Anaconda Documentation

Release 30.25

Anaconda Team

Feb 19, 2019

Contents

Contents:

CHAPTER 1

Introduction to Anaconda

Anaconda is the installation program used by Fedora, Red Hat Enterprise Linux and some other distributions.

During installation, a target computer's hardware is identified and configured and the appropriate file systems for the system's architecture are created. Finally, anaconda allows the user to install the operating system software on the target computer. Anaconda can also upgrade existing installations of earlier versions of the same distribution. After the installation is complete, you can reboot into your installed system and continue doing customization using the initial setup program.

Anaconda is a fairly sophisticated installer. It supports installation from local and remote sources such as CDs and DVDs, images stored on a hard drive, NFS, HTTP, and FTP. Installation can be scripted with kickstart to provide a fully unattended installation that can be duplicated on scores of machines. It can also be run over VNC on headless machines. A variety of advanced storage devices including LVM, RAID, iSCSI, and multipath are supported from the partitioning program. Anaconda provides advanced debugging features such as remote logging, access to the python interactive debugger, and remote saving of exception dumps.

CHAPTER 2

Anaconda Boot Options

Authors Anaconda Developers <anaconda-devel-list@redhat.com> Will Woods <wwoods@redhat.com> Anne Mulhern <amulhern@redhat.com>

These are the boot options that are useful when starting Anaconda. For more information refer to the appropriate Installation Guide for your release and to the Anaconda wiki.

Anaconda bootup is handled by dracut, so most of the kernel arguments handled by dracut are also valid. See dracut.kernel(7) for details on those options.

Throughout this guide, installer-specific options are prefixed with `inst` (e.g. `inst.ks`). Options specified without the `inst` prefix are recognized, but the prefix may be required in a future release.

2.1 Installation Source

Note: An *installable tree* is a directory structure containing installer images, packages, and repodata.[1]

Usually this is either a copy of the DVD media (or loopback-mounted DVD image), or the `<arch>/os/` directory on the Fedora mirrors.

2.1.1 inst.repo

This gives the location of the *Install Source* - that is, the place where the installer can find its images and packages. It can be specified in a few different ways:

`inst.repo=cdrom` Search the system's CDROM drives for installer media. This is the default.

`inst.repo=cdrom:<device>` Look for installer media in the specified disk device.

`inst.repo=hd:<device>:<path>` Mount the given disk partition and install from ISO file on the given path. This installation method requires ISO file, which contains an installable tree.

[1] an installable tree must contain a valid *.treeinfo* file for `inst.repo` or `inst.stage2` to work.

inst.repo=[http,https,ftp]://<host>/<path> Look for an installable tree at the given URL.

inst.repo=nfs:[<options>:]<server>:/<path> Mount the given NFS server and path. Uses NFS version **3** by default.

You can specify what version of the NFS protocol to use by adding `nfsvers=X` to the *options*.

This accepts not just an installable tree directory in the `<path>` element, but you can also specify an `.iso` file. That ISO file is then mounted and used as the installation tree. This is often used for simulating a standard DVD installation using a remote `DVD.iso` image.

Note: Disk devices may be specified with any of the following forms:

Kernel Device Name `/dev/sda1`, `sdb2`

Filesystem Label `LABEL=FLASH`, `LABEL=Fedora`, `CDLABEL=Fedora\x2023\x20x86_64`

Filesystem UUID `UUID=8176c7bf-04ff-403a-a832-9557f94e61db`

Non-alphanumeric characters should be escaped with `\xNN`, where 'NN' is the hexidecimal representation of the character (e.g. `\x20` for the space character (' ').

2.1.2 inst.addrepo

Add additional repository which can be used as another *Installation Source* next to the main repository (see *inst.repo*). This option can be used multiple times during one boot. This can be specified in a few different ways:

inst.addrepo=REPO_NAME,[http,https,ftp]://<host>/<path> Look for the installable tree at the given URL.

inst.addrepo=REPO_NAME,nfs://<server>:/<path> Look for the installable tree at the given nfs path. Note that there is a colon after the host. Anaconda passes everything after "nfs:// " directly to the mount command instead of parsing URLs according to RFC 2224.

inst.addrepo=REPO_NAME,file://<path> Look for the installable tree at the given location in the installation environment. Beware, to be able to use this variant the repo needs to be mounted before Anaconda tries to use it (load available software groups). The main usage for this command is having multiple repositories on one bootable ISO and install both the main repo and additional repositories from this ISO. The path to the additional repositories will be then */run/install/source/REPO_ISO_PATH*. Another solution can be to mount this repo directory in the *%pre* section in the kickstart file. NOTE: The path must be absolute and start with / so the final url starts with *file:///...*.

inst.addrepo=REPO_NAME,hd:<device>:<path> Mount the given <*device*> partition and install from ISO specified by the <*path*>. If the <*path*> is not specified Anaconda will look for the valid installation ISO on the <*device*>. This installation method requires ISO with a valid installable tree. For more detail how to specify <*device*> argument part please see *diskdev*.

The *REPO_NAME* is name of the repository and it is a required part. The name will be used in the installation process. These repositories will be used only during the installation but they **will not** be installed to the installed system.

2.1.3 inst.noverifyssl

Prevents Anaconda from verifying the ssl certificate for all HTTPS connections with an exception of the additional kickstart repos (where –noverifyssl can be set per repo).

2.1.4 inst.proxy

`inst.proxy=PROXY_URL`

Use the given proxy settings when performing an installation from a HTTP/HTTPS/FTP source. The `PROXY_URL` can be specified like this: `[PROTOCOL://][USERNAME[:PASSWORD]@]HOST[:PORT]`.

2.1.5 inst.stage2

This specifies the location to fetch only the installer runtime image; packages will be ignored. Otherwise the same as *inst.repo*.

2.1.6 inst.stage2.all

All locations of type http, https or ftp specified with inst.stage2 will be used sequentially one by one until the image is fetched. Other locations will be ignored.

2.1.7 inst.dd

This specifies the location for driver rpms. May be specified multiple times. Locations may be specified using any of the formats allowed for *inst.repo*.

2.1.8 inst.multilib

This sets dnf's multilib_policy to "all" (as opposed to "best").

2.2 Kickstart

2.2.1 inst.ks

Give the location of a kickstart file to be used to automate the install. Locations may be specified using any of the formats allowed for *inst.repo*.

For any format the `<path>` component defaults to `/ks.cfg` if it is omitted.

For NFS kickstarts, if the `<path>` ends in `/`, `<ip>-kickstart` is added.

If `inst.ks` is used without a value, the installer will look for `nfs:<next_server>:/<filename>`

- `<next_server>` is the DHCP "next-server" option, or the IP of the DHCP server itself
- `<filename>` is the DHCP "filename" option, or `/kickstart/`, and if the filename given ends in `/`, `<ip>-kickstart` is added (as above)

For example:

- DHCP server: `192.168.122.1`
- client address: `192.168.122.100`
- kickstart file: `nfs:192.168.122.1:/kickstart/192.168.122.100-kickstart`

2.2.2 inst.ks.all

All locations of type http, https or ftp specified with inst.ks will be used sequentially one by one until the kickstart file is fetched. Other locations will be ignored.

2.2.3 inst.ks.sendmac

Add headers to outgoing HTTP requests which include the MAC addresses of all network interfaces. The header is of the form:

- `X-RHN-Provisioning-MAC-0: eth0 01:23:45:67:89:ab`

This is helpful when using `inst.ks=http...` to provision systems.

2.2.4 inst.ks.sendsn

Add a header to outgoing HTTP requests which includes the system's serial number.[2]

The header is of the form:

- `X-System-Serial-Number: <serial>`

2.2.5 inst.ksstrict

With this option, all warnings from reading the kickstart file will be treated as errors. They will be printed on the output and the installation will terminate immediately.

By default, the warnings are printed to logs and the installation continues.

2.3 Network Options

Initial network setup is handled by dracut. For detailed information consult the "Network" section of dracut.kernel(7).

The most common dracut network options are covered here, along with some installer-specific options.

2.3.1 ip

Configure one (or more) network interfaces. You can use multiple `ip` arguments to configure multiple interfaces, but if you do you must specify an interface for every `ip=` argument, and you must specify which interface is the primary boot interface with *bootdev*.

