Koji Documentation

Release 1.17.0

Mike McLean, Mike B, Dennis Gilmore, Mathieu Bridon, Ian McLeo

Apr 01, 2019

CONTENTS

CHAPTER

ONE

KOJI

Koji is the software that builds RPM packages for the Fedora project. It uses Mock to create chroot environments to perform builds. To download the source code, report bugs, join the mailing list etc., see the Koji project website.

CHAPTER

TWO

CONTENTS

2.1 Koji HOWTO

2.1.1 Introduction

Koji is a system for building and tracking RPMs. It was designed with the following features in mind:

Security

- New buildroot for each build
- nfs is used (mostly) read-only

Leverage other software

- Uses Yum and Mock open-source components
- XML-RPC APIs for easy integration with other tools

Flexibility

- rich data model
- active code base

Usability

- Web interface with Kerberos authentication
- Thin, portable client
- Users can create local buildroots

Reproducibility

- Buildroot contents are tracked in the database
- Versioned data

This HOWTO document covers the basic tasks that a developer needs to be able to accomplish with Koji.

2.1.2 Getting started

The web interface

The primary interface for viewing Koji data is a web application. Most of the interface is read-only, but if you are logged in (see below) and have sufficient privileges there are some actions that can be performed though the web. For example:

- Cancel a build
- Resubmit a failed task

Those with admin privileges will find additional actions, such as:

- Create/Edit/Delete a tag
- Create/Edit/Delete a target
- Enable/Disable a build host

The web site utilizes Kerberos authentication. In order to log in you will need a valid Kerberos ticket and your web browser will need to be configured to send the Kerberos information to the server.

In Firefox, you will need to use the about:config page to set a Kerberos parameter. Use the search term 'negotiate' to filter the list. Change network.negotiate-auth.trusted-uris to the domain you want to authenticate against, e.g .example.com. You can leave network.negotiate-auth.delegation-uris blank, as it enables Kerberos ticket passing, which is not required.

In order to obtain a Kerberos ticket, use the kinit command.

Installing the Koji cli

There is a single point of entry for most operations. The command is called 'koji' and is included in the main koji package.

Repos/webpage TBD

The koji tool authenticates to the central server using Kerberos, so you will need to have a valid Kerberos ticket to use many features. However, many of the read-only commands will work without authentication.

Building a package

Builds are initiated with the command line tool. To build a package, the syntax is:

```
$ koji build <build target> <git URL>
```

For example:

```
$ koji build f25 git://pkgs.fedoraproject.org/rpms/eclipse-jgit?
↪#00ca55985303b1ce19c632922ebcca283ab6e296
```

The `koji build` command creates a build task in Koji. By default the tool will wait and print status updates until the build completes. You can override this with the `--nowait` option. To view other options to the build command use the `--help` option.

```
$ koji build --help
```

Build Options

There are a few options to the build command. Here are some more detailed explanations of them:

`--skip-tag` Normally the package is tagged after the build completes. This option causes the tagging step to be skipped. The package will be in the system, but untagged (you can later tag it with the tag-build command)

--scratch This makes the build into a scratch build. The build will not be imported into the db, it will just be built. The rpms will land under <topdir>/scratch. Scratch builds are not tracked and can never be tagged, but can be convenient for testing. Scratch builds are typically removed from the filesystem after one week.

--nowait As stated above, this prevents the cli from waiting on the build task.

--arch-override This option allows you to override the base set of arches to build for. This option is really only for testing during the beta period, but it may be retained for scratch builds in the future.

Build Failures

If your package fails to build, you will see something like this.

```
420066 buildArch (kernel-2.6.18-1.2739.10.9.el5.jjf.215394.2.src.rpm,
ia64): open (build-1.example.com) -> FAILED: BuildrootError:
error building package (arch ia64), mock exited with status 10
```

You can figure out why the build failed by looking at the log files. If there is a build.log, start there. Otherwise, look at init.log

```
$ ls -1 <topdir>/work/tasks/420066/*
<topdir>/work/tasks/420066/build.log
<topdir>/work/tasks/420066/init.log
<topdir>/work/tasks/420066/mockconfig.log
<topdir>/work/tasks/420066/root.log
```

Filing Bugs

bug tracking TBD

2.1.3 Koji Architecture

Terminology

In Koji, it is sometimes necessary to distinguish between the a package in general, a specific build of a package, and the various rpm files created by a build. When precision is needed, these terms should be interpreted as follows:

Package The name of a source rpm. This refers to the package in general and not any particular build or subpackage. For example: kernel, glibc, etc.

Build A particular build of a package. This refers to the entire build: all arches and subpackages. For example: kernel-2.6.9-34.EL, glibc-2.3.4-2.19.

RPM A particular rpm. A specific arch and subpackage of a build. For example: kernel-2.6.9-34.EL.x86_64, kernel-devel-2.6.9-34.EL.s390, glibc-2.3.4-2.19.i686, glibc-common-2.3.4-2.19.ia64

Koji Components

Koji is comprised of several components:

- **koji-hub** is the center of all Koji operations. It is an XML-RPC server running under mod_python in Apache. koji-hub is passive in that it only receives XML-RPC calls and relies upon the build daemons and other components to initiate communication. koji-hub is the only component that has direct access to the database and is one of the two components that have write access to the file system.

- **kojid** is the build daemon that runs on each of the build machines. Its primary responsibility is polling for incoming build requests and handling them accordingly. Koji also has support for tasks other than building. Creating install images is one example. kojid is responsible for handling these tasks as well.

  kojid uses mock for building. It also creates a fresh buildroot for every build. kojid is written in Python and communicates with koji-hub via XML-RPC.

- **koji-web** is a set of scripts that run in mod_python and use the Cheetah templating engine to provide an web interface to Koji. koji-web exposes a lot of information and also provides a means for certain operations, such as cancelling builds.

- **koji** is a CLI written in Python that provides many hooks into Koji. It allows the user to query much of the data as well as perform actions such as build initiation.

- **kojirepod** is a daemon that keeps the build root repodata updated.

Package Organization

Tags and Targets

Koji organizes packages using tags. In Koji a tag is roughly analogous to a beehive collection instance, but differ in a number of ways:

- Tags are tracked in the database but not on disk
- Tags support multiple inheritance
- Each tag has its own list of valid packages (inheritable)
- Package ownership can be set per-tag (inheritable)
- Tag inheritance is more configurable
- When you build you specify a *target* rather than a tag

A build target specifies where a package should be built and how it should be tagged afterwards. This allows target names to remain fixed as tags change through releases. You can get a full list of build targets with the following command:

```
$ koji list-targets
```

You can see just a single target with the `--name` option:

```
$ koji list-targets --name dist-fc7
Name                           Buildroot                      Destination
--------------------------------------------------------------------------------
↪-------
dist-fc7                       dist-fc7-build                 dist-fc7
```

This tells you a build for target dist-fc7 will use a buildroot with packages from the tag dist-fc7-build and tag the resulting packages as dist-fc7.

You can get a list of tags with the following command:

```
$ koji list-tags
```

Package lists

As mentioned above, each tag has its own list of packages that may be placed in the tag. To see that list for a tag, use the `list-pkgs` command:

```
$ koji list-pkgs --tag dist-fc7
Package                 Tag                     Extra Arches     Owner
----------------------- ----------------------- ---------------- ----------------
ElectricFence           dist-fc6                                 pmachata
GConf2                  dist-fc6                                 rstrode
lucene                  dist-fc6                                 dbhole
lvm2                    dist-fc6                                 lvm-team
ImageMagick             dist-fc6                                 nmurray
m17n-db                 dist-fc6                                 majain
m17n-lib                dist-fc6                                 majain
MAKEDEV                 dist-fc6                                 clumens
...
```

The first column is the name of the package, the second tells you which tag the package entry has been inherited from, and the third tells you the owner of the package.

