Anaconda Documentation

Release 35.14

Anaconda Team

May 05, 2021

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.

For more news about Anaconda development and planned features you can follow our blog.

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`).

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 repositories added by kickstart (where –noverifyssl can be set per repo). Newly created additional repositories will honor this option.

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.

In the following example, Anaconda will try to fetch the image at first from `http://a`, then from `http://b` and finally from `http://c`.

```
inst.stage2=http://a inst.stage2=http://b inst.stage2=http://c inst.stage2.all
```

Without the boot option `inst.stage2.all`, Anaconda will try to fetch the image only from `http://c`, as usual.

```
inst.stage2=http://a inst.stage2=http://b inst.stage2=http://c
```

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.

In the following example, Anaconda will try to fetch the kickstart file at first from `http://a/a.ks`, then from `http://b/b.ks` and finally from `http://c/c.ks`.

```
inst.ks=http://a/a.ks inst.ks=http://b/b.ks inst.ks=http://c/c.ks inst.ks.all
```

Without the boot option `inst.ks.all`, Anaconda will try to fetch the kickstart file only from `http://c/c.ks`, as usual.

```
inst.ks=http://a/a.ks inst.ks=http://b/b.ks inst.ks=http://c/c.ks
```

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

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:

<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)`.

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

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).

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.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.

2.4.4 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.

2.4.5 inst.keymap

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

2.4.6 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.

2.4.7 inst.graphical

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

2.4.8 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.

2.4.9 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.

2.4.10 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.

2.4.11 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).

2.4.12 inst.vncpassword

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

2.4.13 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`.

2.4.14 inst.xdriver

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

2.4.15 inst.usefbx

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

Equivalent to `inst.xdriver=fbdev`.

2.4.16 inst.xtimeout

Specify the timeout in seconds for starting X server.

2.4.17 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.

2.5 Debugging and Troubleshooting

2.5.1 inst.debug

Run the installer in the debugging mode.

2.5.2 inst.rescue

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

2.5.3 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.

2.5.4 inst.nokill

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

2.5.5 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.

2.6 Boot loader options

2.6.1 inst.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 inst.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. It is not allowed to specify a variant name without a product name, so use `inst.product` to specify the product as well.

For example: `inst.product=Fedora 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_addon

`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 dns

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

2.9.3 netmask, gateway, hostname

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

2.9.4 ip=bootif

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

2.9.5 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 adding kernel modules to a denylist on its own; try `modprobe.blacklist=<mod1>,<mod2>...`

You can add the firewire module to a denylist 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.loglevel

The log level is always set to `debug`.

2.10.9 inst.mediacheck

Use the dracut option rd.live.check instead.

2.10.10 ks=floppy

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

2.10.11 display

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

2.10.12 utf8

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

2.10.13 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.

2.10.14 upgradeany

Anaconda doesn't handle upgrades anymore.

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

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

2.10.16 inst.zram

Anaconda doesn't run `zram.service` anymore. See `zram-generator` for more information.

2.10.17 inst.singlelang

Anaconda does not support single language mode anymore.

2.10.18 repo=nfsiso:...

Anaconda no longer needs explicit specification that a NFS location is an ISO image. The difference between an installable tree and a dir with an `.iso` file is now automatically detected, so this is the same as `inst.repo=nfs:...`

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.

Deprecated since Fedora 34.

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 overridded 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.

Deprecated since Fedora 34.

Note: You can use the configuration option `password_policies`.

Removed since Fedora 35.

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

Anaconda configuration files

The installer loads its default configuration from the Anaconda configuration files. The configuration can be modified by kernel arguments and cmdline options and the result is written into a runtime configuration file. The configuration is not supposed to change after that. The runtime configuration file is loaded by the Anaconda DBus modules when they are started. It means that all Anaconda processes are running with the same configuration.

Note: The `interactive-defaults.ks` file will be replaced by the Anaconda configuration files in the future. Kickstart files should be used only for the automatic installation.

4.1 File structure

The Anaconda configuration files are written in the INI format that can be processed by configparser. The files consist of sections, options and comments. Each section is defined by a `[section]` header. Each option is defined by a key and optionally a value separated by the = sign. Each comment has to start on a new line prefixed by the # character.

See an example of a section:

```
[Storage Constraints]

# Minimal size of the total memory.
min_ram = 320 MiB

# Should we recommend to specify a swap partition?
swap_is_recommended = False

# Recommended minimal sizes of partitions.
# Specify a mount point and a size on each line.
min_partition_sizes =
    /      250 MiB
    /usr   250 MiB
```

(continues on next page)

(continued from previous page)

```
# Required minimal sizes of partitions.
# Specify a mount point and a size on each line.
req_partition_sizes =
```

The supported sections and options are documented in the default configuration file.

4.2 Default configuration file

The default configuration file provides a full default configuration of the installer. It defines and documents all supported sections and options. The file is located at `/etc/anaconda/anaconda.conf`:

```
# Anaconda configuration file.
# Version: unstable


[Anaconda]
# Run Anaconda in the debugging mode.
debug = False

# Enable Anaconda addons.
addons_enabled = True

# List of enabled Anaconda DBus modules.
kickstart_modules =
    org.fedoraproject.Anaconda.Modules.Timezone
    org.fedoraproject.Anaconda.Modules.Network
    org.fedoraproject.Anaconda.Modules.Localization
    org.fedoraproject.Anaconda.Modules.Security
    org.fedoraproject.Anaconda.Modules.Users
    org.fedoraproject.Anaconda.Modules.Payloads
    org.fedoraproject.Anaconda.Modules.Storage
    org.fedoraproject.Anaconda.Modules.Services


[Installation System]
# Type of the installation system.
# FIXME: This is a temporary solution.
type = UNKNOWN

# Should the installer show a warning about unsupported hardware?
can_detect_unsupported_hardware = False

# Should the installer show a warning about removed support for hardware?
can_detect_support_removed = False

# Should the installer show a warning about enabled SMT?
can_detect_enabled_smt = False


[Installation Target]
# Type of the installation target.
type = HARDWARE

# A path to the physical root of the target.
```

(continues on next page)