Accepts a few different forms; the most common are:

ip=<dhcp|dhcp6|auto6|ibft> Try to bring up every interface using the given autoconf method. Defaults to `ip=dhcp` if network is required by `inst.repo`, `inst.ks`, `inst.updates`, etc.

ip=<interface>:<autoconf> Bring up only one interface using the given autoconf method, e.g. `ip=eth0:dhcp`.

ip=<ip>::<gateway>:<netmask>:<hostname>:<interface>:none Bring up the given interface with a static network config, where:

[2] as read from `/sys/class/dmi/id/product_serial`

<ip> The client IP address. IPv6 addresses may be specified by putting them in square brackets, like so: `[2001:DB8::1]`.

<gateway> The default gateway. IPv6 addresses are accepted here too.

<netmask> The netmask (e.g. `255.255.255.0`) or prefix (e.g. `64`).

<hostname> Hostname for the client machine. This component is optional.

ip=<ip>::<gateway>:<netmask>:<hostname>:<interface>:<autoconf>:<mtu> Bring up the given interface with the given autoconf method, but override the automatically obtained IP/gateway/etc. with the provided values.

Technically all of the items are optional, so if you want to use dhcp but also set a hostname you can use `ip=::::<hostname>::dhcp`.

2.3.2 nameserver

Specify the address of a nameserver to use. May be used multiple times.

2.3.3 bootdev

Specify which interface is the boot device. Required if multiple `ip=` options are used.

2.3.4 ifname

ifname=<interface>:<MAC> Assign the given interface name to the network device with the given MAC. May be used multiple times.

Note: Dracut applies ifname option (which might involve renaming the device with given MAC) in initramfs only if the device is activated in initramfs stage (based on ip= option). If it is not the case, installer still binds the current device name to the MAC by adding HWADDR setting to the ifcfg file of the device.

2.3.5 inst.dhcpclass

Set the DHCP vendor class identifier[3]. Defaults to `anaconda-$(uname -srm)`.

2.3.6 inst.waitfornet

inst.waitfornet=<TIMEOUT_IN_SECONDS> Wait for network connectivity at the beginning of the second stage of installation (after switchroot from early initramfs stage when the installer process is run).

[3] ISC `dhcpd` will see this value as "option vendor-class-identifier".

2.4 Console / Display Options

2.4.1 console

This is a kernel option that specifies what device to use as the primary console. For example, if your console should be on the first serial port, use `console=ttyS0`.

You can use multiple `console=` options; boot messages will be displayed on all consoles, but anaconda will put its display on the last console listed.

Implies *inst.text*.

2.4.2 inst.lang

Set the language to be used during installation. The language specified must be valid for the `lang` kickstart command.

2.4.3 inst.singlelang

Install in single language mode - no interactive options for installation language and language support configuration will be available. If a language has been specified via the *inst.lang* boot option or the *lang* kickstart command it will be used. If no language is specified Anaconda will default to en_US.UTF-8.

2.4.4 inst.geoloc

Configure geolocation usage in Anaconda. Geolocation is used to pre-set language and time zone.

`inst.geoloc=0` Disables geolocation.

`inst.geoloc=provider_fedora_geoip` Use the Fedora GeoIP API (default).

`inst.geoloc=provider_hostip` Use the Hostip.info GeoIP API.

2.4.5 inst.geoloc-use-with-ks

Enable geolocation even during a kickstart installation (both partial and fully automatic). Otherwise geolocation is only enabled during a fully interactive installation.

2.4.6 inst.keymap

Set the keyboard layout to use. The layout specified must be valid for use with the `keyboard` kickstart command.

2.4.7 inst.cmdline

Run the installer in command-line mode. This mode does not allow any interaction; all options must be specified in a kickstart file or on the command line.

2.4.8 inst.graphical

Run the installer in graphical mode. This is the default.

2.4.9 inst.text

Run the installer using a limited text-based UI. Unless you're using a kickstart file this probably isn't a good idea; you should use VNC instead.

2.4.10 inst.noninteractive

Run the installer in a non-interactive mode. This mode does not allow any user interaction and can be used with graphical or text mode. With text mode it behaves the same as the `inst.cmdline` mode.

2.4.11 inst.resolution

Specify screen size for the installer. Use format nxm, where n is the number of horizontal pixels, m the number of vertical pixels. The lowest supported resolution is 800x600.

2.4.12 inst.vnc

Run the installer GUI in a VNC session. You will need a VNC client application to interact with the installer. VNC sharing is enabled, so multiple clients may connect.

A system installed with VNC will start in text mode (runlevel 3).

2.4.13 inst.vncpassword

Set a password on the VNC server used by the installer.

2.4.14 inst.vncconnect

`inst.vncconnect=<host>[:<port>]` Once the install starts, connect to a listening VNC client at the given host. Default port is 5900.

Use with `vncviewer -listen`.

2.4.15 inst.headless

Specify that the machine being installed onto doesn't have any display hardware, and that anaconda shouldn't bother looking for it.

2.4.16 inst.xdriver

Specify the X driver that should be used during installation and on the installed system.

2.4.17 inst.usefbx

Use the framebuffer X driver (`fbdev`) rather than a hardware-specific driver.

Equivalent to `inst.xdriver=fbdev`.

2.4.18 inst.xtimeout

Specify the timeout in seconds for starting X server.

2.4.19 inst.sshd

Start up `sshd` during system installation. You can then ssh in while the installation progresses to debug or monitor its progress.

> **Caution:** The `root` account has no password by default. You can set one using the `sshpw` kickstart command.

2.4.20 inst.decorated

Run GUI installer in a decorated window. By default, the window is not decorated, so it doesn't have a title bar, resize controls, etc.

2.5 Debugging and Troubleshooting

2.5.1 inst.rescue

Run the rescue environment. This is useful for trying to diagnose and fix broken systems.

2.5.2 inst.updates

Give the location of an `updates.img` to be applied to the installer runtime. Locations may be specified using any of the formats allowed for `inst.repo`.

For any format the `<path>` component defaults to `/updates.img` if it is omitted.

2.5.3 inst.nokill

A debugging option that prevents anaconda from and rebooting when a fatal error occurs or at the end of the installation process.

2.5.4 inst.loglevel

`inst.loglevel=<debug|info|warning|error|critical>` Set the minimum level required for messages to be logged on a terminal (log files always contain messages of all levels). The default value is `info`.

2.5.5 inst.noshell

Do not put a shell on tty2 during install.

2.5.6 inst.notmux

Do not use tmux during install. This allows for output to get generated without terminal control characters and is really meant for non-interactive uses.

2.5.7 inst.syslog

inst.syslog=<host>[:<port>] Once installation is running, send log messages to the syslog process on the given host. The default port is 514 (UDP).

Requires the remote syslog process to accept incoming connections.

2.5.8 inst.virtiolog

Forward logs through the named virtio port (a character device at `/dev/virtio-ports/<name>`).

If not provided, a port named `org.fedoraproject.anaconda.log.0` will be used by default, if found.

See the Anaconda wiki logging page for more info on setting up logging via virtio.

2.5.9 inst.zram

Forces/disables (on/off) usage of zRAM swap for the installation process.

2.6 Boot loader options

2.6.1 extlinux

Use extlinux as the bootloader. Note that there's no attempt to validate that this will work for your platform or anything; it assumes that if you ask for it, you want to try.

2.6.2 leavebootorder

Boot the drives in their existing order, to override the default of booting into the newly installed drive on Power Systems servers and EFI systems. This is useful for systems that, for example, should network boot first before falling back to a local boot.

2.7 Storage options

2.7.1 inst.nodmraid

Disable support for dmraid.

> **Warning:** This option is never a good idea! If you have a disk that is erroneously identified as part of a firmware RAID array, that means it has some stale RAID metadata on it which must be removed using an appropriate tool (dmraid and/or wipefs).

2.7.2 inst.nompath

Disable support for multipath devices. This is for systems on which a false-positive is encountered which erroneously identifies a normal block device as a multipath device. There is no other reason to use this option.

> **Warning:** Not for use with actual multipath hardware! Using this to attempt to install to a single path of a multipath is ill-advised, and not supported.

2.7.3 inst.gpt

Prefer creation of GPT disklabels.