Latest Builds

To see the latest builds for a tag, use the `latest-pkg` command:

```
$ koji latest-pkg --all dist-fc7
Build                                   Tag                   Built by
--------------------------------------  --------------------  ----------------
ConsoleKit-0.1.0-5.fc7                  dist-fc7              davidz
ElectricFence-2.2.2-20.2.2              dist-fc6              jkeating
GConf2-2.16.0-6.fc7                     dist-fc7              mclasen
ImageMagick-6.2.8.0-3.fc6.1             dist-fc6-updates      nmurray
MAKEDEV-3.23-1.2                        dist-fc6              nalin
MySQL-python-1.2.1_p2-2                 dist-fc7              katzj
NetworkManager-0.6.5-0.3.cvs20061025.fc7  dist-fc7            caillon
ORBit2-2.14.6-1.fc7                     dist-fc7              mclasen
```

The output gives you not only the latest builds, but which tag they have been inherited from and who built them (note: for builds imported from beehive the "built by" field may be misleading)

Exploring Koji

We've tried to make Koji self-documenting wherever possible. The command line tool will print a list of valid commands and each command supports `--help`. For example:

```
$ koji help
Koji commands are:
        build                Build a package from source
        cancel-task          Cancel a task
        help                 List available commands
        latest-build         Print the latest rpms for a tag
        latest-pkg           Print the latest builds for a tag
...
$ koji build --help
usage: koji build [options] tag URL
(Specify the --help global option for a list of other help options)

options:
  -h, --help            show this help message and exit
  --skip-tag            Do not attempt to tag package
  --scratch             Perform a scratch build
```

(continues on next page)

(continued from previous page)

```
  --nowait              Don't wait on build
...
```

2.1.4 Getting Involved

If you would like to be more involved with the Koji project...

Project data TBD

2.2 Defining Hub Policies

Defining a policy on the hub allows you fine control over certain activities in the system. At present, policy allows you to control:

- tag/untag/move operations
- allowing builds from srpm
- allowing builds from expired repos
- managing the package list for a tag
- managing which channel a task goes to

In the future, we expect to add more policy hooks for controlling more aspects of the system.

Policy configuration is optional. If you don't define one, then by default:

- tag/untag/move operations are governed by tag locks/permissions
- builds from srpm are only allowed for admins
- builds from expired repos are only allowed for admins
- only admins may modify package lists
- tasks go to the default channel

2.2.1 Configuration

The hub policy is configured in the `hub.conf` file, which is an ini-style configuration file. Policies are defined in the section named `[policy]`. Each `name = value` pair defines the policy of that name. With multiple line policies, successive lines should be indented so that the parser treats them as part of the whole.

Consider the following simple (and strict) example:

```
[policy]
tag =
    has_perm admin :: allow
    tag *-candidate :: allow
    all :: deny
```

This policy section defines a single policy (named 'tag'). The policy is a series of rules, one per line. The rule lines must be indented. Each rule is a test and an action, separated by a double colon. The valid actions for current policies are 'allow' and 'deny'. There are many tests available, though not all of them are applicable for all policies. Each test is specified by giving the name of the test followed by any arguments the test accepts.

Each rule in the policy is checked until a match is found. Upon finding a match, the action is applied. Our example above limits non-admins to tags ending in -candidate.

Getting a bit more complicated The example above is very simple. The policy syntax also supports compound tests, negated tests, and nested tests. Consider the following example:

```
[policy]
tag =
    buildtag *epel* :: {
        tag *epel* !! deny
    }
    tag *-updates :: {
        operation move :: {
            fromtag *-updates-candidate :: allow
            fromtag *-updates-testing :: allow
            all :: deny Tagging from some tags to *-updates is forbidden.
        }
        operation tag && hastag *-updates-candidate *-updates-testing :: deny
    }
    all :: allow
```

This policy sets up some rules concerning tags ending in -updates and tags containing epel, but is otherwise permissive.

The first nested rule limits builds built from a tag matching `epel` to only such tags. Note the use of !! instead of :: negates the test.

For tags matching `*-updates`, a particular work-flow is enforced. Moving is only allowed if the move is coming from a tag matching `*-updates-candidate` or `*-updates-testing`. Conversely, a basic tag operation (not a move) is denied if the build also has such a tag (the policy requires a move instead).

For denied operations some clarifying message is sent to user. If there is no specific message (everything after action keyword), only generic 'policy violation (policy_name)' is sent, so it could be helpful to specify such messages in more complicated cases.

2.2.2 General format

The general form of a basic policy line is one of the following

```
test [params] [&& test [params] ...] :: action-if-true
test [params] [&& test [params] ...] !! action-if-false
```

And for nested rules:

```
test [params] [&& ...] [::|!!] {
    test [params] [&& ...] [::|!!] action
    test [params] [&& ...] [::|!!] {
        ...
        }
}
```

Note that each closing brace must be on a line by itself. Using `!!` instead of `::` negates the entire test. Tests can only be joined with &&, the syntax does not support `||`.

2.2.3 Available policies

The system currently looks for the following policies

- tag : checked during tag/untag/move operations
- build_from_srpm : checked when a build from srpm (not an SCM reference) is requested.
- build_from_repo_id : checked when a build from a specified repo id is requested
- package_list : checked when the package list for a tag is modified
- channel : consulted when a task is created
- cg_import : consulted during content generator imports
- volume : determine which volume a build should live on

These policies are set by assigning a rule set to the given name in the policy section.

Note that the use of tag policies does not bypass tag locks or permissions

Note that an admin can bypass the tag policy by using `--force`.

2.2.4 Actions

Most of the policies are simply allow/deny policies. They have two possible actions: `allow` or `deny`.

The channel policy is used to determine the channel for a task. It supports the following actions:

use <channel>

- use the given channel

req

- use the requested channel
- generally this means the default, though some calls allow the client to request a channel

parent

- use the parent's channel
- only valid for child tasks
- recommend using the `is_child_task` test to be sure

2.2.5 Available tests

true

- always true. no arguments

all

- an alias of true

false

- always false. no arguments

none

- an alias of false

operation

- for tag operations, the operation is one of: tag, untag, move. This test checks its arguments against the name of the operation and returns true if there is a match. Accepts glob patterns.

- only applicable to the tag policy

package

- Matches its arguments against the package name. Accepts glob patterns.

tag

- matches its arguments against the tag name. Accepts glob patterns.
- for move operations, the tag name tested is the destination tag (see fromtag)
- for untag operations, the tag name is null and this test will always be false (see fromtag)
- for the build_from_* policies, tests the destination tag for the build (which will be null is –skip-tag is used)

fromtag

- matches against the tag name that a build is leaving. Accepts glob patterns
- for tag operations, the tag name is null and this test will always be false
- for move operations, the tag name test is the one that the build is moving from
- for untag operations, tests the tag the build is being removed from
- only applicable to the tag policy

hastag

- checks the current tags for the build in question against the arguments.