(continued from previous page)

```
physical_root = /mnt/sysimage

# A path to the system root of the target.
system_root = /mnt/sysroot

# Should we install the network configuration?
can_configure_network = True


[Network]
# Network device to be activated on boot if none was configured so.
# Valid values:
#
#   NONE                    No device
#   DEFAULT_ROUTE_DEVICE    A default route device
#   FIRST_WIRED_WITH_LINK   The first wired device with link
#
default_on_boot = NONE


[Payload]
# Default package environment.
default_environment =

# List of ignored packages.
ignored_packages =

# Names of repositories that provide latest updates.
updates_repositories =

# List of .treeinfo variant types to enable.
# Valid items:
#
#   addon
#   optional
#   variant
#
enabled_repositories_from_treeinfo = addon optional variant

# Enable installation from the closest mirror.
enable_closest_mirror = True

# Enable possibility to skip packages with conflicts and broken dependencies.
enable_ignore_broken_packages = True

# Default installation source.
# Valid values:
#
#    CLOSEST_MIRROR  Use closest public repository mirror.
#    CDN             Use Content Delivery Network (CDN).
#
default_source = CLOSEST_MIRROR

# Enable ssl verification for all HTTP connection
verify_ssl = True

# GPG keys to import to RPM database by default.
```

(continues on next page)

(continued from previous page)

```
# Specify paths on the installed system, each on a line.
# Substitutions for $releasever and $basearch happen automatically.
default_rpm_gpg_keys =

[Security]
# Enable SELinux usage in the installed system.
# Valid values:
#
#  -1  The value is not set.
#   0  SELinux is disabled.
#   1  SELinux is enabled.
#
selinux = -1


[Bootloader]
# Type of the bootloader.
# Supported values:
#
#   DEFAULT   Choose the type by platform.
#   EXTLINUX  Use extlinux as the bootloader.
#
type = DEFAULT

# Name of the EFI directory.
efi_dir = default

# Hide the GRUB menu.
menu_auto_hide = False

# Are non-iBFT iSCSI disks allowed?
nonibft_iscsi_boot = False

# Arguments preserved from the installation system.
preserved_arguments =
    cio_ignore rd.znet rd_ZNET zfcp.allow_lun_scan
    speakup_synth apic noapic apm ide noht acpi video
    pci nodmraid nompath nomodeset noiswmd fips selinux
    biosdevname ipv6.disable net.ifnames net.ifnames.prefix
    nosmt


[Storage]
# Enable dmraid usage during the installation.
dmraid = True

# Enable iBFT usage during the installation.
ibft = True

# Do you prefer creation of GPT disk labels?
gpt = False

# Tell multipathd to use user friendly names when naming devices during the␣
↪installation.
multipath_friendly_names = True

# Do you want to allow imperfect devices (for example, degraded mdraid array devices)?
```

(continues on next page)

(continued from previous page)

```
allow_imperfect_devices = False

# Btrfs compression algorithm and level. e.g. zstd:1
btrfs_compression =

# Default file system type. Use whatever Blivet uses by default.
file_system_type =

# Default partitioning.
# Specify a mount point and its attributes on each line.
#
# Valid attributes:
#
#   size <SIZE>    The size of the mount point.
#   min <MIN_SIZE> The size will grow from MIN_SIZE to MAX_SIZE.
#   max <MAX_SIZE> The max size is unlimited by default.
#   free <SIZE>    The required available space.
#
default_partitioning =
    /     (min 1 GiB, max 70 GiB)
    /home (min 500 MiB, free 50 GiB)

# Default partitioning scheme.
# Valid values:
#
#   PLAIN      Create standard partitions.
#   BTRFS      Use the Btrfs scheme.
#   LVM        Use the LVM scheme.
#   LVM_THINP  Use LVM Thin Provisioning.
#
default_scheme = LVM

# Default version of LUKS.
# Valid values:
#
#   luks1  Use version 1 by default.
#   luks2  Use version 2 by default.
#
luks_version = luks2


[Storage Constraints]

# Minimal size of the total memory.
min_ram = 320 MiB

# Minimal size of the available memory for LUKS2.
luks2_min_ram = 128 MiB

# Should we recommend to specify a swap partition?
swap_is_recommended = False

# Recommended minimal sizes of partitions.
# Specify a mount point and a size on each line.
min_partition_sizes =
    /      250 MiB
    /usr   250 MiB
```

(continues on next page)

(continued from previous page)

```
    /tmp   50  MiB
    /var   384 MiB
    /home  100 MiB
    /boot  512 MiB

# Required minimal sizes of partitions.
# Specify a mount point and a size on each line.
req_partition_sizes =

# Allowed device types of the / partition if any.
# Valid values:
#
#   LVM         Allow LVM.
#   MD          Allow RAID.
#   PARTITION   Allow standard partitions.
#   BTRFS       Allow Btrfs.
#   DISK        Allow disks.
#   LVM_THINP   Allow LVM Thin Provisioning.
#
root_device_types =

# Mount points that must be on a linux file system.
# Specify a list of mount points.
must_be_on_linuxfs = / /var /tmp /usr /home /usr/share /usr/lib

# Paths that must be directories on the / file system.
# Specify a list of paths.
must_be_on_root = /bin /dev /sbin /etc /lib /root /mnt lost+found /proc

# Paths that must NOT be directories on the / file system.
# Specify a list of paths.
must_not_be_on_root =

# Mount points that are recommended to be reformatted.
#
# It will be recommended to create a new file system on a mount point
# that has an allowed prefix, but doesn't have a blocked one.
# Specify lists of mount points.
reformat_allowlist = /boot /var /tmp /usr
reformat_blocklist = /home /usr/local /opt /var/www


[User Interface]
# The path to a custom stylesheet.
custom_stylesheet =

# The path to a directory with help files.
help_directory = /usr/share/anaconda/help

# Default help pages for TUI, GUI and Live OS.
default_help_pages =

# Is the partitioning with blivet-gui supported?
blivet_gui_supported = True

# A list of spokes to hide in UI.
# FIXME: Use other identification then names of the spokes.
```

(continues on next page)

(continued from previous page)

```
hidden_spokes =

# Run GUI installer in a decorated window.
decorated_window = False

# Should the UI allow to change the configured root account?
can_change_root = False

# Should the UI allow to change the configured user accounts?
can_change_users = False

# Define the default password policies.
# Specify a policy name and its attributes on each line.
#
# Valid attributes:
#
#   quality <NUMBER>  The minimum quality score (see libpwquality).
#   length <NUMBER>   The minimum length of the password.
#   empty             Allow an empty password.
#   strict            Require the minimum quality.
#
password_policies =
    root (quality 1, length 6)
    user (quality 1, length 6, empty)
    luks (quality 1, length 6)


[License]
# A path to EULA (if any)
#
# If the given distribution has an EULA & feels the need to
# tell the user about it fill in this variable by a path
# pointing to a file with the EULA on the installed system.
#
# This is currently used just to show the path to the file to
# the user at the end of the installation.
eula =
```

4.3 Product configuration files

The product configuration files allow to override some of the configuration options for specific products. The files are located at `/etc/anaconda/product.d/`.