2.8 Other options

2.8.1 inst.selinux

Enable SELinux usage in the installed system (default). Note that when used as a boot option, "selinux" and "inst.selinux" are not the same. The "selinux" option is picked up by both the kernel and Anaconda, but "inst.selinux" is processed only by Anaconda. So when "selinux=0" is used, SELinux will be disabled both in the installation environment and in the installed system, but when "inst.selinux=0" is used SELinux will only be disabled in the installed system. Also note that while SELinux is running in the installation environment by default, it is running in permissive mode so disabling it there does not make much sense.

2.8.2 inst.nosave

Controls what installation results should not be saved to the installed system, valid values are: "input_ks", "output_ks", "all_ks", "logs" and "all".

`input_ks` Disables saving of the input kickstart (if any).

`output_ks` Disables saving of the output kickstart generated by Anaconda.

`all_ks` Disables saving of both input and output kickstarts.

`logs` Disables saving of all installation logs.

`all` Disables saving of all kickstarts and all logs.

Multiple values can be combined as a comma separated list, for example: `input_ks,logs`

Note: The nosave option is meant for excluding files from the installed system that *can't* be removed by a kickstart %post script, such as logs and input/output kickstarts.

2.8.3 inst.nonibftiscsiboot

Allows to place boot loader on iSCSI devices which were not configured in iBFT.

2.8.4 Product options

Use the options `inst.product` and `inst.variant` to specify a product. The installer will be customized based on configuration files from `/etc/anaconda/product.d` that are specific for this product.

inst.product

Set the name of a product.

For example: `inst.product=Fedora`

inst.variant

Set the name of a variant.

For example: `inst.variant=Workstation`

2.8.5 Third-party options

Since Fedora 19 the Anaconda installer supports third-party extensions called *addons*. The *addons* can support their own set of boot options which should be documented in their documentation or submitted here.

inst.kdump

`inst.kdump_addon=on/off`

Enable kdump anaconda addon to setup the kdump service.

2.9 Deprecated Options

These options should still be accepted by the installer, but they are deprecated and may be removed soon.

2.9.1 method

This is an alias for *inst.repo*.

2.9.2 repo=nfsiso:...

The difference between an installable tree and a dir with an `.iso` file is autodetected, so this is the same as `inst.repo=nfs:...`

2.9.3 dns

Use *nameserver* instead. Note that `nameserver` does not accept comma-separated lists; use multiple `nameserver` options instead.

2.9.4 netmask, gateway, hostname

These can be provided as part of the *ip* option.

2.9.5 ip=bootif

A PXE-supplied BOOTIF option will be used automatically, so there's no need

2.9.6 ksdevice

Not present The first device with a usable link is used

`ksdevice=link` Ignored (this is the same as the default behavior)

`ksdevice=bootif` Ignored (this is the default if `BOOTIF=` is present)

`ksdevice=ibft` Replaced with `ip=ibft`. See *ip*

`ksdevice=<MAC>` Replaced with `BOOTIF=${MAC/:/-}`

`ksdevice=<DEV>` Replaced with *bootdev*

2.10 Removed Options

These options are obsolete and have been removed.

2.10.1 askmethod, asknetwork

Anaconda's initramfs is now is completely non-interactive, so these have been removed.

Instead, use *inst.repo* or specify appropriate *Network Options*.

2.10.2 blacklist, nofirewire

`modprobe` handles blacklisting kernel modules on its own; try `modprobe.blacklist=<mod1>,<mod2>...`

You can blacklist the firewire module with `modprobe.blacklist=firewire_ohci`.

2.10.3 serial

This option was never intended for public use; it was supposed to be used to force anaconda to use `/dev/ttyS0` as its console when testing it on a live machine.

Use `console=ttyS0` or similar instead. See *console* for details.

2.10.4 updates

Use *inst.updates* instead.

2.10.5 essid, wepkey, wpakey

Dracut doesn't support wireless networking, so these don't do anything.

2.10.6 ethtool

Who needs to force half-duplex 10-base-T anymore?

2.10.7 gdb

This was used to debug `loader`, so it has been removed. There are plenty of options for debugging dracut-based initramfs - see the dracut "Troubleshooting" guide.

2.10.8 inst.mediacheck

Use the dracut option rd.live.check instead.

2.10.9 ks=floppy

We no longer support floppy drives. Try `inst.ks=hd:<device>` instead.

2.10.10 display

For remote display of the UI, use *inst.vnc*.

2.10.11 utf8

All this option actually did was set `TERM=vt100`. The default `TERM` setting works fine these days, so this was no longer necessary.

2.10.12 noipv6

ipv6 is built into the kernel and can't be removed by anaconda.

You can disable ipv6 with `ipv6.disable=1`. This setting will be carried onto the installed system.

2.10.13 upgradeany

Anaconda doesn't handle upgrades anymore.

2.10.14 inst.repo=hd:<device>:<path> for installable tree

Anaconda can't use this option with installable tree but only with an ISO file.

CHAPTER 3

Anaconda Kickstart Documentation

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

Anaconda uses kickstart to automate installation and as a data store for the user interface. It also extends the kickstart commands documented here by adding a new kickstart section named `%anaconda` where commands to control the behavior of Anaconda will be defined.

3.1 pwpolicy

program: pwpolicy <name> [--minlen=LENGTH] [--minquality=QUALITY] [--strict|notstrict] [--

Set the policy to use for the named password entry.

name Name of the password entry, currently supported values are: root, user and luks

--minlen (6) Minimum password length. This is passed on to libpwquality.

--minquality (1) Minimum libpwquality to consider good. When using `--strict` it will not allow passwords with a quality lower than this.

--strict Strict password enforcement. Passwords not meeting the `--minquality` level will not be allowed.

--notstrict (DEFAULT) Passwords not meeting the `--minquality` level will be allowed after Done is clicked twice.

--emptyok (DEFAULT) Allow empty password.

--notempty Don't allow an empty password

--changesok Allow UI to be used to change the password/user when it has already been set in the kickstart.

--nochanges (DEFAULT) Do not allow UI to be used to change the password/user if it has been set in the kickstart.

The defaults for interactive installations are set in the `/usr/share/anaconda/interactive-defaults.ks` file provided by Anaconda. If a product, such as Fedora Workstation, wishes to override them then a `product.img` needs to be created with a new version of the file included.

When using a kickstart the defaults can be overridden by placing an `%anaconda` section into the kickstart, like this:

```
%anaconda
pwpolicy root --minlen=10 --minquality=60 --strict --notempty --nochanges
%end
```

Note: The commit message for pwpolicy included some incorrect examples.

3.2 installclass

`installclass --name=<name>`

Require the specified install class to be used for the installation. Otherwise, the best available install class will be used.

`--name=`

Name of the required install class.

Removed since Fedora 30.

Note: You can use the boot options `inst.product` and `inst.variant`.

CHAPTER 4

Contribution guidelines

4.1 How to Contribute to the Anaconda Installer (the short version)

1. I want to contribute to the upstream Anaconda Installer (used in Fedora):
   - open a pull request for the `<next Fedora number>-devel` branch (f25-devel, etc.)
   - check the *Commit Messages* section below for how to format your commit messages
2. I want to contribute to the RHEL Anaconda installer:
   - open a pull request for the `<RHEL number>-branch` branch (rhel7-branch, etc.)
   - check the *Commits for RHEL Branches* section below for how to format your commit messages

If you want to contribute a change to both the upstream and RHEL Anaconda then follow both a) and b) separately.

4.2 Anaconda Installer Branching Policy (the long version)

The basic premise is that there are the following branches:

- master
- <next fedora number>-release
- <next fedora number>-devel

`Master` branch never waits for any release-related processes to take place and is used for Fedora Rawhide Anaconda builds.