buildtag

- checks the build tag name against the arguments
- for the build_from_* policies the build tag is determined by the build target requested
- for the tag policies, determines the build tag from the build data, which will by null for imported builds

skip_tag

- checks to see if the –skip-tag option was used
- only applicable to the build_from_* policies

imported

- checks to see if the build in question was imported
- takes no arguments
- true if any of the component rpms in the build lacks buildroot data
- only applicable to the tag policy

is_build_owner

- Check if requesting user owns the build (not the same as package ownership)
- take no arguments

user_in_group

- matches the users groups against the arguments
- true if user is in /any/ matching group

has_perm

- matches the user's permissions against the arguments

- true is user has /any/ matching permission

source

- test the build source against the arguments
- for the build_from_* policies, this is the source specified for the build
- for the tag policy, this comes from the task corresponding to the build (and will be null for imported builds)

policy

- takes a single argument, which is the name of another policy to check
- checks the named policy. true if the resulting action is one of: yes, true, allow
- additional policies are defined in the [policy] section, just like the others

is_new_package

- true if the package being added is new to the system
- intended for use with the package_list policy

is_child_task

- true if the task is a child task
- for use with the channel policy

method

- matches the task method name against glob pattern(s)
- true if the method name matches any of the patterns
- for use with the channel policy

user

- checks the username against glob patterns
- true if any pattern matches
- the user matched is the user performing the action

match

- matches a field in the data against glob patterns
- true if any pattern matches

2.3 External Repository Server Bootstrap

2.3.1 Bootstrapping a new External Repo Koji build environment

These are the steps involved in pointing a new Koji server at external repositories so that it can be used for building. This assumes that the Koji hub is up, appropriate authentication methods have been configured, the Koji repo administration daemon (`kojira`) is properly configured and running, and at least one Koji builder (`kojid`) is properly configured and running. All koji cli commands assume that the user is a Koji `admin`. If you need help with these tasks, see the *Server How To*.

- Create a new tag.

```
$ koji add-tag dist-foo
```

- Create a build tag with the desired arches, and the previously created tag as a parent.

```
$ koji add-tag --parent dist-foo --arches "i386 x86_64 ppc ppc64" dist-foo-build
```

- Add an external repository to your build tag. Koji substitutes $arch for the arches in your build tag. The `$` needs to be escaped for the shell to send it though without interpreting it. See below for some additional examples of external repo URLs.

```
$ koji add-external-repo -t dist-foo-build dist-foo-external-repo http://repo-
↪server.example.com/path/to/repo/for/foo/\$arch/
```

Note: If you are adding multiple external repos, koji assigns a priority to each repo in FIFO order. This may cause updated packages to not be visible if a repo with older packages is ranked at a higher priority (lower numeric value). Use the `-p` flag to set specific repo priorities.

Note: This uses $arch NOT $basearch

- Create a build target that includes the tags you've already created.

```
$ koji add-target dist-foo dist-foo-build
```

At this point you can verify that your external repository is set with the "taginfo" command. You should see it listed under "External repos". Here is an example with several CentOS external repos:

```
$ koji taginfo dist-foo-build
  Tag: dist-foo-build [740]
  Arches: x86_64
  Groups: build, srpm-build
  Tag options:
  This tag is a buildroot for one or more targets
  Current repo: repo#55077: 2017-11-29 03:34:18.847127
  Targets that build from this tag:
    dist-foo
  External repos:
      2 centos7-cr (http://mirror.centos.org/centos/7/cr/$arch/)
      3 centos7-extras (http://mirror.centos.org/centos/7/extras/$arch/)
      5 centos7-updates (http://mirror.centos.org/centos/7/updates/$arch/)
     10 centos7-os (http://mirror.centos.org/centos/7/os/$arch/)
```

- Create a ''build" and ''srpm-build" group associated with your build tag.

```
$ koji add-group dist-foo-build build
$ koji add-group dist-foo-build srpm-build
```

- Populate the `build` and `srpm-build` group with packages that will be installed into the minimal buildroot. You can find out what the is in the current groups for Fedora by running `koji list-groups dist-f9-build` against the Fedora Koji instance. This is probably a good starting point for your minimal buildroot and srpm creation buildroot. If you are rebuilding Fedora packages, it is recommended that you add all packages listed in the Fedora Koji instance.

```
$ koji add-group-pkg dist-foo-build build <pkg1> <pkg2> .....
```

- Add packages that you intend to build to your tag.

```
$ koji add-pkg --owner <kojiuser> dist-foo <pkg1> <pkg2> .....
```

- Wait for the repo to regenerate, and you should now be able to run a build successfully.

2.3.2 Regenerating your repo

koji doesn't monitor external repositories for changes. new repositories will be generated when packages you build land in a tag that populates the buildroot or you manually regenerate the repository. you should be sure to regularly regenerate the repositories manually to pick up updates.

```
$ koji regen-repo dist-foo-build
```

2.3.3 Examples of urls to use for external Repositories

all these examples use mirrors.kernel.org please find the closest mirror to yourself. Note that the Fedora minimal buildroots download ~100Mb then build dependencies on top. these are downloaded each build you can save a lot of network bandwidth by using a local mirror or running through a caching proxy.

NOTE: this uses $arch **NOT** $basearch

Fedora 10

```
https://mirrors.kernel.org/fedora/releases/10/Everything/\$arch/os/
https://mirrors.kernel.org/fedora/updates/10/\$arch/
```

CentOS 5 and EPEL

```
https://mirrors.kernel.org/centos/5/os/\$arch/
https://mirrors.kernel.org/centos/5/updates/\$arch/
https://mirrors.kernel.org/fedora-epel/5/\$arch/
```

2.3.4 Example tags and targets

In the simplest setup, where you just want to build against what is available in the external repositories, you may want to go with a simple layout of *dist-f***X*-build* tags inheriting one another, and *dist-f***X*-updates* tags and targets that inherit the *dist-f***X*-build* tag and have external repos attached to them. This way, a *dist-f***Y*-build* or *dist-f***Y*-updates* tag will not automatically inherit the external repos of your *dist-f***X*** tags.

Tags

```
dist-f10-updates                - This is where the external repos for f10 release and␣
↪f10 updates are attached
 `- dist-f10-build              - This is the f10 build target with the 'build' and
↪'srpm-build' group inherited from dist-f9-build,
     |                            so that your buildroot gets populated but you do not␣
↪have to maintain these groups for each
     |                            separate release.
     `- dist-f9-build           - etc.
         `- dist-f8-build       - etc.
```

Targets

Each *dist-f**X*-build* tag has a *dist-f**X*-updates* child tag, and each *dist-f**X*-updates* tag has a corresponding *dist-f**X*-updates-candidate* build target.

2.4 Building Images in Koji

2.4.1 Image Building

Koji is capable of building many different types of images or appliances. They broadly fall into a few categories: LiveCDs, Disk Images, and Containers. All types of images end up with a Name-Version-Release just like an RPM build. What you need to provide Koji to perform a build varies by category, and the specifics will be explained below.

For additional questions and information, ask around in `#koji` on FreeNode IRC and sign up to the right mailing lists as the Koji project website dictates.

2.4.2 Kickstart First

No matter which type of image you want to build, you need to feed Koji a kickstart file, so we touch on how to create them and how to get them to Koji first. Once you have a kickstart to try out, look over the kickstart-specific caveats for each image type in the later sections.

Kickstart Reference

If you're new to automated installations, you should read the Anaconda Kickstart Guide first.

Getting your Kickstart to Koji

For scratch builds, you can get away with just using the `--kickstart` parameter, which accepts a path (relative or absolute) to your kickstart file on disk. For non-scratch builds though, the kickstart file needs to live in a remote Source Control Manager (SCM) such as git or Subversion. To access that, you need to pass the `--ksurl` parameter, which accepts a wacky string in this form:

```
git://git.fedorahosted.org/git/spin-kickstarts.git?fedora22#68c40eb7
```

Here we pointed to a repository hosted at `git://git.fedorahosted.org/git/spin-kickstarts.git`. Note the `?` in there, this delimiter separates the host URL with a directory structure to look in for the kickstart file. In other words, if you were to clone the repository, you should expect to see these (sub)directories laid out below the root of the clone. The important thing to realize is the behavior of `--kickstart` changes if `--ksurl` is specified.