Note: Anaconda previously used so called install classes for the product-specific configuration. Install classes were completely removed and replaced by the product configuration files.

4.3.1 Product identification

The product is defined by a product name and optionally a variant name. For example, Fedora Server has a product name `Fedora` and a variant name `Server`. See the terminology of productmd.

The product can be specified by the boot options `inst.product` and `inst.variant`. Otherwise, it will be automatically loaded from the `.buildstamp` file on the installation media. This file is generated by `lorax` using the `--product` and `--variant` options.

Based on the provided product and variant names, the installer will look up the right configuration file in the `/etc/anaconda/product.d/` directory.

4.3.2 File structure

Product configuration files have one or two extra sections that identify the product. The `[Product]` section specifies the product and variant names of a product. The `[Base Product]` section specifies the product and variant names of a base product if any. For example, Fedora is a base product of Fedora Server.

Note: We are not going to support wildcards in product names. This used to be supported in install classes and it caused a lot of problems. The product name can match unrelated configurations and you cannot be sure which configuration is going to be used at the end.

We support a simple inheritance of product configurations. The installer loads configuration files of the base products before it loads the configuration file of the specified product. For example, it will first load the configuration for Fedora and then the configuration for Fedora Server.

Note: We are not going to support multiple inheritance. It would significantly increase the complexity of the product configuration files in an unintuitive way. You can easily compare two configuration files and verify the parts they are supposed to share. We do that in our unit tests.

See an example of the product configuration file for Fedora Server:

```
# Anaconda configuration file for Fedora Server.

[Product]
product_name = Fedora
variant_name = Server

[Base Product]
product_name = Fedora

[Payload]
default_environment = server-product-environment

[Storage]
file_system_type = xfs
default_scheme = LVM
```

4.4 Custom configuration files

The custom configuration files allow to override some of the configuration options for specific installations. The files are located at `/etc/anaconda/conf.d/`.

The installer finds all files with the `.conf` extension in the `/etc/anaconda/conf.d/` directory, sorts them by their name and loads them in this order. These files are loaded after the product configuration files, so they have a higher priority.

For example, the initial setup installs the `10-initial-setup.conf` file with a custom configuration.

Note: All configuration files have to be loaded before the installer starts to parse the kickstart file, so it is not possible to generate a configuration file in the `%pre` section of the kickstart file. Please, use `updates.img` or `product.img` instead.

4.5 Runtime configuration file

The runtime configuration file is a temporary file that provides a full configuration of the current installer run. It is generated by the installer and it exists only during its lifetime. The file is located at `/run/anaconda/anaconda.conf`.

The runtime configuration file is loaded by the Anaconda DBus modules when they are started. It allows us to run all Anaconda processes with the same configuration.

The installer makes the following steps to create the runtime configuration file. The configuration is not supposed to change after that.

1. Load the default configuration file from `/etc/anaconda/anaconda.conf`.
2. Load the selected product configuration files from `/etc/anaconda/product.d/*.conf`.
3. Load the custom configuration files from `/etc/anaconda/conf.d/*.conf`.
4. Apply the kernel arguments.
5. Apply the cmdline options.
6. Generate the runtime configuration file `/run/anaconda/anaconda.conf`.

4.6 Python representation

The Anaconda configuration is represented by the `conf` object from `pyanaconda.core.configuration.anaconda`. The configuration sections are represented by properties of the `conf` object. The configuration options are represented by properties of the section representation. All these properties are read-only.

The `conf` object is initialized on the first import. It loads the runtime configuration file, if it exists, otherwise it loads the default configuration file. Its main purpose is to provide access to the configuration of the current installer run.

It is safe to use the `conf` object in the Anaconda DBus modules and in any other Python processes that are started after a runtime configuration file has been generated.

See an example of a Python code:

```
from pyanaconda.core.configuration.anaconda import conf

# Is Anaconda in the debugging mode?
print(conf.anaconda.debug)

# Is the type of the installation target hardware?
print(conf.target.is_hardware)

# A path to the system root of the target.
print(conf.target.system_root)
```

CHAPTER 5

Reporting bugs

Please, report a new bug or a feature request on Bugzilla against the anaconda component. You can use one of the quick links for Fedora or RHEL.

For the bug report, attach logs from the installation. You can find them during the installation at `/tmp/*log` or on the installed system at `/var/log/anaconda/*log`. These files are usually essential for the investigation.

CHAPTER 6

Common bugs and issues

Below you will find the most common bugs and issues, that we encounter at Bugzilla, and their solutions.

6.1 Bug report issues

These issues require more information from the reporter.

6.1.1 Too old version of Anaconda

Issue The bug is reported against a too old version of the operating system. It it possible that the code has changed and the problem no longer exists.

Solution Are you able to reproduce the problem with Fedora XY?

6.1.2 Missing logs

Issue There are no useful logs attached to the bug.

Solution Please, attach all files with installation logs, especially the file named `syslog`. You can find them during the installation in `/tmp` or on the installed system in `/var/log/anaconda/`.

6.2 DBus issues

Anaconda runs several DBus modules and communicates with them from the user interface, so you can easily come across a DBus-related issue.

6.2.1 Traceback of DBusError

Issue Anaconda fails with the `dasbus.error.DBusError` exception. This usually happens when a DBus module raises an unexpected exception. Anaconda shows a traceback only for the DBus call, so it is necessary to look up a traceback of the DBus module to have complete information about the bug.

Solution You can find the original exception in the logs (usually in `syslog` or in the output of `journalctl`).