Concerning current RHEL branches, they are too divergent to integrate into this scheme. Thus, commits are merged onto, and builds are done on the RHEL branches. In this case, two pull requests will very likely be needed:

- one for the `rhel<number>-branch`
- one for the `master` or `<fedora number>-devel` branch (if the change is not RHEL only)

4.3 Releases

For specific Fedora version, the release process is as follows:

- `<next Fedora number>-devel` is merged onto `<next Fedora number>-release`
- a release commit is made (which bumps version in spec file) & tagged

Concerning Fedora Rawhide, the release process is slightly different:

- a release commit is made (which bumps version in spec file) & tagged

Concerning the `<next Fedora number>` branches (which could also be called `next stable release` if we wanted to decouple our versioning from Fedora in the future):

- work which goes into the next Fedora goes to `<next Fedora number>-devel`, which is periodically merged back to `master`
- this way we can easily see what was developed in which Fedora timeframe and possibly due to given Fedora testing phase feedback (bugfixes, etc.)
- stuff we *don't* want to go to the next Fedora (too cutting edge, etc.) goes only to `master` branch
- commits specific to a given Fedora release (temporary fixes, etc.) go only to the `<next Fedora number>-release` branch
- the `<next Fedora number>-release` branch also contains release commits

4.4 Example for the F25 cycle

- master
- f25-devel
- f25-release

This would continue until F25 is released, after which we:

- drop the f25-devel branch
- keep f25-release as an inactive record of the f25 cycle
- branch f26-devel and f26-release from the master branch

This will result in the following branches for the F26 cycle:

- master
- f26-devel
- f26-release

4.5 Guidelines for Commits

4.5.1 Commit Messages

The first line should be a succinct description of what the commit does. If your commit is fixing a bug in Red Hat's bugzilla instance, you should add '' (#123456)'' to the end of the first line of the commit message. The next line should be blank, followed (optionally) by a more in-depth description of your changes. Here's an example:

Stop kickstart when space check fails

Text mode kickstart behavior was inconsistent, it would allow an installation to continue even though the space check failed. Every other install method stops, letting the user add more space before continuing.

4.5.2 Commits for RHEL Branches

If you are submitting a patch for any rhel-branch, the last line of your commit must identify the bugzilla bug id it fixes, using the `Resolves` or `Related` keyword, e.g.: `Resolves:  rhbz#111111`

or

`Related:  rhbz#1234567`

Use `Resolves` if the patch fixes the core issue which caused the bug. Use `Related` if the patch fixes an ancillary issue that is related to, but might not actually fix the bug.

4.5.3 Pull Request Review

Please note that there is a minimum review period of 24 hours for any patch. The purpose of this rule is to ensure that all interested parties have an opportunity to review every patch. When posting a patch before or after a holiday break it is important to extend this period as appropriate.

All subsequent changes made to patches must be force-pushed to the PR branch before merging it into the main branch.

4.6 Merging examples

4.6.1 Merging the Fedora `devel` branch back to the `master` branch

(Fedora 25 is used as an example, don't forget to use appropriate Fedora version.)

Checkout and pull the master branch:

```
git checkout master git pull
```

Merge the Fedora devel branch to the master branch:

```
git merge --no-ff f25-devel
```

Push the merge to the remote:

```
git push origin master
```

4.6.2 Merging a GitHub pull request

(Fedora 25 is used as an example, don't forget to use appropriate Fedora version.)

Press the green *Merge pull request* button on the pull request page.

If the pull request has been opened for:

- master
- f25-release
- rhel7-branch

Then you are done.

If the pull request has been opened for the `f25-devel` branch, then you also need to merge the `f25-devel` branch back to `master` once you merge your pull request (see "Merging the Fedora devel branch back to the master branch" above).

4.6.3 Merging a topic branch manually

(Fedora 25 is used as an example, don't forget to use appropriate Fedora version.)

Let's say that there is a topic branch called "fix_foo_with_bar" that should be merged to a given Anaconda non-topic branch.

Checkout the given target branch, pull it and merge your topic branch into it:

```
git checkout <target branch> git pull git merge --no-ff fix_foo_with_bar
```

Then push the merge to the remote:

```
git push origin <target branch>
```

If the <target branch> was one of:

- master
- f25-release
- rhel7-branch

Then you are done.

If the pull request has been opened for the `f25-devel` branch, then you also need to merge the `f25-devel` branch back to `master` once you merge your pull request (see "Merging the Fedora devel branch back to the master branch" above).

CHAPTER 5

Rules for commit messages

git commit messages for anaconda should follow a consistent format. The following are rules to follow when committing a change to the git repo:

1. The first line of the commit message should be a short summary of the change in the patch. We also place (#BUGNUMBER) at the end of this line to indicate the bugzilla.redhat.com bug number addressed in this patch. The bug number is optional since there may be no bug number, but if you have one you are addressing, please include it on the summary line. Lastly, the summary lines need to be short. Ideally less than 75 characters, but certainly not longer than 80.

   Here are acceptable first lines for git commit messages:

   ```
   Check partition and filesystem type on upgrade (#123456)
   Fix bootloader configuration setup on ppc64 (#987654)
   Introduce a new screen for setting your preferred email client
   ```

   The last one would be a new feature that we didn't have a bug number for.

2. The main body of the commit message should begin TWO LINES below the summary line you just entered (that is, there needs to be a blank line between the one line summary and the start of the long commit message). Please document the change and explain the patch here. Use multiple paragraphs and keep the lines < 75 chars. DO NOT indent these lines. Everything in the git commit message should be left justified. PLEASE wrap long lines. If you don't, the 'git log' output ends up looking stupid on 80 column terminals.

3. For RHEL bugs, all commits need to reference a bug number. You may follow one of two formats for specifying the bug number in a RHEL commit.

   (a) Put the bug number on the summary line in (#BUGNUMBER) format. Bugs listed this way are treated as 'Resolves' patches in the RHEL universe.

   (b) If you have a patch that is Related to or Conflicts with another bug, you may add those lines to the end of the long commit message in this format:

   ```
   Related: rhbz#BUGNUMBER
   Conflicts: rhbz#BUGNUMBER
   Resolves: rhbz#BUGNUMBER
   ```

These entries should come at the end of the long commit message and must follow the format above. You may have as many of these lines as appropriate for the patch.

(c) Patches that are 'Resolves' patches have two methods to specify the bug numbers, but Related and Conflicts can only be listed in the long commit message.

On RHEL branches, the 'bumpver' process will verify that each patch for the release references a RHEL bug number. The scripts/makebumpver script will extract the bug numbers from RHEL branch commits and do two things. First, it verifies that the bug referenced is a RHEL bug and in correct states. Second, it adds the appropriate Resolves/Related/Conflicts line to the RPM spec file changelog.

It is recommended to use the pre-push hook checking commit messages for RHEL bug numbers and checking the referenced bugs for all the necessary acks. To make it work, just copy the scripts/githooks/pre-push and scripts/githooks/check_commit_msg.sh scripts to the .git/hooks/ directory.

CHAPTER 6

Rawhide release & package build

This guide describes how one create a new Anaconda release, from release commit to a new build in Koji. While aimed primarily on core Anaconda developers and package maintainers doing official release and package build, it could very well be useful for other use cases, such as for scratch builds or creation of custom Anaconda packages. In that case just ignore all section that require you to be an Anaconda maintainer or developer. :)

0. prerequisites

- you need an up to date anaconda source code checkout
- you need to have commit access to the anaconda repository (so that you can push release commits)
- you need to have write access to the corresponding Fedora Zanata project so that you can push .pot file updates
- you need to have the `rpmbuild` or `mock` and `fedpkg` tools installed
- you need to have the Fedora Kerberos based authentication setup
- you need to have committer access to the anaconda package on Fedora distgit

6.1 Using `rpmbuild` path

This is more standard and stable way to make Anaconda release. The drawback of this method is you need to have everything installed locally so you are required to install a lot of dependencies to your system. For the mock environment way see mock path below.

1. do any changes that are needed to anaconda.spec.in

```
vim anaconda.spec.in
```

2. do a release commit

```
./scripts/makebumpver -c --skip-zanata
```

3. check the commit and tag are correct
4. push the master branch to the remote

```
git push master --tags
```

5. configure anaconda

```
make clean
./autogen
./configure
```

6. create tarball

```
make release
```

7. copy tarball to SOURCES

```
cp anaconda-*.tar.bz2 ~/rpmbuild/SOURCES/
```

8. create SRPM

```
rpmbuild -bs --nodeps anaconda.spec
```

9. if you don't have it yet checkout Anaconda from Fedora distgit, switch to the master branch & make sure it's up to date

```
cd <some folder>
fedpkg clone anaconda
cd anaconda
fedpkg switch-branch master
git pull
```

10. switch to Fedora distgit folder and import the SRPM

```
fedpkg import ~/rpmbuild/SRPMS/anaconda-<version>.src.rpm
```

11. this will stage a commit, check it's content and commit

- Do not forget to replace the `<new-version>` with correct version!!

```
fedpkg commit --with-changelog --message "New version <new-version>"
```

12. push the update

```
fedpkg push
```

13. start the build

```
fedpkg build
```

6.2 Using mock path solution

This is an alternative to the `rpmbuild` tutorial above using `mock` container environment. This way has the benefit that you don't need to install all Anaconda dependencies to your system. To be able to use the mock without root privileges you should be member of a `mock` group.