Instead of indicating a path to a kickstart file, you just indicate a filename. The file itself is not specified in the fragment of the string between `?` and `#` that is passed to `--ksurl`, Koji still uses the `--kickstart` option for that. After the directory structure comes a `#`, which indicates the start of the commit ID. This is the commit Koji will reset the repository to after cloning. If you created a git repository which had just the spec file template in it, your SCM URL would have the `?` and `#` right next to each other.

If we passed `--ksurl` the string above, and gave `--kickstart "server-ec2.ks"`, then Koji would do the following:

- Clone the repository locally
- Call `git reset --hard 68c40eb7`
- Search in the "fedora22" subdirectory of the clone for a file called `server-ec2.ks`, and pass that to Anaconda
- Perform an automated install

2.4.3 Building LiveCDs

Let's describe building LiveCDs and the mechanics behind it.

Getting Started

What you need before proceeding:

- a name for your LiveCD, and a version number
- a Koji build target
- what architectures you want
- a flattened kickstart file
- you may also need yum repositories generated by release engineering if you require signed packages be installed into the LiveCD
- the livecd permission in Koji

The Koji command that creates a LiveCD is called `spin-livecd`. It calls out to livecd-tools in a mock chroot to construct the LiveCD. To run a LiveCD build, use the spin-livecd command like so:

```
$ koji spin-livecd --release 4 fedora-workstation 23 f23-image-build fedora-
↪workstation.ks
```

In this example a LiveCD will be created with the N-V-R of fedora-workstation-23-4. If `--release` was not included an incrementing value would be computed by Koji instead. spin-livecd takes a minimum of 5 arguments:

Name the name of the image, without versioning information. Examples could be `fedora` or `fedora-workstation`.

Version an arbitrary version string. For Fedora images, this usually matches the release version, such as 21, 22, or 23.

Build Target just like RPM builds Koji must be told which target to use. This controls what tag the image will be tagged into, and what packages are available to install into the image.

Architecture only x86_64 or i386 is supported

Kickstart File this is a recipe file that performs drives the construction of the image.

Use `--help` to discover more options to `spin-livecd`. You can override the Release field with `--release`, or `--repo` to override what yum repo is used to provide packages to install to the image. Note that regardless of what repo you use, it must contain RPMs built by Koji, otherwise you will get an error. (unless you are building a scratch image)

Mechanics

The way LiveCDs are created in Koji is fairly straightforward. A chroot is initialized and populated with the packages and their dependencies from the `livecd-build` package group. Next, the kickstart file is copied into it if it was provided from local storage. If not, it is checked out into it from an SCM. It is then modified to use the repo associated with the build tag for the target specified in the command unless the `--repo` option was given. Both the original and the modified kickstart files are saved as part of the output for the task for later review. A livecd-creator command is executed using the mock('–chroot', ...) method.

Note: This process runs as root. This produces the desired image which is uploaded to /mnt/koji/images/<image>/$imageID if it is not a scratch image.

Caveats for LiveCDs

There are some known caveats with using the spin-livecd command that users should be aware of.

%include macros in the kickstart

A word of caution about kickstart files and the `%include` macro. livecd-creator is smart enough to search the current directory of the submitted kickstart file if it has %include macros. If the kickstart specified to koji is from local storage, only that kickstart file will be copied into the chroot, and this creates a problem if it has `%include` macros, because the other kickstart files it needs will be inaccessible. This issue is not present when the kickstart file is retrieved from a remote SCM (such as the spin-kickstarts git repo), because the entire repository is checked out. Presumably it will include any other kickstart files the specified one is including in the same directory. A workaround for the issue would be to use `ksflatten` (from pykickstart) on kickstart files with `%include` macros that are going to be submitted to koji from the user's local disk.

Package Groups in the Kickstart File

Package Groups in the kickstart file cause a problem if the Koji repos do not define them, which they most likely don't since Koji's comps.xml is based on the "groups" set up from the CLI. livecd-creator's behavior is to ignore package groups that are not defined in the repo it is using, so this can be troublesome when creating the image since packages could be left out. There are a couple possible workarounds:

- do not use package groups in the kickstart file and just specify a huge list of packages
- use `--repo` and specify a repo that does have a comps.xml that defines the groups it uses

Only Include RPMs Built in Koji

The image building tasks will fail if your image tries to include a package that was not built in your build system. This is because the package does not have any origin information stored in Koji's database. The repos defined in the kickstart will automatically be overridden with the repo of the build tag for the build target, unless you use the

`--repo` option. Since only packages you have built (or include from an external repo) should be there, you should never have this problem unless you use `--repo`.

No Signed or Debuginfo RPMs in Koji's Build Tags

If you need signed RPMs or debuginfo RPMs, you will run into trouble because Koji does not keep those in its build tag repos. The only work around for this is to create a repo yourself that includes these RPMs and then use `--repo`. This will force the image to take RPMs from that repo. Remember, the task will fail if Koji detects RPMs were installed that were not built in the build system.

%post Section in Kickstart

While livecd-tools does support building on SELinux disabled hosts, you can run into denials when booting if you create and use new files in the `%post` section of your kickstart file. If you do, you should either set the labels appropriately at the end of the `%post` section, or instigate an autorelabel at boot time.

Troubleshooting

If your build fails, you will be notified on the command line. In the output will be a URL to the Koji UI, visit that and click on the red subtask link. From that page review `root.log` and `livecd.log` for errors. Often errors are caused by packages being missing, or malformed kickstart files. The log files are at the bottom of the page. If you're stuck, contact Release Engineering.

Build System Preparation

This section assumes you have know-how required to install and configure a new instance of Koji, and that you have already done so. You can learn how to do so *here* if you need to. Please ensure you are using the latest version of the software and that your database schema is updated as well. You should also have some familiarity with how livecd-creator works. This section only covers preparation for LiveCD builds.

Follow this procedure step by step to get things prepared they way they need to be.

1. **`koji add-host-to-channel <your-host> livecd`** add a builder to the livecd channel
2. **`koji grant-permission livecd <user>`** grant the permission to build an image type to a user. This step is optional since admins have all permissions.
3. You will need a tag and target to build the images from. The yum repo generated for the build tag of the target is what Koji will use to populate the LiveCDs with by default. (the alternative is to use the `--repo` option, more on that later)
4. `koji add-group <build-tag> livecd-build` add the livecd-build group
5. **`koji add-group-pkg <build-tag> livecd-build <pkg> ...`** add packages to the livecd-build group. These package lists vary has packages and dependencies change. As of October, 2015 for Fedora 23 the needed packages for each image type are:
    - bash, coreutils, fedora-logos, fedora-release, livecd-tools, policycoreutils, python-dbus, sed, selinux-policy-targeted, shadow-utils, squashfs-tools, sssd-client, tar, unzip, util-linux, which, yum

2.4.4 Building Disk Images

Disk images are files that represent virtual disks. They have a partition table and filesystems on them, and are available in a variety of formats: qcow2, vmdk, ova, Hyper-V, raw, "base" container images, and more.