Example rhbz#1828614

6.3 Installation environment issues

You can find here issues related to the installation environment. Anaconda usually runs in the stage2 environment provided by `boot.iso`, in Live OS, in a mock environment or locally.

6.3.1 Mismatched stage2

Issue Anaconda fails early in stage2 with an exception "ValueError: new value non-existent xfs filesystem is not valid as a default fs type".

Solution This error occurs when `initrd.img`, `vmlinuz` and the repository (or stage2) are not from the same media or location.

Example rhbz#1169034

6.3.2 Out of memory

Issue Anaconda fails in stage1 with a message "Failed writing body" or "No space left on device" in the dracut logs. This usually happens when installing from http or ftp source on a machine with insufficient memory size. See the minimal requirements for RHEL.

Solution Increase the memory size or try installing from NFS, CD-Rom or HDD source.

Example rhbz#1630763

6.3.3 Changes in Live OS

Issue The Live OS requires changes.

Solution Reassigning to spin-kickstarts.

6.3.4 Changes in boot.iso

Issue The `boot.iso` requires changes.

Solution Reassigning to lorax.

6.3.5 Icon issues

Issue The Anaconda icons in Live OS requires changes.

Solution Reassigning to fedora-logos

Example rhbz#1699034

6.3.6 Font issues

Issue In the Welcome spoke, there are replacement glyphs (rectangles) instead of characters in a name of a language. This usually means that that there is no font for this language installed in the installation environment.

Solution Reassigning to lorax or spin-kickstarts.

Example rhbz#1530086

6.4 Payload issues

These issues are related to the content that is installed on the target system.

6.4.1 Non-fatal POSTIN scriptlet failure

Issue The package installation fails with a message "Non-fatal POSTIN scriptlet failure in rpm package". The failing package has to fix its scriptlet, because all scriptlets MUST exit with the zero exit status.

Solution All RPM errors are fatal during the installation (see the bug 1565123). Reassigning.

Example rhbz#1588409

6.4.2 Changes in package groups and environments

Issue The reporter wants a new package to be installed by default.

Solution Reassigning to comps.

Example rhbz#1787018

6.4.3 Corrupted ISO

Issue The package installation fails with a message "Some packages from local repository have incorrect checksum". This happens when the packages cannot be accessed, because they are located on a corrupted ISO or an unmounted device.

Solution The ISO might be corrupted. Please, try to download it again and verify the checksum.

Example rhbz#1551311

6.4.4 Issues with live payload

Issue The image installed by the live OS payload requires changes.

Solution Anaconda doesn't create the live image. Reassigning to spin-kickstarts.

6.4.5 Issues with OSTree

Issue The installation with the OSTree payload fails.

Solution It might be related to the OSTree payload. Reassigning to Colin Walters.

6.4.6 Failed to mount the install tree

Issue The payload fails to set up and raises the error "Failed to mount the install tree". This usually happens when Anaconda is unexpectedly terminated and started again. Some of the Anaconda's mount points stays mounted and that causes the crash.

Example rhbz#1562239

6.4.7 System upgrades

Issue The system was upgraded, not installed.

Solution Anaconda is not doing system upgrades. That is done by dnf-system-upgrade. Reassigning to dnf.

6.5 Storage issues

These issues are related to hardware, partitioning and storage configuration.

6.5.1 Bug in blivet

Issue The exception starts in `blivet` or `libblockdev`.

Solution It seems to be an issue in the storage configuration library. Reassigning to blivet.

Example rhbz#1827254

6.5.2 Bug in blivet-gui

Issue The exception starts in `blivet-gui` or there is a problem with partitioning and the reporter used Blivet-GUI as the partitioning method.

Solution It seems to be an issue in blivet-gui. Reassigning.

Example rhbz#1833775

6.5.3 Failing hardware

Issue The logs (journal or syslog) are full of kernel messages about I/O errors. For example:

```
kernel: [sdb] tag#9 FAILED Result: hostbyte=DID_OK driverbyte=DRIVER_
↪SENSE
kernel: [sdb] tag#9 Sense Key : Medium Error [current]
kernel: [sdb] tag#9 Add. Sense: Unrecovered read error - auto reallocate␣
↪failed
```

(continues on next page)

(continued from previous page)

```
kernel: [sdb] tag#9 CDB: Read(10) 28 00 1d 04 10 00 00 00 08 00
kernel: print_req_error: I/O error, dev sdb, sector 486805504
```

Solution It looks like a hardware failure. Please, check your hardware.

Example rhbz#1685047

6.5.4 LVM on disks with inconsistent sector size

Issue The storage configuration fails with an error message mentioning "inconsistent sector size".

Solution LVM is now demanding that all disks have consistent sector size, otherwise they can't be used together. Please adjust your disk selection to use only disks with the consistent sector size.

Example rhbz#1754683

6.5.5 Unlocked LUKS

Issue The storage configuration fails with a message "luks device not configured".

Solution Anaconda doesn't support LUKS devices that are unlocked outside the installer. The device has to be unlocked in Anaconda.

Example rhbz#1624856

6.5.6 Undetected partitions

Issue When the custom partitioning spoke is entered, it raises an exception with a message: "cannot initialize a disk that has partitions". Anaconda tries to initialize disks that are supposed to be empty, but there are partitions that were not discovered by kernel after boot.

Solution Duplicate of the bug 1825067.

Example rhbz#1828188

6.5.7 Too little memory for LUKS setup

Issue Anaconda crashes with an exception: `No such interface "org.freedesktop.DBus.Properties" on object at path /org/fedoraproject/Anaconda/Modules/Storage/Task/.`

Solution The installation environment does not have enough memory to run LUKS setup, and its crash resets the Storage module. In logs, the following lines can be found:

- `WARNING:blivet:Less than (...) MiB RAM is currently free, LUKS2 format may fail.`
- `ui.gui.spokes.storage: Partitioning has been applied: ValidationReport(error_messages=[], warning_messages=['The available memory is less than 128 MiB which can be too small for LUKS2 format. It may fail.'])`
- `Activating service name='org.fedoraproject.Anaconda.Modules.Storage'` (present more than once)

Note that the user must have ignored a warning in the GUI.

Workaround There are several possible workarounds:

- Use more memory for the machine,
- use `--pbkdf*` options in kickstart file,
- change LUKS version to `LUKS1`,
- disable encryption.