1. allow network access in a mock environment

```
vim /etc/mock/site-defaults.cfg
```

find line which contains

```
config_opts['rpmbuild_networking']
```

uncomment it and set it to True instead

1. do any changes that are needed to anaconda.spec.in

```
vim anaconda.spec.in
```

2. do a release commit

```
./scripts/makebumpver -c --skip-zanata
```

3. check the commit and tag are correct
4. push the master branch to the remote

```
git push origin master --tags
```

5. prepare mock environment

```
./scripts/testing/setup-mock-test-env.py --init -c -p --release fedora-rawhide-x86_64
```

6. connect to the prepared mock environment

```
mock -r fedora-rawhide-x86_64 --chroot -- "cd anaconda && make clean; ./autogen.sh &&␣
↪./configure && make release"
```

7. copy tarball to SOURCES from a mock

```
mock -r fedora-rawhide-x86_64 --copyout "/anaconda/anaconda-*.tar.bz2" .
mock -r fedora-rawhide-x86_64 --copyout "/anaconda/anaconda.spec" .
```

8. create SRPM

```
mock -r fedora-rawhide-x86_64 --buildsrpm --spec ./anaconda.spec --sources ./anaconda-
↪*.tar.bz2 --resultdir /tmp/anaconda-srpm/
cp /tmp/anaconda-srpm/anaconda-*.src.rpm .
```

9. if you don't have it yet checkout Anaconda from Fedora distgit, switch to the master branch & make sure it's up to date

```
cd <some folder>
fedpkg clone anaconda
cd anaconda
fedpkg switch-branch master
git pull
```

10. switch to Fedora distgit folder and import the SRPM; to make this work you have to be authenticated in FAS by a kerberos ticket

```
fedpkg import <anaconda directory>/anaconda-<version>.src.rpm
```

11. this will stage a commit, check it's content and commit

- Do not forget to replace the <new-version> with correct version!!

```
fedpkg commit --with-changelog --message "New version <new-version>"
```

12. push the update

```
fedpkg push
```

13. start the build

```
fedpkg build
```

CHAPTER 7

Upcomming Fedora release & package build

Creating and anaconda release and build for an upcoming Fedora release is pretty similar to a Rawhide build with a few key differences:

- the branches are named differently
- you need to create a Bodhi update so that the build actually reaches the stable package repository

So let's enumerate the steps that doe something differently in more detail (we use Fedora 28 in the CLI examples):

1. merge f<fedora version>-devel to f<fedora version>-release

```
git checkout f28-devel
git pull
git checkout f28-release
git pull
git merge --no-ff f28-devel
```

5. push the f<fedora version>-release branch to the remote

```
git push f28-release --tags
```

9. if you don't have it yet checkout Anaconda from Fedora distgit, switch to the f<fedora version> branch & make sure it's up to date

```
cd <some folder>
fedpkg clone anaconda
fedpkg switch-branch f28
git pull
```

As this is a build for a upcoming Fedora release we need to also submit a Bodhi update:

14. create a Bodhi update from the command line (from the distgit folder)

- you can only do this once the Koji build finishes successfully
- it's also possible to create the update from the Bodhi web UI

```
fedpkg --update
```

Next an update template should open in your editor of choice - fill it out, save it & quite the editor. A link to the update should be returned and you should also start getting regular spam from Bodhi when anything remotely interesting happens with the update. :)

CHAPTER 8

Releasing during a Fedora code freeze

There are two generally multi-week phases during which the upcoming Fedora release development a temporary code freeze:

- the Beta freeze
- the Final freeze

During these periods of time only accepted freeze exceptions and blocker fixes are allowed to reach the stable repository.

To reconcile the freeze concept with the idea that the -devel branch should should be always open for development and that it should be always possible to merge the -devel branch to the -release branch (even just for CI requirements) we have decided temporarily use downstream patches for package builds during the freeze.

That way we avoid freeze induced cherry picks that might break merges in the future and can easily drop the patches once the freeze is over and resume the normal merge-devel-to-release workflow.

8.1 How it should work

Once Fedora enters a freeze:

- all freeze exceptions and blocker fixes are cherry picked into patch files
- patch files are added to distgit only as downstream patches

Once Fedora exits the freeze:

- drop the downstream patches and do merge based releases as before

CHAPTER 9

Branching for the next Fedora release

Anaconda uses separate branches for each Fedora release to make parallel Anaconda development for Rawhide and next Fedora possible. The branches are named like this:

- f<number>-devel
- f<number>-release

The `-devel` branch is where code changes go and it is periodically merged to the master branch. The `-release` branch contains release commits and any Fedora version specific hotfixes.

9.1 How to branch Anaconda

Create the `-devel` branch:

```
git checkout master
git pull
git checkout -b f<version>-devel
```

Create the `-release` branch:

```
git checkout master
git pull
git checkout -b f<version>-release
```

Push the branches to the origin (`-u` makes sure to setup tracking) :

```
git push -u origin f<version>-devel
git push -u origin f<version>-release
```

9.2 How to create translation branch for next Fedora in Zanata

The Fedora project uses the fedora.zanata.org translation system, so for each Fedora release we also need to create a new translation branch there.

To do this you need to have:

- a FAS account
- be in the admin group of the Anaconda project on Zanata

1. Go to the Anaconda project on the Fedora Zanata instance: https://fedora.zanata.org/project/view/anaconda
2. Make sure you are logged in.
3. Click on the small arrow next to the `master` branch and select `Copy to new version`
4. On the new page `version id` should be `f<version>` and make sure `Copy from previous version` is ticked
5. Wait till the new branch is created.

9.3 How to bump Rawhide Anaconda version

- major version becomes major version `+1`
- minor version is set to 1

For example, for the F27 branching:

- at the time of branching the Rawhide version was `27.20`
- after the bump the version is `28.1`

Do the major version bump and verify that the output looks correct:

```
./scripts/makebumpver --skip-zanata -c --bump-major-version
```

If everything looks fine (changelog, new major version & the tag) push the changes to the origin:

```
git push origin master --tags
```

Then continue with the normal Rawhide Anaconda build process.

9.4 How to add release version for next Fedora

The current practise is to keep the Rawhide major & minor version from which the given Anaconda was branched as-is and add a third version number (the release number in the NVR nomenclature) and bump that when releasing a new Anaconda for the upcoming Fedora release.

For example, for the F27 branching:

- the last Rawhide Anaconda release was 27.20
- so the first F27 Anaconda release will be 27.20.1, the next 27.20.2 and so on

First checkout the `f<version>-release` branch and merge `f<version>-devel` into it:

```
git checkout f<version>-release
git merge --no-ff f<version>-devel
```

Then correct pykickstart version for the new Fedora release by changing all occurrences of the DEVEL constant imported from pykickstart for the F<version> constant, for example:

```
from pykickstart.version import DEVEL as VERSION
```

to

```
from pykickstart.version import F29 as VERSION
```

Pykickstart generally does not do per Fedora version branches, so this needs to be done in the Fedora version specific branch on Anaconda side.

Commit the result. The commit will become one of the few exclusive release branch commits, as we can't let it be merged back to master via the devel branch for obvious reasons.

Next add the third (release) version number:

```
./scripts/makebumpver --skip-zanata -c --add-version-number
```

If everything looks fine (changelog, the version number & tag) push the changes to the origin:

```
git push origin f<version>-release --tags
```

Then continue with the normal Upcoming Fedora Anaconda build process.

CHAPTER 10

Brief description of DriverDisc version 3

For a new major release we decided to introduce a new version of DriverDisc feature to ensure the smoothest vendor and user experience possible. We had many reasons for it:

- the old DD didn't support multiple architectures
- the old DD wasn't particulary easy to create
- the old DD had two copys of modules, one for anaconda and one for instalation
- the modules in old DD weren't checked for kernel version

We also changed the feature internal code to enable some functionality that was missing from the old version. More about it below.

10.1 Devices which can contain DDs

The best place to save your DriverDisc to is USB flash device. We also support IDE and SATA block devices with or without partitions, DriverDisc image stored on block device, initrd overlay (see documentation below) and for special cases even network retrieval of DriverDisc image.