Getting Started

What you need:

- a name for your LiveCD, and a version number
- what architectures you want
- a Koji build target
- kickstart file
- installation tree
- you may also need yum repositories generated by Rel-Eng if you require signed packages be installed into the image

The Koji command to build a disk image is called `image-build`. The `image-build` command uses ImageFactory and Oz to start a VM guest and perform an automated Anaconda installation. Here is a (lengthy) example for building a disk image.

```
$ koji image-build --repo 'https://alt.fedoraproject.org/pub/alt/releases/22/Cloud/
↪$arch/os/' --kickstart fedora-server.ks --scratch --distro Fedora-22 --format qcow2␣
↪fedora-server-kvm 22 'https://alt.fedoraproject.org/pub/alt/releases/22/Cloud/$arch/
↪os/' x86_64
```

This example builds a scratch qcow2 disk image using packages from an additional yum repository. Without this option the yum repo to populate the build root would be used instead. If this was the first image with the N-V of fedora-server-kvm-22, then the N-V-R would be fedora-server-kvm-22-1, because Koji uses an incrementing number for the release if you do not provide one. Like all Koji commands, use `--help` to see more options that are available.

For Docker, Koji only supports Base Images right now using a kickstart file as described above. In the future it will support layered images, but not before some Docker requirements are met, and Koji is maintaining a Registry of its own. This scoping effort is ongoing.

`image-build` takes a minimum of 5 positional arguments, and 2 options must be specified. They are reviewed in the list below, with the positional arguments first.

Name the name of the image, without versioning information. Examples could be `fedora-server` or `fedora-workstation`.

Version an arbitrary version string. For Fedora images, this usually matches the release version, such as 22 or 23.

Build Target just like RPM builds Koji must be told which target to use. This controls what tag the image will be tagged into, and what packages are available to install into the image.

Installation Tree URL this is a URL to a location you can install an operating system from. It is the same place you would direct a PXE-booted system to go. In 99% of cases this location is provided by Release Engineering. It should have an "isolinux" subdirectory and yum metadata somewhere within.

Architecture only x86_64 or i386 is supported, and you can specify both on the command line. This will cause two subtasks to be run, allowing you to build for both arches in parallel. If either fail, the whole build will fail.

Kickstart File this is a recipe file that performs drives the construction of the image. Pass in the path to a kickstart file, which must be flattened.

Kickstart URL in a non-scratch build, you'll need this too. For more details, see the Getting your Kickstart to Koji section.

Distro a string that indicates what OS is being built. These always follow the convention of "Fedora-X", where X is the release number.

Since this command can get very long, a configuration file can drive the task as well, using the `--config` option. It accepts a path to a configuration file written in the Python ConfigParser format (like a Windows .ini). The options are all named the same with one caveat, see below. Here's what one could look like:

```
[image-build]
name = fedora-server-docker
version = 22
target = f22-candidate
install_tree = https://alt.fedoraproject.org/pub/alt/releases/22/Cloud/$arch/os/
arches = x86_64

format = qcow2,rhevm-ova,vsphere-ova
distro = Fedora-22
repo = https://alt.fedoraproject.org/pub/alt/releases/22/Cloud/$arch/os/
disk_size = 20

ksversion = DEVEL
kickstart = fedora-22-server-docker.ks
ksurl = git://git.fedorahosted.org/git/spin-kickstarts.git?fedora22#68c40eb7
specfile = git://git.fedorahosted.org/git/spin-kickstarts.git?spec_templates/fedora22
↪#68c40eb7
```

A few notes on the syntax:

- it allows for comments too, the lines start with a hash (#)
- options on the command line that can be used multiple times can accept values here as comma-separated strings
- options with a hyphen need to use an underscore instead. `--disk-size` for example would be `disk_size` in the config file

OVA Features

If you're building OVAs, either for RHEVM or vSphere, you can specify OVA options with a special section in the configuration file. It looks something like this:

```
[ova-options]
vsphere_product_version=22
rhevm_description=Fedora Cloud 22
vsphere_product_vendor_name=Fedora Project
ovf_memory_mb=6144
rhevm_default_display_type=1
vsphere_product_name=Fedora Cloud 22
ovf_cpu_count=4
rhevm_os_descriptor=Fedora-22
```

or this:

```
[ova-options]
vsphere_ova_format = vagrant-virtualbox
rhevm_ova_format = vagrant-libvirt
vagrant_sync_directory = /home/vagrant/sync
```

The second one is actually the secret sauce for generating an image for use in Vagrant. At this time, you would need rename the image file extension from .ova to .box, but otherwise this should work fine.

Kickstart Preparation

Kickstarts for the image-build command have some specific requirements which are covered in this section.

Required Kickstart Arguments

Anaconda of course requires many commands to be defined in the kickstart file. If you're starting from scratch you should review the reference linked above, or use an existing kickstart file in the spin-kickstarts git repo. It is critically important that the installation be completely automated, if Anaconda has to prompt for input for any reason, the build will fail because you cannot send input to the guest. Some of the kickstart commands are optional to Anaconda, but are required in Koji for your build to succeed. Here's the list and the reasons why.

zerombr You must tell Anaconda to wipe out the MBR in the virtual block device, if you don't Anaconda will ask you.

clearpart –all –initlabel Anaconda has to be told to wipe out all data on the virtual block device we install on otherwise it will ask for confirmation to do so. Since it is blank anyway this is harmless.

reboot When the installation completes, the guest is rebooted. ImageFactory is specifically looking for this behavior to conclude the installation completed. Anaconda's default behavior is to wait for a key press to reboot the system, but this is impossible from outside of Koji.

locking the root account You have to lock the root account (rootpw –lock) or create a non-root user (user), otherwise Anaconda will prompt for one.

Do not use the url command The repo commands are overridden by Koji to point to internal Koji repos, or what you specified on the command line with `--repo`, it does not override the url command if you provided it. Anaconda has a behavior where it will prefer packages from the repositories given with the url command over those with the repo command, and this is generally not what you want. If Koji sees an RPM was installed that was not built in the system, it will fail the build.

Recommended Kickstart Arguments

Often you want a `%post` section in your kickstart to perform post-installation configuration steps. Review that section of the reference and note that you can specify `--log` and `--interpreter`. Both of these are recommended (but not required) to assist with the development and debugging process. Here are some other recommendations:

- You probably want the network to use dhcp, sshd to be started, and port 22 opened in the firewall to allow access as well.
- If you're building an image that will be shipped with a product, SELinux should be enabled.
- Images that will be used in cloud deployments like OpenStack or EC2 should have `cloud-init` in the package list.
- It is discouraged to have root passwords in plaintext in your kickstart file.
- If your %post section is written in bash, consider setting -x.
- For images that have multiple partitions, use the `--asprimary` option for the part command that defines the root file system. This will ensure it is the first partition on the image, which is a requirement in some cloud environments like EC2.

Troubleshooting

If your disk image build fails, follow the link in the command line output that takes you to the task page in the Koji web UI. Click on the failed createImage subtask in red. On that page review the screenshot.ppm file if it was provided, or oz.log. Most failures are from Anaconda rejecting a malformed kickstart file, which will be indicated in the screenshot. Your installation must be completely automatic, there can be no interactivity at all, otherwise Anaconda will sit there indefinitely until Koji (actually ImageFactory) kills the task.

It is very easy to write a kickstart file with bugs or that results in a system that does not boot. This section will present a series of questions to ask yourself and examples to help diagnose where the problem lies. Once you know that, it should be easier to understand what you can do to inspect further.

There are 4 steps in the process:

1. create a guest
2. perform an automated installation in the guest
3. boot the guest and extract the list of installed RPMs
4. upload and archive the disk image of the guest

Is it a problem with guest creation?

There have been unusual cases where libvirt, ImageFactory, or Oz was misconfigured and guests could not be started properly. A misconfiguration with Puppet or whatever Fedora Infrastructure is up to can cause this. So far the errors have been clear in the task output, look either in the results string or oz.log. The bad news is that in this case you really can only inform Rel-Eng about the issue and wait for a resolution. The good news is these cases are very rare.