Example rhbz#1902464

6.6 Bootloader issues

There issues are related to bootloader issues.

6.6.1 Bug in bootloader

Issue The exception is raised during a bootloader installation with a message that usually says "failed to write bootloader" or "boot loader install failed". Look into `program.log` or `storage.log` for more information.

Solution Could the bootloader team have a look at this bug, please?

6.6.2 Disable `rhgb quiet`

Issue The reporter doesn't want the default boot options `rhgb quiet` to be used.

Solution The installer adds the boot options `rhgb quiet` only if `plymouth` is installed. In a kickstart file, you can disable these options with the following snippet:

```
%packages
-plymouth
%end
```

6.6.3 Invalid environment block

Issue The bootloader installation fails with an exception "failed to write boot loader configuration". You can find the following message in the logs:

```
/usr/bin/grub2-editenv: error: invalid environment block
```

Solution Duplicate of the bug 1814690.

Example rhbz#1823104

6.7 User interface issues

These issues are related to the text and graphical user interfaces of the installation program.

6.7.1 Allocating size to pyanaconda+ui+gui+MainWindow

Issue Anaconda shows a Gtk warning "Allocating size to pyanaconda+ui+gui+MainWindow without calling gtk_widget_get_preferred_width/height(). How does the code know the size to allocate?"

Solution This is an issue in the GTK library: See: https://gitlab.gnome.org/GNOME/gtk/issues/658

Example rhbz#1619811

6.7.2 Bug in Gtk

Issue When Anaconda is started in the graphical mode, some of the Gtk widgets look weird.

Solution Reassigning to gtk3.

6.7.3 Weirdly displayed GUI

Issue When Anaconda is started in the graphical mode, the whole screen looks weird.

Solution It looks like an Xorg or kernel issue. Reassigning to xorg-x11 for further triaging.

6.7.4 Rotated screen

Issue The screen is rotated.

Solution It seems to be a problem with drivers. Reassigning to kernel.

Contact kernel or iio-sensor-proxy

6.8 Localization issues

These issues are related to the localization support in Anaconda.

6.8.1 Changes in localization data

Issue Languages, locales, keyboard layouts or territories are not correct.

Solution This content is provided by langtable. Reassigning.

Example rhbz#1698984

6.9 Kickstart issues

These issues are related to automated installations that use kickstart files.

6.9.1 Automatic installation in Live OS

Issue The reporter would like to run a kickstart installation in Live OS.

Solution Kickstart installations in Live OS are not supported. Please, run the installation with `boot.iso`.

Example rhbz#1027160

6.9.2 Invalid partitioning in the output kickstart file

Issue The kickstart file generated by Anaconda at the end of the installation defines an invalid partitioning.

Solution This part of the kickstart file is generated by the storage configuration library. Reassigning to blivet.

Example rhbz#1851230

6.9.3 The *ignoredisk –only-use* command hides installation sources

Issue The installer fails to find an installation media on the USB drive if the *ignoredisk –only-use=* command is specified in a kickstart file.

Workaround You can use the *harddrive* command instead of the *cdrom* command. For example:

harddrive –partition=sda –dir=/

where *sda* is the name of the USB device, or use *LABEL*:

harddrive –partition=LABEL=CentOS-8-3-2011-x86_64-dvd –dir=/

Example rhbz#1945779

6.9.4 Missing options of the *repo* command

Issue The *repo* kickstart command doesn't support the requested configuration options.

Workaround We get a lot of feature requests for the *repo* command, but we don't really want to support every repo configuration option. Please, use a repo file to configure the repo.

For example:

```
# Enable the custom repo.
repo --name "my-custom-repo"

%pre
# Generate the custom repo file.
cat >> /etc/anaconda.repos.d/custom.repo << EOF

[my-custom-repo]
name=My Custom Repository
baseurl=http://my/custom/repo/url/
priority=10
module_hotfixes=1

EOF
%end
```

6.9.5 Enabling root password SSH login via password.

Issue There is no kickstart command or option to enable password based root login via SSH.

Solution It's really not good practice to enable password based SSH root login on a machine as the attacker only needs to guess a password for root and then gets full access to the machine. For a user account the attacker needs to guess both the username and password and might only get to a non-admin user, making such an attack much harder and less worthwhile.

This was the reasoning for the OpenSSH project disabling password logins for root back in 2015. Fedora patched this out temporarily but in 2019 it was decided to drop this downstream patch and respect the upstream behavior of not allowing password based root login over SSH by default. Anaconda accommodated this change by adding and override checkbox in the root password GUI to make the transition easier for users still needing SSH login via root during the transition period.

While there is currently no set deadline for removing the checkbox from the GUI, it is still considered a temporary element helping users during the transition to future where no use cases requiring password based root login exist. The option will most likely be dropped in the longer term, when it is considered no longer necessary - of course with a proper heads-up and feedback period for the Anaconda user community.

This is also the reason why we did not add any kickstart support for the SSH root password login override - dropping something from the GUI is certainly not without impact, but doing the same for a kickstart command or option is much harder.

Also as already mentioned above, enabling password based root login over SSH can quite significantly compromise the security of a system and should be an explicit and easy to spot action performed by the user during the installation. Clicking a checkbox satisfies this condition in the GUI.

A kickstart command option on the other hand could be easily missed during the common practice of reusing kickstarts and kickstart snippets - there are already quite a few options even just for the rootpw command and one more option copy pasted from a test-run kickstart could easily by missed & turn all production image installs vulnerable to remote password guessing attack.

Workaround If you really need to enable password based SSH root login, you can just easily use the following two line %post script (one line without comments):

```
%post
# permit root login via SSH with password authetication
echo "PermitRootLogin yes" > /etc/ssh/sshd_config.d/01-permitrootlogin.
↪conf
%end
```

This does 100% the same as a rootpw command option would, but unlike the option is quite explicit about what it does and easy to spot in a kickstart file. Or even better, use the sshkey command to use a key instead of password, making a remote guessing attack essentially impossible.

CHAPTER 7

Contribution guidelines

This guide describes rules for how to get your contributions into Anaconda. However, if you seek help with implementing changes in Anaconda, please follow our blog series or an addon guide to create Anaconda addon.