10.2 What can be updated using DDs?

All drivers for block devices, which weren't used for retrieving DriverDiscs, the same applies also for network drivers eg. you cannot upgrade network driver for device, which was used prior the DriverDisc extraction.

RPMs for installation. If the DriverDisc repo contains newer package, than the official repository, the newer package will get used.

We also plan to support anaconda's updates.img placement on the DriverDisc to update stage2 behaviour of anaconda.

10.3 Selecting DD manually

Use the 'inst.dd' kernel command line option to trigger DD mode. If no argument is specified, the UI will prompt for the location of the driver rpm. Otherwise, the rpm will be fetched from the specified location.

Please consult the appropriate Installer Guide for further information.

10.4 Automatic DriverDisc detection

Anaconda automatically looks for driverdiscs during startup.

The DriverDisc has to be on partition or filesystem which has been labeled with 'OEMDRV' label.

10.5 DDv3 structure

The new DriverDisc format uses simple layout which can be created on top of any anaconda's supported filesystem (vfat, squashfs, ext2 and ext3).

```
/
|rhdd3   - DD marker, contains the DD's description string
/rpms
  |  /i386 - contains RPMs for this arch and acts as package repo
  |  /i586
  |  /x86_64
  |  /ppc
  |  /...  - any other architecture the DD provides drivers for
```

There is a special requirement for the RPMs used to update drivers. Anaconda picks up only RPMs which provide "kernel-modules = <running kernel version>".

10.6 Initrd overlay driverdisc image

We have designed another possible way of providing updates in network boot environments. It is possible to update all modules this way, so if special storage module (which gets used early) needs to be updated, this is the preffered way.

This kind of driverdisc image is applied over the standard initrd and so has to respect some rules.

- All updated modules belong to /lib/modules/<kernel version>/.. according to their usual location
- All new modules belong to /lib/modules/<kernel version>/updates
- All new firmware files belong to /lib/firmware
- The rpm repo with updated packages belongs to /tmp/DD-initrd/
- The (empty) trigger file /.rundepmod must be present

10.7 Firmware and module update

The firmware files together with all .ko files from the RPMs are exploded to special module location, which has preference over built-in Anaconda modules.

Anaconda doesn't use built-in modules (except some storage modules needed for the DD to function properly) during the DriverDisc mode, so even in case when you are updating some modules with second (or later) DriverDisc, the updated modules will be loaded. There is one exception though, if your module depends on a module which is only present in built-in module directory, that built-in module gets also loaded.

10.8 Package installation

It is also possible to include arbitrary packages on the DriverDisc media and mark them for installation. You just have to include the package name in the package repo for correct architecture and mark it as mandatory.

10.9 Summary

This new DriverDisc format should simplify the DD creation and usage a lot. We will gladly hear any comments as this is partially still work in progress.

CHAPTER 11

iSCSI and Anaconda

Authors Ales Kozumplik <akozumpl@redhat.com>

11.1 Introduction

iSCSI device is a SCSI device connected to your computer via a TCP/IP network. The communication can be handled either in hardware or in software, or as a hybrid — part software, part hardware.

The terminology:

- 'initiator', the client in the iscsi connection. The computer we are running Anaconda on is typically an initiator.
- 'target', the storage device behind the Network. This is where the data is physically stored and read from. You can turn any Fedora/RHEL machine to a target (or several) via scsi-target-utils.
- 'HBA' or Host Bus Adapter. A device (PCI card typically) you connect to a computer. It acts as a NIC and if you configure it properly it transparently connects to the target when started and all you can see is a block device on your system.
- 'software initiator' is what you end up with if you emulate most of what HBA is doing and just use a regular NIC for the iscsi communication. The modern Linux kernel has a software initiator. To use it, you need the Open-ISCSI software stack [1, 2] installed. It is known as iscsi-initiator-utils in Fedora/RHEL.
- 'partial offload card'. Similar to HBA but needs some support from kernel and iscsi-initiator-utils. The least pleasant to work with, particularly because there is no standardized amount of the manual setting that needs to be done (some connect to the target just like HBAs, some need you to bring their NIC part up manually etc.). Partial offload cards exist to get better performing I/O with less processor load than with software initiator.
- 'iBFT' as in 'Iscsi Boot Firmware Table'. A table in the card's bios that contains its network and target settings. This allows the card to configure itself, connect to a target and boot from it before any operating system or a bootloader has the chance. We can also read this information from /sys/firmware/ibft after the system starts and then use it to bring the card up (again) in Linux.
- 'CHAP' is the authentication used for iSCSI connections. The authentication can happen during target discovery or target login or both. It can happen in both directions too: the initiator authenticates itself to the target and the target is sometimes required to authenticate itself to the initiator.

11.2 What is expected from Anaconda

We are expected to:

- use an HBA like an ordinary disk. It is usually smart enough to bring itself up during boot, connect to the target and just act as an ordinary disk.
- allow creating new software initiator connections in the UI, both IPv4 and IPv6.
- facilitate bringing up iBFT connections for partial offload cards.
- install the root and/or /boot filesystems on any iSCSI initiator known to us
- remember to install dracut-network if we are booting from an iSCSI initiator that requires iscsi-initiator-utils in the ramdisk (most of them do)
- boot from an iSCSI initiator using dracut, this requires generating an appropriate set of kernel boot arguments for it [3].

11.3 How Anaconda handles iscsi

iSCSI comes into play several times while Anaconda does its thing:

In loader, when deciding what NIC we should setup, we check if we have iBFT information from one of the cards. If we do we set that card up with what we found in the table, it usually boils down to an IPv4 static or IPv4 DHCP-obtained address. [4][5]

Next, after the main UI startup during filtering (or storage scan, whatever comes first) we startup the iscsi support code in Anaconda [6]. This currently involves: - manually modprobing related kernel modules - starting the iscsiuio daemon (required by some partial offload cards) - most importantly, starting the iscsid daemon

All iBFT connections are brought up next by looking at the cards' iBFT data, if any. The filtering screen has a feature to add advanced storage devices, including iSCSI. Both connection types are handled by libiscsi (see below). The brought up iSCSI devices appear as /dev/sdX and are treated as ordinary block devices.

When DeviceTree scans all the block devices it uses the udev data (particularly the ID_BUS and ID_PATH keys) to decide if the device is an iscsi disk. If it is, it is represented with an iScsiDiskDevice class instance. This helps Anaconda remember that:

- we need to install dracut-network so the generated dracut image is able to bring up the underlying NIC and establish the iscsi connection.
- if we are booting from the device we need to pass dracut a proper set of arguments that will allow it to do so.

11.4 Libiscsi

How are iSCSI targets found and logged into? Originally Anaconda was just running iscsiadm as an external program through execWithRedirect(). This ultimately proved awkward especially due to the difficulties of handling the CHAP passphrases this way. That is why Hans de Goede <hdegoede@redhat.com>, the previous maintainer of the Anaconda iscsi subsystem decided to write a better interface and created libiscsi (do not confuse this with the libiscsi.c in kernel). Currently libiscsi lives as a couple of patches in the RHEL6 iscsi-initiator-utils CVS (and in Fedora package git, in somewhat outdated version). Since Anaconda is libiscsi's only client at the moment it is maintained by the Anaconda team.

The promise of libiscsi is to provide a simple C/Python API to handle iSCSI connections while being somewhat stable and independent of the changes in the underlying initiator-utils (while otherwise being tied to it on the implementation level).

And at the moment libiscsi does just that. It has a set of functions to discover and login to targets software targets. It supports making connections through partial offload interfaces, but the only discovery method supported at this moment is through firmware (iBFT). Its public data structures are independent of iscsi-initiator-utils. And there is some python boilerplate that wraps the core functions so we can easily call those from Anaconda.

To start nontrivial hacking on libiscsi prepare to spend some time familiarizing yourself with the iscsi-initiator-utils internals (it is complex but quite nice).

11.5 Debugging iSCSI bugs

There is some information in anaconda.log and storage.log but libiscsi itself is quite bad at logging. Most times useful information can be found by sshing onto the machine and inspecting the output of different iscsiadm commands [2][7], especially querying the existing sessions and known interfaces.

If for some reason the DeviceTree fails at recognizing iscsi devices as such, 'udevadm info –exportdb' is of interest.

