, enhancements and bugs are welcome. You can use bugzilla to report bugs against the lorax component.

CHAPTER 5

Product and Updates Images

Lorax now supports creation of product.img and updates.img as part of the build process. This is implemented using the installimg command which will take the contents of a directory and create a compressed archive from it. The x86, ppc, ppc64le and aarch64 templates all look for /usr/share/lorax/product/ and /usr/share/lorax/updates/ directories while creating the final install tree. If there are files in those directories lorax will create images/product.img and/or images/updates.img

These archives are just like an anaconda updates image – their contents are copied over the top of the filesystem at boot time so that you can drop in files to add to or replace anything on the filesystem.

Anaconda has several places that it looks for updates, the one for product.img is in /run/install/product. So for example, to add an installclass to Anaconda you would put your custom class here:

```
/usr/share/lorax/product/run/install/product/pyanaconda/installclasses/custom.py
```

If the packages containing the product/updates files are not included as part of normal dependencies you can add specific packages with the `--installpkgs` command or the installpkgs paramater of `pylorax.treebuilder.RuntimeBuilder`

CHAPTER 6

pylorax

6.1 pylorax package

6.1.1 Submodules

6.1.2 pylorax.base module

6.1.3 pylorax.buildstamp module

6.1.4 pylorax.decorators module

6.1.5 pylorax.discinfo module

6.1.6 pylorax.dnfhelper module

6.1.7 pylorax.executils module

6.1.8 pylorax.imgutils module

6.1.9 pylorax.ltmpl module

6.1.10 pylorax.monitor module

6.1.11 pylorax.mount module

6.1.12 pylorax.output module

6.1.13 pylorax.sysutils module

6.1.14 pylorax.treebuilder module

6.1.15 pylorax.treeinfo module

6.1.16 Module contents

CHAPTER 7

Indices and tables

- genindex
- modindex
- search