7.1 How to run make commands

Anaconda has plenty of dependencies and because of that it's hard to set environment to with Anaconda properly. To get all the dependencies you are free to use script in the Anaconda repository.

Follow these steps to keep your machine clean from all the Anaconda dependencies. It will create a container where you can install all the dependencies. If you are not interested in dealing with container just skip this part and continue on the next one:

```
sudo dnf install toolbox
toolbox create
toolbox enter
```

To prepare the environment in the container or on your system just run these commands:

```
sudo ./scripts/testing/install_dependencies.sh
./autogen.sh && ./configure
```

7.2 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.

7.3 Which is my target git branch?

Depending on where you want to make your contribution please choose your correct branch based on the table below.

Fedora Rawhide	master
Fedora XX	fXX-devel
RHEL-X / CentOS Stream X	rhel-x

7.4 Finding Bugs to Fix

The development team can mark bugs with specific keywords to show that they belong to a specific category. You can quickly list these by searching the Red Hat bugzilla for bugs in the `anaconda` component with specific keywords in Whiteboard:

- For good first issues and simple fixes, the keyword is EasyFix.
- For Btrfs-related issues, use keyword Btrfs.
- For issues that are good candidates for pure community features, search for CommunityFeature.

(A single issue could potentially have more than one of these keywords.)

Patches for bugs without keywords are welcome, too!

7.5 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)

7.6 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

7.7 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

7.8 Guidelines for Commits

7.8.1 Commit Messages

The first line should be a succinct description of what the commit does, starting with capital and ending without a period ('.'). 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.

7.8.2 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.

7.8.3 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.

7.9 Code conventions

It is important to have consistency across the codebase. This won't necessarily make your code work better, but it might help to make the codebase more understandable, easier to work with, and more pleasant to go through when doing a code review.

In general we are trying to be as close as possible to PEP8 but also extending or modifying minor PEP8 rules when it seems suitable in the context of our project. See list of the conventions below:

- Limit all lines to a maximum of 99 characters.
- **Format strings with .format() instead of `%` (https://pyformat.info/)**
  - Exception: Use `%` formatting in logging functions and pass the `%` as arguments. See logging format interpolation for the reasons.
- Follow docstring conventions. See PEP257.
- Use Enum instead of constants is recommended.
- Use `super()` instead of `super(ParentClass, self)`.
- Use only absolute imports (instead of relative ones).
- Use `ParentClass.method(self)` only in case of multiple inheritance.
- Instance variables are preferred, class variables should be used only with a good reason.
- Global instances and singletons should be used only with a good reason.
- Never do wildcard (`from foo import *`) imports with the exception when all Anaconda developers agree on that.
- Use `raise` & `return` in the doc string. Do not use `raises` or `returns`.
- Methods that return a task should have the suffix '_with_task' (for example discover_with_task and DiscoverWithTask).
- Prefer to use `pyanaconda.util.join_paths` over `os.path.join`. See documentation for more info.
- Never call `upper()` on translated strings. See the bug 1619530

- Names of signal handlers defined in `.glade` files should have the `on_` prefix.

7.10 Merging examples

7.10.1 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
```

7.10.2 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).

7.10.3 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).

7.11 Pure community features

The pure community features are features which are part of the Anaconda code base but they are maintained and extended mainly by the community. These features are not a priority for the Anaconda project.

In case of issues in pure community features, the Anaconda team will provide only sanity checking. It is the responsibility of the community (maintainers of the feature) to provide fix for the issue. If the issue will have bigger impact on other parts of the Anaconda project or if it will block a release or another priority feature and the fix won't be provided in a reasonable time the Anaconda team reserves the rights to remove or disable this feature from the Anaconda code base.

Below is a list of pure community features, their community maintainers, and maintainers contact information:

7.11.1 /boot on btrfs subvolume

- Origin: https://github.com/rhinstaller/anaconda/pull/2255
- Bugzilla: https://bugzilla.redhat.com/show_bug.cgi?id=1418336
- Maintainer: Neal Gompa <ngompa13@gmail.com>
- Description:

```
Enable boot of the installed system from a BTRFS subvolume.
```

CHAPTER 8

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 9

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
- it is recommended to make the release on a fresh clone (prevent you from pushing local work into the upstream repository)
- 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 https://github.com/rhinstaller/anaconda-l10n localization repository
- 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

9.1 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
```

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
```

14. push new translations

```
make po-push
```

15. check repository on path returned by the above command and push if it's correct

CHAPTER 10

Upcoming 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 11

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.

11.1 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 12

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.

12.1 Create new localization branch for Anaconda

First thing which needs to be done before branching in Anaconda is to create a new localization branch which will be used by the new Anaconda branch.

Start by cloning translation repository (ideally outside of Anaconda git) and enter this repository:

```
git clone git@github.com:rhinstaller/anaconda-l10n.git
cd anaconda-l10n
```

Create a new localization directory from `master` directory:

```
cp -r master f<version>
```

Add the new folder to git:

```
git add f<version>
```

Commit these changes:

```
git commit -m "Branch new Fedora <version> from master"
```

Push new localization directory. This will be automatically discovered and added by Weblate service:

```
git push origin
```

12.2 How to branch Anaconda

FIXME: This does not reflect latest changes required by containers for CI and tests.

First make sure that localization branch for the next Fedora is already created.

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
```

Switch to f<version>-release branch for Fedora specific settings:

```
git checkout f<version>-release
```

Edit branch specific settings. This have to be done on f<version>-release branch only:

```
vim ./branch-config.mk
```

And change content according to comments in the file.

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.

Check if everything is correctly set:

```
make check-branching
```

If everything works correctly you can push the branches to the origin (`-u` makes sure to setup tracking) :

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

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

12.3 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
```

Next add the third (release) version number:

```
./scripts/makebumpver -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.