The booting problems are either due to incorrectly generated dracut boot arguments or they are simply dracut bugs.

Note that many of the iscsi adapters are installed in different Red Hat machines and so the issues can often be reproduced and debugged.

11.6 Future of iSCSI in Anaconda

- extend libiscsi to allow initializing arbitrary connections from a partial offload card. Implement the Anaconda UI to utilize this. Difficulty hard.
- extend libiscsi with device binding support. Difficulty hard.
- work with iscsi-initiator-utils maintainer to get libiscsi.c upstream and then to rawhide Fedora. Then the partial offload patches in the RHEL6 Anaconda can be migrated there too and partial offload can be tested. This is something that needs to be done before RHEL7. Difficulty medium.
- improve libiscsi's logging capabilities. Difficulty easy.

CHAPTER 12

Multipath and Anaconda

Authors Ales Kozumplik <akozumpl@redhat.com>

12.1 Introduction

If there are two block devices in your /dev for which udev reports the same 'ID_SERIAL' then you can create a certain device mapper device which arbitrarily uses those devices to access the physical device. And that is Multipath [1].

For instance, suppose there are:

```
/dev/sda, with ID_SERIAL of 20090ef12700001d2, and
/dev/sdb, with the same ID_SERIAL.
```

Those are probably some adapters in the system that just connect your box to a storage area network (SAN) somewhere. There are perhaps two cables, one for sda, one for sdb, and if one of the cables gets cut the other can still transmit data. Normally the system won't recognize that sda and sdb have this special relation to each other, but by creating a suitable device map using multipath tools [2] we can create a DM device /dev/mapper/mpatha and use it for storing and retrieving data.

The device mapper then automatically routes IO requests to /dev/mapper/mpatha to either sda or sdb depending on the load of the line or network congestion on the particular network etc.

The nomenclature I will use here is: - 'multipath device' for the smart /dev/mapper/mpathX device. - 'multipath member device' for the '/dev/sdX' devices. Also 'a path'.

12.2 What is expected from Anaconda

Anaconda is expected to: - detect that there are multipath devices present - coalesce all relevant (e.g. exclusiveDisks) multipath devices. - only let the user interact with the multipath devices in filtering,

> cleardiskssel and partition screen, that is once we know 'sdc' and 'sdd' are part of 'mpathb' show only 'mpathb' and never the paths.

- install bootloader and boot from an mpath device
- make it happen so all the multipath devices (carrying or not the root filesystem) we used for installation are correctly coalesced in the booted system. This is achieved by generating a suitable /etc/multipath.conf and writing it into sysroot.
- be able to refer to mpath devices from kickstart, either by name like 'mpatha' or by their id like 'disk/by-id/scsi-20090ef12700001d2'

12.3 How Anaconda handles multipath

To detect presence of multipath devices we rely on multipath tools. The same we do for coalescing, see pyanaconda/storage/devicelibs/mpath.py, the file that provides some abstraction from mpath tools. During the device scan we use the 'multipath -d' output to find out what devices are going to end up as multipath members. The MultipathTopology object also enhances the multipath member's udev dictionaries with 'ID_FS_TYPE' set to 'multipath_member' (yes, this is a hack surviving from the original mpath implementation, and righteous is he who eradicates it). This information is picked up by DeviceTree when populating itself. Meaning, if 'sda' and 'sdb' are multipath member devices DeviceTree gives them MultipathMember format and creates one MultipathDevice for them (we know its name from 'multipath -d'). We end up with:

DiskDevice 'sda', format 'MultipathMember' DiskDevice 'sdb', format 'MultipathMember' MultipathDevice 'mpatha', parents are 'sda' and 'sdb'.

From then on, Anaconda only deals with the MultipathDevice and generally leaves anything with 'MultipathMember' format alone (understand, this is an inert format that really is not there but we use it just to mark the device as "useless beyond a multipath member", kind of like MDRaidMember).

Partition happens over the multipath device and during the preinstallconfig step /mnt/sysimage/etc/multipath.conf is created and filled with information about the coalesced devices. This is handled in the Storage.write() method. It is important this file and /etc/multipath/wwids (autogenerated by mpath tools) make it to the sysimage before the dracut image is generated.

12.4 Debugging multipath bugs

Unlike with iSCSI, to reproduce a multipath bug one does not need the same specific hardware as the reporter. Just found any box connected to a multipathed SAN and you are fine (at the moment, connecting to the same iSCSI target through its IPv4 and IPv6 address also produces a multipathed device).

On top of that, much of the necessary information is already included in the anaconda logs or can be easily extracted from the reporter. The things to particularly look at are:

- storage.log, the output around 'devices to scan for multipath' and 'devices post multipath scan'. The latter shows a triple with regular disks, disks comprising multipath devices and partitions. This helps you quickly find out what the target system is about.
- this information is also in program.log's calls to 'multipath' [3]. If mpath devices are mysteriously appearing/disappearing between filtering and partitioning screens look at those. 'multipath -ll' is called to display currently coalesced mpath devices, 'multipath -d' is called to show the mpath devices that would be coalesced if we ran 'multipath' now. This is exploited by the device filtering screen.

12.5 Future of multipath in Anaconda

Overall as of RHEL6.2, the shape of multipath in Anaconda is good and what's more important it is flexible enough to sustain new RFEs and bugs. Those are however bugs that I expect to appear sometime soon:

- enable or disable mpath_friendly_names in kickstart. Disabling friendly names just means the mpath devices are called by their wwid, e.g. /dev/mapper/360334332345343234, not '/dev/mapper/mpathc'. This is straightforward to implement.
- extend support for mpath devices in kickstart in general. Currently mpath devices should be accepted in most commands but I am sure there will be corner cases. Difficulty medium.
- [rawhide] stop extending the udev info dictionary with 'ID_FS_TYPE' and 'ID_MPATH_NAME'. Doing it this way is asking for the trouble if a dictionary of particular mpath device is reloaded from udev without running it through the MultipathTopology object as it will miss those entries (and DeviceTree depends on them a lot). Difficulty hard, but includes a lot of pleasant refactoring.
- Improve support for multipathing iSCSI devices. Someone might ask for it one day (in fact, with the NIC bounding they already did), and it will make mpath debugging possible on any virt machine with multiple virt NICs.

CHAPTER 13

The list-harddrives script

Authors Martin Kolman <mkolman@redhat.com>

13.1 Introduction

The list-harddrives script is primarily meant for use in the kickstart %post scriptlets for listing all individual harddrives on the system.

13.2 Output format

The list-harddrives script outputs two values per line separated by a single whitespace: - the device node name (eq. sda for /dev/sda) - the size in MB as a floating point number It does this for each individual harddrive on the system.

Example output:

sda 61057.3359375 sdb 476940.023438 sdc 30524.0

13.3 What devices are not listed

The list harddrives script will not list: - CD/DVD drives (/dev/sr*) - zram block devices - software RAID (/dev/md*) - all device mapper devices - anything that is not a block device

CHAPTER 14

Specification of the user interaction configuration file

Version 1.0

Authors Martin Kolman <mkolman@redhat.com>

This specification aims to establish a configuration file format that can be used to communicate information about which installation screens have been seen by the user during the installation. Optionally the configuration file might also contain information about which configuration options have been changed by the user and if the user has explicitly requested post installation setup tools to be disabled.

While this configuration file is primarily meant to be read (and potentially changed) by *post* installation tools (such as for example Initial Setup and Gnome Initial setup), Anaconda will take an existing configuration file into account at startup. This is meant to accommodate tools that are run *before* Anaconda is started, such as a system-wide language selection tool that runs before Anaconda is started and which sets language for both Anaconda and a Live installation environment.

14.1 Configuration file location

The user interaction configuration file is stored in: `/etc/sysconfig/anaconda`

14.2 General configuration file syntax

The configuration file is based on the INI file de-facto standard, eq.: key=value assignments and square bracket framed section headers.

Comments start with a hash (#) sign and need to be on a separate line. Inline comments (eq. behind section or key/value definitions) are not supported.

For Python programs this file format can be parsed and written by the ConfigParser[0] module available from the Python standard library. For programs written in C the GKeyFile[1] parser might be a good choice. Comparable INI file parsing and writing modules are available for most other programming languages.