Did the installation fail?

The Anaconda installation can fail for many reasons: missing packages, network problems, or syntax errors in %post. Tasks will also fail if Anaconda prompts for input for any reason. If Koji detects a lack of disk activity in the guest for more than 5 minutes, it will fail the build and tear down the guest. Looking in oz.log may have the answer: dracut, anaconda, and yum logs are all printed there.

These sorts of failures often have a screenshot taken and saved with the task output called screenshot.ppm. Viewing this will usually tell you what Anaconda is complaining about if the installer detected an issue or prompted for input. The string in the results output that says "No disk activity in 300 seconds, failing." This almost always means Anaconda hit an issue and either gave up or waited.

If Anaconda claims it is missing packages, confirm they exist in the repos you are using with `--repo`, if you are using that option. If you are not, confirm the builds you expect are in the tag inheritance for the target you are running. This is a lot like checking whether an RPM will build against the right libraries, except we're building an image instead.

If you get the rare Anaconda dialog box that says something like "An unexpected error occurred", try using the `text` command in kickstart, which will have Anaconda boot in text mode. Sometimes the Python traceback (or whatever the error condition is) will be printed there. I have also seen cases where text-mode yields a black screen, but booting in graphical mode (the default) does produce a useful dialog box. Issues like this stem from syntax errors in the kickstart file, or bugs in pykickstart itself. If you think it is a pykickstart bug, then someone in Rel-Eng needs to update pykickstart on the builders.

Did the guest boot?

Koji waits 5 minutes for a guest to boot in this step. It unfortunately does not give a lot of insight to why a guest may not boot, so these are a tougher class of issues to work through. You can usually answer this question by looking in

results string. If you see "Timed out waiting for guest to boot", then this is your problem. You can also confirm this in oz.log.

For now, the best way to investigate an issue like this is to drive a guest installation locally using something like Gnome's Virtual Machine Manager (VMM). The steps to perform are:

- Select a Network Install
- For the Operating System Install URL use the same one you gave to Koji. It will be something like https://alt.fedoraproject.org/pub/alt/releases/22/Cloud/x86_64/os/
- Set the Kickstart URL to where your kickstart file is. You may need to make it available over http.
- Bump the memory to 2048M for good measure
- Launch the guest and let it complete installation
- Open a VNC session and watch what happens when the guest attempts to boot.

If the console is not providing enough information, we have to get more creative. Anaconda supports starting an SSH daemon while the installation is happening with the sshpw command in kickstart. Set that and comment out the reboot command. This will let the installation complete locally and wait for a keystroke to reboot the guest. At this point you should be able to ssh in and inspect the environment to figure out what is going on. You should also consider making use of the `--log` option to %post so that output from the script is saved somewhere.

Another option would be to scp logs and other files off of the guest as part of the `%post` script.

Other Guest Misconfigurations

If the guest boots but you're having problems accessing it I'd suggest following same procedure as when the guest fails to boot. This could be a result of firewall misconfigurations or SSH not being available for some reason. Usually in this case the build is succeeding in Koji, but there's something still fundamentally broken in the image. If the issue is something you can investigate while the guest is online (you can log in), then I'd suggest importing it locally using the libvirt.xml and the disk image provided in Koji's task output.

You can also do investigative work in an offline mode by mounting the image locally or using something like libguestfs to poke around without starting the guest. The fast, dirty way to do it is by mounting it. This can often pollute your guest environment. Here's how to do it:

- Download the image from Koji
- If the image format is not raw, you have to convert it first with qemu-img. Something like:

```
$ qemu-img convert -O raw <image-file> <output-file></pre>
```

- Now mount it up using loopback devices. (as root) If your image has multiple partitions in it, you may need to pass in a different mapped loopback device like `loop0p2`. Whichever one you think is the root partition or has the issue you're trying to fix.

```
# kpartx -av <raw-image>
# mount -o loop /dev/mapper/loop0p1 /mnt/my_directory
```

Hopefully at this point you figure out the issue. To tear down the image you'll run commands as root like so:

```
# umount /mnt/my_directory
# dmsetup remove loop0p1
# losetup -d /dev/loop0
```

Again, if you used different loopback devices, substitute those in to the dmsetup and losetup commands.

Build System Preparation

Follow this guide if you're a Koji admin and would like to enable image building or want to set up some testing before enabling the integration.

When moving to ImageFactory to do image builds Koji lost the ability to easily reproduce the build environment for images the way we do for RPMs using Mock. This section will document how to set up an image builder for Koji. This is a lengthy task, it will take first-timers about a week to have a useful instance, and it is painful because it requires a bare metal system be provisioned since ImageFactory provisions VMs to build the image. There is a significant performance penalty for using nested virtualization.

Follow the steps below to set up your builder.

Note: You do not have to stand up a complete Koji instance to test the way Koji builds images. However, if you want to test image builds with an accurate representation of how Koji does it, or you want to test code changes related to image builds in Koji, you should follow all the steps below.

ImageFactory/Oz Preparation

1. Provision a system with at least 4G of memory with the current release of RHEL 6 or later. Sometimes builders lag behind a month or two before taking in updates, but this will still get you pretty close to where you want to be. For Fedora, use the latest release.
2. Install the following packages to the builder. #. oz #. imagefactory #. imagefactory-plugins-TinMan #. imagefactory-plugins-vSphere #. imagefactory-plugins-ovfcommon #. imagefactory-plugins-docker #. imagefactory-plugins #. imagefactory-plugins-OVA #. imagefactory-plugins-RHEVM #. python-psphere => 0.5 #. VMDKStream => 0.2 #. pykickstart
3. Edit `/etc/kojid/kojid.conf`, and set an second value, eg: 7200 for `oz_install_timeout`. It's a timeout waiting guest installing. Default value is 0, that means oz will use its default value. Since `oz-0.16.0`, it can be configured in `/etc/oz/oz.cfg` as `install` in `[timeouts]` section.
4. Edit `/etc/oz/oz.cfg`, and set the memory value in the `[libvirt]` section to at least 2048. Set `safe_generation` under `[icicle]` to yes.
5. Run: `mkdir -p ~root/.psphere/templates`, and then copy the following code into `~root/.psphere/config.yaml`. Do not worry about the server, username, and password credentials; they are not used anywhere.

```
general:
  server: 10.16.120.224
  username: Administrator
  password: whatever
  template_dir: ~/.psphere/templates/
logging:
  destination: ~/.psphere/psphere.log
  level: DEBUG # DEBUG, INFO, etc
```

Start up the services and ImageFactory/Oz should be ready to go. You should read more about how to use Oz and how to use ImageFactory. If you want to try calling ImageFactory as if from a Koji Builder (but not set up a whole Koji instance), you can use the code below to emulate that. If you want to test the Koji integration with a full Koji instance, proceed to the next section instead.

```
#!/usr/bin/python -tt
```

(continues on next page)