12.4 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`

Make sure you are in the Rawhide branch:

```
git checkout master
```

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

```
./scripts/makebumpver -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.

CHAPTER 13

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.

13.1 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.

13.2 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.

13.3 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.

13.4 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.

13.5 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>".

13.6 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

13.7 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.

13.8 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.

13.9 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 14

iSCSI and Anaconda

Authors Ales Kozumplik <akozumpl@redhat.com>

14.1 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.

14.2 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].

14.3 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.

14.4 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).

14.5 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.

14.6 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 15

Multipath and Anaconda

Authors Ales Kozumplik <akozumpl@redhat.com>

15.1 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'.

15.2 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'

15.3 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.

15.4 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.

15.5 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 16

The list-harddrives script

Authors Martin Kolman <mkolman@redhat.com>

16.1 Introduction

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

16.2 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

16.3 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 17

Anaconda sysconfig file

This specification aims to establish a configuration file format that can be used to configure post-installation tools. 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).

17.1 Configuration file location

The Anaconda sysconfig file is stored in: `/etc/sysconfig/anaconda`

17.2 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]
```

(continues on next page)

(continued from previous page)

```
# 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
```

17.3 Toplevel namespace

The toplevel configuration file namespace can only contain section headers.

There is only one special section called *General* that can contain top-level settings not directly corresponding to any screen.

17.4 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 Anaconda sysconfig file should first make sure the tool won't be started again on next boot, and then terminate immediately.

17.5 Full configuration file example

```
# This is the Anaconda sysconfig file.

[General]
post_install_tools_disabled=0
```

The specified 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.

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

Non-Anaconda system configuration tools should parse the Anaconda sysconfig file at startup and write it out once done. All valid data already present in the configuration file should be kept and updated accordingly.

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.

17.7 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 18

Installation mount points

Below you can find mount points that the installer uses during the installation.

18.1 Target system

The root of the target system is mounted under the directories `/mnt/sysimage` and `/mnt/sysroot`.

18.1.1 /mnt/sysimage

This is a mount point of the physical root of the target system. It is used to mount a device that contains `/` of the target system.

18.1.2 /mnt/sysroot

This is a mount point of the system root of the target system. It is used to mount `/` of the target system.

Usually, the physical root and the system root are the same, so `/mnt/sysroot` is attached to the same file system as `/mnt/sysimage`. The only exceptions are rpm-ostree systems, where the system root is changing based on the deployment. Then `/mnt/sysroot` is attached to a subdirectory of `/mnt/sysimage`.

It is recommended to use `/mnt/sysroot` for `chroot`.

CHAPTER 19

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. For integration tests there is a separate repository kickstart-tests containing also tooling for running the tests.

19.1 Run unit tests inside of container

This is the primary and recommended way to run the tests.

Right now only unit tests are supported by the container, not rpm-tests. You can use our container image on quay.io or you can build your own image. (Optional) to build the container image run:

```
make -f Makefile.am anaconda-ci-build
```

Then you are free to run the tests without dependency installation by running:

```
make -f Makefile.am container-ci
```

This will run all the tests, including Python test coverage reports. To run just some tests you can pass parameters which will replace the current one. For example to run just some nose-tests please do this:

```
make -f Makefile.am container-ci CI_CMD="make tests-nose-only NOSE_TESTS_
↪ARGS=nosetests/pyanaconda_tests/kernel_test.py"
```

WARNING:

Just one command can be passed like this, if *&&* is used then only first one is run in the container but everything else is started on host!

Logs from the run are stored in the `test-logs/` folder; no other files are modified/touched by the container (it works on an internal copy of the host's anaconda directory).

19.2 Interactively work inside of container

For interactively working in the container you can run:

```
make -f Makefile.am container-shell
```

This command will open bash inside the container for you with mounted current folder at the */anaconda* path. This is a convenient way how to run tests but avoid constant call of autotools and build during the development.

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=nosetests.sh check
```

To run a subset of nose tests do:

```
make TESTS=nosetests.sh NOSE_TESTS_ARGS="nosetests/pyanaconda_tests/keyboard_test.py␣
↪nosetests/pyanaconda_tests/timezone_test.py" 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.

19.3 Note

Please update your container from time to time to have newest dependencies. To do that, run *podman pull quay.io/rhinstaller/anaconda-ci:master* or build it locally again.

19.4 Run rpm tests inside of container

First, build the container image for running the test, as it does not yet get published to any registry:

```
make -f Makefile.am anaconda-rpm-build
```

Then run the test in that container:

```
make -f Makefile.am container-rpm-test
```

19.5 GitHub workflows

All test and maintenance actions are run by GitHub workflows. These YAML files completely describe what steps are required to run some action, what are its triggers and so on.

19.5.1 Pull request for master:

Unit and rpm tests are run by the validate.yml workflow. We use GitHub's runners for this so we don't have to care about what is executed there.

The workflow rebuilds the `anaconda-ci` container if the container files have changed, otherwise it is pulling the container from quay.io. For more information see below.

19.5.2 Pull request for RHEL:

Unit and rpm tests are run by the validate-rhel-8.yml workflow on (fully automatically deployed) self-hosted runners in our Upshift instance.

These runners are `anaconda-ci:rhel8` containers with all the dependencies in place so the yml configuration will just execute tests. You can start runners locally by running the container and providing GitHub token. That is pretty valuable in case of workflow testing. See github-action-run-once for more details.

To protect our self-hosted runners, tests only run automatically for rhinstaller organization members. For external contributors, an organization member needs to approve the test run by sending a comment starting with `/tests`.

19.5.3 Running kickstart-tests:

The kickstart-tests.yml workflow allows rhinstaller organization members to run kickstart-tests against an anaconda PR (only `master` for now). Send a comment that starts with `/kickstart-tests <launch options>` to the pull request to trigger this. See the kickstart launch script documentation and its `--help` for details what is supported; the two basic modes are running a set of individual tests:

```
/kickstart-tests keyboard [test2 test3 ...]
```

or running all tests of one or more given types:

```
/kickstart-tests --testtype network,autopart
```

19.6 Container maintenance

All active branches run tests in containers. Containers have all the dependencies installed and the environment prepared to run tests or connect our GitHub runners (used by RHEL only).

19.6.1 Automatic container build

Containers are updated daily by the container-autoupdate.yml workflow from Anaconda `master` repository. Before pushing a new container, tests are executed on this container to avoid regressions.