Example:

```
# comment example - before the section headers

[section_1]
# comment example - inside section 1
key_a_in_section1=some_value
key_b_in_section1=some_value

[section_2]
# comment example - inside section 2
key_a_in_section2=some_value
```

Boolean values are marked with 1 for true and 0 for false.

Example:

```
true_key=1
false_key=0
```

14.3 Toplevel namespace

The toplevel configuration file namespace can only contain section headers.

There are two section types:

- One special section called *General* that can contain top-level settings not directly corresponding to any screen.
- Other sections that correspond to Anaconda spoke screens.
  - Hubs are not represented as sections as there is nothing that can be directly set on a hub in Anaconda.
  - All currently visible spokes in Anaconda will be enumerated as a section headers by Anaconda in the configuration file. Note that this can include spokes provided by addons[2], if any Anaconda addons that provide additional spokes are present during the installation run.

14.4 The General section

The *General* section is optional and is not required to be present in the config file. At the moment it can contain only the `post_install_tools_disabled` key.

The `post_install_tools_disabled` key corresponds to using the `firstboot --disable` command in the installation kickstart file. This requests that the post-installation setup tools be skipped. If this key is present and set to 1, any post-installation tools that parse the user interaction file should first make sure the tool won't be started again on next boot, and then terminate immediately.

14.5 Naming of sections corresponding to Anaconda screens

All section headers not named *General* are named according to the Anaconda spoke class name. For example `DatetimeSpoke` or `KeyboardSpoke`.

To get a list of all such spokes run the `list_spokes` script from the `scripts` directory in the Anaconda source code tree:

```
git clone https://github.com/rhinstaller/anaconda
cd anaconda/scripts
./list_screens
```

Note that this script only lists Anaconda spokes, not spokes provided by addons[2] or Initial Setup.

It is also possible to check for the *Entered spoke:* entries in the `/tmp/anaconda.log` file during an installation to correlate spokes on the screen to spoke class names.

14.6 Screen section namespace

Each section corresponding to a screen *must* contain the `visited` key with a value of either 1 if the user has visited the corresponding screen or 0 if not.

Optionally each section can contain one or more keys with the `changed_` prefix which track if the user has changed an option on the screen. If the option is changed by the user, the corresponding key is set to 1. If the given option has not been changed by the user then the corresponding key can either be omitted or set to 0.

Example:

```
[DatetimeSpoke]
visited=1
changed_timezone=1
changed_ntp=0
changed_timedate=1
```

In this example the user has visited the date & time spoke and has changed the timezone & time/date, but not the NTP settings. Note that the `changed_ntp` key could also be omitted as the user has not changed the NTP options.

Another example:

```
[KeyboardSpoke]
visited=0
```

Here the user has not visited the keyboard spoke and thus could not have changed any options, so all `changed_*` keys (if any) have been omitted.

Note that if a spoke section is missing, it should be assumed that the corresponding screen has not been visited. On the other hand, if a screen *has been visited*, the section *must* be present, with the `visited` key being equal to 1.

14.7 Full configuration file example

```
# this is the user interaction config file

[General]
post_install_tools_disabled=0

[DatetimeSpoke]
# the date and time spoke has been visited
visited=1
changed_timezone=1
changed_ntp=0
changed_timedate=1
```

(continues on next page)

(continued from previous page)

```
[KeyboardSpoke]
# the keyboard spoke has not been visited
visited=0
```

The first section is the special section for top-level settings called *General*. It contains only one option, `post_install_tools_disabled`, which is in this case equal to 0 This means that post installation setup tools should proceed as usual. In this case (being equal to 0) the `post_install_tools_disabled` key and the whole *General* section might also be omitted.

Next there are two sections corresponding to two screens - `DatetimeSpoke` and `KeyboardSpoke`.

The user has visited the date & time screen and has changed various options, but not the NTP settings. On the other hand the keyboard screen has not been visited at all.

14.8 Parsing and writing the of the configuration file by Anaconda

If the user interaction file exists during Anaconda startup, it will be parsed and taken into account when deciding which screens to show during the installation. This make it possible for secondary installation setup tools to run before Anaconda and query the user for information.

This can be for example a tool querying the user for language settings. Then once Anaconda starts it can skip the language selection screen as language has already been set by the tool.

Once the installation process is done, Anaconda will write out information about what screens the user has and has not visited and optionally which settings have been changed by the user.

If Anaconda successfully parsed an existing user interaction configuration file, any valid settings present in the file will propagate to the configuration file when it is written-out by Anaconda.

Note that comments present in the configuration file at the time Anaconda parses it might not be present in the output file, therefore tools should not depend on comments being present or on information contained in comments.

14.9 Parsing and writing of the configuration file by tools other than Anaconda

Non-Anaconda system configuration tools should also parse the user interaction file at startup and write it out once done. All valid data already present in the configuration file should be kept and updated accordingly (the user has visited a not-yet-visited screen, changed another option, etc.).

Non-Anaconda tools should try to keep comments present in the input file, but this is not strictly required.

Also note that a variable number of tools might be working with the configuration file in sequence, so no single tool should expect that it is the first or last tool working with the configuration file.

14.10 Links

- [0] https://docs.python.org/3/library/configparser.html
- [1] https://developer.gnome.org/glib/stable/glib-Key-value-file-parser.html
- [2] https://rhinstaller.github.io/anaconda-addon-development-guide/

CHAPTER 15

Testing Anaconda

This document describes how to run Anaconda tests. Anaconda has various tests such as unit tests, rpm tests and translation tests. All the tests will be run together if you follow the steps below.

You have two possible ways how to run these tests:

- running the tests directly on your system
- using mock utility which run a container on your system

Read below about their benefits and drawbacks.

15.1 Run tests locally

Before you are able to run Anaconda tests you need to install all required dependencies. To get list of dependencies you can use:

```
[dnf|yum] install -y $(./scripts/testing/dependency_solver.py)
```

Prepare the environment and build the sources:

```
./autogen.sh
./configure
make
```

Executing the tests can be done with:

```
make check
```

To run a single test do:

```
make TESTS=install/nosetests.sh check
```

See *tests/Makefile.am* for possible values. Alternatively you can try:

```
make ci
```

This has the advantage of producing Python test coverage for all tests. In case the *ci* target fails there is also a *coverage-report* target which can be used to combine the multiple *.coverage* files into one and produce a human readable report.

15.2 Run tests inside Mock

When using the 'ci' target in a mock you need to use a regular user account which is a member of the 'mock' group. You can update your account by running the command:

```
# usermod -a -G mock <username>
```

To prepare testing mock environment call:

```
./scripts/testing/setup-mock-test-env.py [mock-configuration]
```

Mock configuration can be path to a file or name of file in */etc/mock/*.cfg* without suffix. For detail configuration look on the script help output.

Then you can run tests by:

```
mock -r [mock_configuration] --chroot -- "cd /anaconda && ./autogen.sh && ./configure␣
↪&& make ci"
```

Or you can just attach to shell inside of the prepared mock environment:

```
mock -r [mock_configuration] --shell
```

15.3 Test Suite Architecture

Anaconda has a complex test suite structure where each top-level directory represents a different class of tests. They are

- *cppcheck/* - static C/C++ code analysis using the *cppcheck* tool;
- *dd_tests/* - Python unit tests for driver disk utilities (utils/dd);
- *dracut_tests/* - Python unit tests for the dracut hooks used to configure the installation environment and load Anaconda;
- *gettext/* - sanity tests of files used for translation; Written in Python and Bash;
- *glade/* - sanity tests for .glade files. Written in Python;
- *gui/* - specialized test suite for the graphical interface of anaconda. This is written in Python and uses the dogtail accessibility module. All tests are executed using ./anaconda.py from the local directory;
- *install/* - basic RPM sanity test. Checks if anaconda.rpm can be installed in a temporary directory without failing dependencies or other RPM issues;
- *lib/* - helper modules used during testing;
- *pyanaconda_tests/* - unit tests for the `pyanaconda` module;
- *pylint/* - checks the validity of Python source code using the *pocketlint* tool;
- *regex_tests/* - Python unit tests for regular expressions defined in `pyanaconda.regexes`;

- *storage/* - test cases used to verify partitioning scenarios for success or expected failures. The scenarios are described using kickstart snippets. Written in Python with a custom test case framework based on blivet;

Note: All Python unit tests inherit from the standard `unittest.TestCase` class unless specified otherwise!

Some tests require root privileges and will be skipped if running as regular user!