(continued from previous page)

```
import logging
import os.path
import random
import sys

from imgfac.BuildDispatcher import BuildDispatcher
from imgfac.PluginManager import PluginManager
from imgfac.ReservationManager import ReservationManager
plugin_mgr = PluginManager('/etc/imagefactory/plugins.d')
plugin_mgr.load()
from imgfac.ApplicationConfiguration import ApplicationConfiguration

# logging
handler = logging.StreamHandler(sys.stdout)
tlog = logging.getLogger()
tlog.setLevel(logging.DEBUG)
tlog.addHandler(handler)

# configuration
ks = open('oztest.ks').read()
workdir = '/tmp/koji/test'
config =  {
    #Oz specific
    'oz_data_dir': os.path.join(workdir, 'oz_data'),
    'oz_screenshot_dir': os.path.join(workdir, 'oz_screenshots'),
    #IF specific
    'imgdir': os.path.join(workdir, 'scratch_images'),
    'tmpdir': os.path.join(workdir, 'oz-tmp'),
    'verbose' : True,
    'timeout': 3600,
    'output': 'log',
    'raw': False,
    'debug': True,
    'image_manager': 'file',
    'plugins': '/etc/imagefactory/plugins.d',
    'tdl_require_root_pw': False,
    'image_manager_args': {
        'storage_path': os.path.join(workdir, 'output_image')},
}
random.seed() # necessary to ensure a unique mac address
rm = ReservationManager()
rm._listen_port = random.randint(rm.MIN_PORT, rm.MAX_PORT)
ApplicationConfiguration(configuration=config)
params = {'install_script': ks}
template = """<template>
    <name>test-appliance</name>
        <os>
            <name>Fedora</name>
            <version>22</version>
            <arch>x86_64</arch>
            <install type='url'>
                <url>https://alt.fedoraproject.org/pub/alt/releases/22/Cloud/x86_64/
↪os/</url>
            </install>
            <icicle>
                <extra_command>rpm -qa --qf '%{NAME},%{VERSION},%{RELEASE},%{ARCH},%
↪{EPOCH},%{SIZE},%{SIGMD5},%{BUILDTIME}\n'</extra_command>
```

(continues on next page)

(continued from previous page)

```
            </icicle>
        </os>
    <description>test-appliance OS</description>
    <disk>
        <size>16G</size>
    </disk>
</template>
"""
bd = BuildDispatcher()

# build the image
base = bd.builder_for_base_image(template, parameters=params)
base.base_thread.join()
tlog.removeHandler(handler)
```

This script is run as root with no arguments. It uses `oztest.ks` in your local directory as the kickstart you want to try to use. The URL in the XML template is where the RPM packages will be installed from, and what the guest will be booted with.

Koji Preparation

1. *Install Koji* if you need it. This will easily be the most time-consuming part of the process; my first time took 3 days to get it working properly. Follow the guide closely, and go with the SSL authentication method. SSL is a lot easier to set up locally. You will need to install every Koji component (except koji-vmd) on the same system. Proceed to the next step after you've had a successful Kojira repository generated.
2. At this point, you have a system that should be ready to build images. We just have to do some Koji configuration so that your instance is pulling content from Koji. Replace the base tag names with whatever fits your conventions.

   (a) Add tags, tag inheritance, new pkg (with pkg owner), new external repo, and regen the repo

   ```
   koji add-tag fedora22
   koji add-tag jay-fedora22
   koji add-tag jay-fedora22-override --parent jay-1-fedora22
   koji add-tag jay-fedora22-build --arches x86_64 --parent jay-fedora22-override
   koji add-tag jay-fedora22-candidate --parent jay-fedora22
   koji add-tag-inheritance --priority 40 jay-fedora22-build fedora22
   koji add-pkg --owner kojiadmin jay-fedora22 fedora-server-ec2 fedora-server-
   ↪kvm
   koji add-external-repo -t fedora::20 fedora22 'https://alt.fedoraproject.org/
   ↪pub/alt/releases/22/Cloud/$arch/os/'
   koji add-target jay-fedora22-candidate jay-fedora22-build
   koji regen-repo jay-fedora22-build
   ```

   (b) Grab a kickstart file from an image task in Koji that relates to what you want to test.

   (c) Finally, kick off a build!

   ```
   koji image-build fedora-server-ec2 22 --distro Fedora-22 jay-fedora22-
   ↪candidate --kickstart fedora-server-starter-ec2.ks 'https://alt.
   ↪fedoraproject.org/pub/alt/releases/22/Cloud/$arch/os/'
   ```

2.4.5 Building Appliances

This section is here for the sake of legacy. Unless you are trying to build ARM images, you should use the image-build command described in the previous section.

Note: The spin-appliance command, described herein, is deprecated.

Getting Started

Here's what you need before proceeding:

- a name for your Appliance, and a version number
- a Koji build target
- what architectures you want
- a flattened kickstart file
- you may also need yum repositories generated by release engineering if you require signed packages be installed into the appliance
- the appliance permission in Koji

The Koji command that creates an Appliance is called `spin-appliance`. It calls out to appliance-creator in a mock chroot to construct the Appliance. To run an Appliance build, use the spin-appliance command like so:

```
$ koji spin-appliance --release 4 fedora-workstation 23 f23-build fedora-workstation.
↪ks
```

In this example an Appliance will be created with the N-V-R of `fedora-workstation-23-4`. If `--release` was not included an incrementing value would be computed by Koji instead. spin-appliance takes a minimum of 5 arguments:

Name the name of the image, without versioning information. Examples could be `fedora` or `fedora-workstation`.

Version an arbitrary version string. For Fedora images, this usually matches the release version, such as 21, 22, or 23.

Build Target just like RPM builds Koji must be told which target to use. This controls what tag the image will be tagged into, and what packages are available to install into the image.

Architecture only arm, x86_64, or i386 are supported

Kickstart File this is a recipe file that performs drives the construction of the image.

Use `--help` to discover more options to spin-appliance. You can override the Release field with `--release`, or `--repo` to override what yum repo is used to provide packages to install to the image. Note that regardless of what repo you use, it must contain RPMs built by Koji, otherwise you will get an error. (unless you are building a scratch image)

Mechanics

The way Appliances are created in Koji is the same as *Building LiveCDs*.

Caveats for Appliances

Caveats for using `spin-appliance` are the same as using `spin-livecd` to *Caveats for LiveCDs*.

Troubleshooting

If your build fails, you will be notified on the command line. In the output will be a URL to the Koji UI, visit that and click on the red subtask link. From that page review `root.log` and `appliance.log` for errors. Often errors are caused by packages being missing, or malformed kickstart files. The log files are at the bottom of the page. If you're stuck, contact Release Engineering.

Build System Preparation

This section assumes you have know-how required to install and configure a new instance of Koji, and that you have already done so. You can learn how to do so *here* if you need to. Please ensure you are using the latest version of the software and that your database schema is updated as well. You should also have some familiarity with how appliance-creator works. This section only covers preparation for Appliance builds.

Follow this procedure step by step to get things prepared they way they need to be.

1. **Add a builder to the appliance channel**

   ```
   koji add-host-to-channel <your-host> appliance
   ```

2. Grant the permission to build an appliance to a user. This step is optional since admins have all permissions.

   ```
   koji grant-permission appliance <user>
   ```

3. You will need a tag and target to build the images from. The yum repo generated for the build tag of the target is what Koji will use to populate the Appliances with by default. (the alternative is to use the `--repo` option, more on that later)

4. Add the appliance-build group

   ```
   koji add-group <build-tag> appliance-build``
   ```

5. Add packages to the appliance-build group. These package lists vary has packages and dependencies change. As of October, 2015 for Fedora 24 the needed packages for appliances:

   - appliance-tools, bash, coreutils, grub, parted, perl, policycoreutils, selinux-policy, shadow-utils, sssd-client

   ```
   koji add-group-pkg <build-tag> appliance-build <pkg> ...
   ```

2.5 Koji Miscellaneous Notes

2.5.1 Notes and miscellaneous details about Koji

This document is intended to illuminate some of the small quirks that you may encounter while running your own koji server.