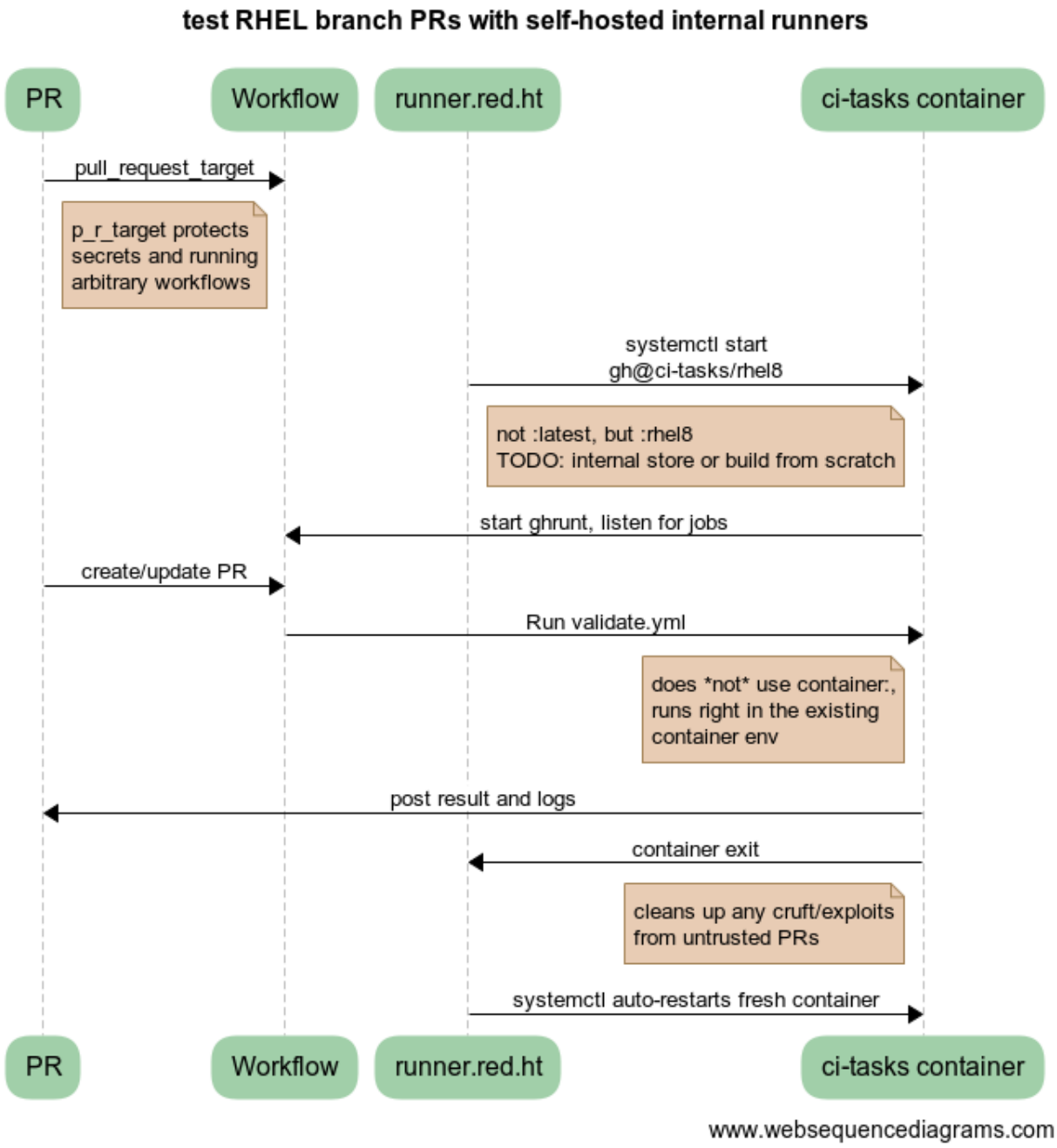
19.6.2 Manual container build

Just go to the actions tab in the Anaconda repository to the "Refresh container images" and press the `Run workflow` button on a button on a particular branch. Usually `master`, but for testing a change to the container you can push your branch to the origin repo and run it from there.

19.7 Security precautions for testing RHEL

19.7.1 Getting into our host/internal network

One of the main precautions is that each container test run has a limited time and is destroyed after timeout/end of test. That should narrow what attackers could do or how they can create a backdoor. See the image for more info:



Another hardening of this is potential issue is that only PRs approved by/created by users with permission to write are able to run the tests. To achieve this we have two ways how to start the test.

PR created by rhinstaller member – these are started from the RHEL branch workflow file by `pull_request_target` as usual. This workflow has two dependent jobs. First will check user privileges, second will run the tests in case the first one succeeded.

PR created by external contributors – these have to be started by workflow file validate-rhel-8.yml workflow from the `master` branch checking all the comments. If comment starts with `/test` phrase it will check the owner of the comment. When everything succeed it will set progress on the pull request originating the comment and start the tests. This progress is updated based on the result of the tests. As explained above, the whole implementation of the workflow is in the `master` branch which could be pretty confusing.

19.7.2 Changing workflow file by attacker

Because test description is part of the repository, attackers may change workflow files by creating PR to do their malicious attack. Because of that we are using `pull_request_target` instead of `pull_request` trigger. The main difference is that `pull_request_target` will run your PR tests on the target branch not on your PR branch. So workflow configuration has to be merged first to apply workflow changes. This has to be set on all workflow files in all branches, otherwise attackers could change existing workflow files to use our runners even for branches where they are not normally used. Unfortunately, self-hosted runners can't be bound to the branch, they are bound to the repo.

19.7.3 How can I change the workflow

Due to our hardening it's not possible to just create PR and see the result of your change on the PR checks tab. You have to create PR on your fork branch which has the updated workflow. I would recommend you to create a test organization for this and avoid creating a new account.

Similar situation works even for workflow to automatically update our containers. This workflow has `schedule` and `manual_dispatch` triggers. `schedule` triggers are always run on the default branch. For testing updates, always add `manual_dispatch` so that you can run them from your branch (on either origin or your fork).

19.8 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);
- *nosetests/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;
- *rpm_tests/* - basic RPM sanity test. Checks if anaconda.rpm can be installed in a temporary directory without failing dependencies or other RPM issues and checks if all files are correctly present in the RPM;
- *lib/* - helper modules used during testing;
- *nosetests/pyanaconda_tests/* - unit tests for the `pyanaconda` module;
- *pylint/* - checks the validity of Python source code using the *pocketlint* tool;
- *nosetests/regex_tests/* - Python unit tests for regular expressions defined in `pyanaconda.regexes`;

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!