Koji CLI

Getting Help

- There are multiple ways to receive help from the koji command, each of them gives you something a little different:
  - List of user commands:

```
koji help
```

  - Command-specific help:

```
koji [command] --help
```

  - List of admin commands:

```
koji help --admin
```

Building from SRPM

Koji only allows admins access to build directly from srpm. It expects normal users to build out of an SCM.

Building from SCM

SCM Layout

When building out of an SCM, Koji expects there will be a Makefile in the project root that has a 'sources' target. Koji will call 'make sources' on the checked out files.

This target simply needs to download all the sources for the SRPM that are not already included in the SCM repository.

SCM URI Structure

In Fedora, this detail is typically handled by the fedpkg tool. Outside of Fedora, you may need to know how to manually submit builds from an SCM to koji.

Koji accepts an SCM URI in this format:

```
koji build [tag] [scheme]://[user]@[hostname]/[path/to/repository]?[path/to/project]
↪#[revision]
```

Note the division between repository path and project path. During setup of kojid, the allowed_scms parameter is configured in /etc/kojid/kojid.conf. This value of this parameter should match the path to the repository.

In some SCM configurations, there isn't a difference between repository path and project path. In these cases, it should be understood that dividing your SCM path into two components, URI path and URI query, can seem somewhat arbitrary. An easy way to remember this detail is that the path specified in allowed_scms is the portion of your SCM path that goes before the URI query, any sub-directories not specified in allowed_scms is given to koji as the URI query.

Koji tasks

BuildNotifications

- Koji sends build notifications to the package owner and the user who submitted the build. For BuildNotifications to work successfully, the package owner's username needs to match a valid username for an e-mail address, because kojihub sends to username@domain_in_kojihub.conf.
  - For SSL authentication, this means that your CN must be valid as the user portion of an e-mail address.

Koji server administration

Importing an rpm without srpm

If you are running a private koji instance, you may run into a situation where you need to integrate a proprietary rpm into your build system. To do this, it is similar to the package imports you did when bootstrapping your koji instance:

```
koji import --create-build [package]
```

Multi-arch builds

There are some packages that need to build across multiple arches. For example, the kernel package no longer builds an i386 kernel, kernels are built on i586 and above. To instruct koji to build these additional arches, use this command:

```
koji set-pkg-arches [options] <arches> <tag> <package> [<package2> ...]
```

Note: is a single entity, so denote multiple arches within quotes (e.g. 'i386 i586 i686').

Manually submitting a task

Occasionally, you may need to manually submit a task to koji. One likely usage is to manually trigger a createRepo. To do this, use this command:

```
koji make-task [options] <arg1> [<arg2>...]
```

The make-task command is a bit of under-documented black magic. Task parameters are defined in kojihub.py. The easiest way I have found to figure out the right incantations for make-task is to query the *task* table in the koji database directly. Find a similar task to the one you want to create, and look in the request field for the parameters the task used, and mimic those.

So, citing the createRepo case above, here is an example:

```
kojiadmin@localhost$ koji make-task --channel=createrepo --priority=15 \
newRepo dist-foo-build
```

Managing your tags

Occasionally an unwanted package or version of a package will be built by koji. Don't fret. There are two mechanisms to handle rescinding a package or specific package version.

- To remove a specific version of a package, you can untag it:

```
   koji untag-build [options] <tag> <pkg> [<pkg>...]

supports either %name or %name-%version-%release
```

- To remove all versions of a package, you can untag it as above or you can administratively block it from being listed in a tag:

```
koji block-pkg [options] tag package [package2 ...]
```

Spec file processing

Macro processing

Macros in the spec file are expanded before Requires and BuildRequires are processed. If there are any custom macros in the spec file, the package that drops those macros into /etc/rpm must be tagged under your dist-build tag

%dist tags

For packages that incorporate the %dist tags in their filename, they expect %dist to be defined in /etc/rpm/macros.dist, which was added in Fedora 7. For building on RHEL5/FC6 and earlier, koji needs the https://buildsys.fedoraproject.org/buildgroups/ buildsys-macros package tagged under the dist-build tag.

2.6 Release Notes

2.6.1 Koji 1.17.0 Release notes

Migrating from Koji 1.16

For details on migrating see *Migrating to Koji 1.17*

Security Fixes

CVE-2018-1002161 - SQL injection in multiple remote calls

PR: https://pagure.io/koji/pull-request/1274

This release includes the fix for *CVE-2018-1002161*

Client Changes

Volume id option for livemedia and livecd tasks

PR: https://pagure.io/koji/pull-request/1227

The `spin-livecd` and `spin-livemedia` commands now accept a `--volid` argument to specify the volume id for the media. If unspecified, the volume id is chosen via the same heuristic as before.

Volume ids must be 32 characters or less.

Build order preserved by clone-tag

PR: https://pagure.io/koji/pull-request/1014

This is an improvement to the `clone-tag` command. Previously, when the command was used without the `--latest-only` option, it could get the ordering of builds wrong in the destination tag. Now, the order will match the source tag.

Configurable authentication timeout

PR: https://pagure.io/koji/pull-request/1172

Previously, the network timeout during authentication was hard coded to 60 seconds. It is now configurable via the `auth_timeout` configuration option.

Additional information from list-channels command

PR: https://pagure.io/koji/pull-request/940

The `list-channels` command now shows three separate host counts for each channel:

- the number of enabled hosts in the channel
- the number of ready hosts in the channel
- the number of disabled hosts in the channel

The free-task command requires at least one task-id

PR: https://pagure.io/koji/pull-request/1045

Previously this command was a no-op when given no arguments. Now it will return an error.

Library Changes

Drop encode_int function

PR: https://pagure.io/koji/pull-request/852

This is a follow up to the large integer support that we added in version 1.14

See also: *Koji 1.14 Release Notes*

The `encode_int` function is no longer used and has been dropped from the library.

Because we no longer call `encode_int`, the hub will now always use i8 tags when returning large integers, rather than returning them as strings in some cases.

Use custom Kerberos context with krb_login

PR: https://pagure.io/koji/pull-request/1187

Clients can now pass in their own Kerberos context to `ClientSession.krb_login()` using the `ctx` parameter. This is intended for multi-threaded clients.

Custom keyboard interrupt handling in watch_tasks

PR: https://pagure.io/koji/pull-request/981

The new `ki_handler` option for the `koji_cli.lib.watch_tasks()` function allows other cli tools to set their own handler for keyboard interrupts. If specified, the value should be callable and will be called when a keyboard interrupt is encountered. If unspecified, the original behavior is retained.

_unique_path() -> unique_path

PR: https://pagure.io/koji/pull-request/980

The `_unique_path` function is deprecated. It has been replaced by `unique_path`.

Web UI Changes

Additional info on builders in channelinfo page

PR: https://pagure.io/koji/pull-request/989

The channelinfo page now shows enabled/ready status for each host and a count for each.

Builder Changes

Builder task_avail_delay check

PR: https://pagure.io/koji/pull-request/1176

This delay works around a deficiency in task scheduling. The default delay is 300 seconds and can be adjusted with the `task_avail_delay` option to kojid. However, it is unlikely that admins will need to adjust this setting.

Despite the name, this does not introduce any new delay compared to the old behavior. The setting controls how long a host will wait before taking a task in a given channel-arch "bin" when that host has an available capacity lower than the median for that bin. Previously, such hosts could wait forever.

System Changes

Python 3 Support

PR: https://pagure.io/koji/pull-request/1117
PR: https://pagure.io/koji/pull-request/891
PR: https://pagure.io/koji/pull-request/921
PR: https://pagure.io/koji/pull-request/1184
PR: https://pagure.io/koji/pull-request/1019
PR: https://pagure.io/koji/pull-request/685
. . . and many fixes

Support for Python 3 has been extended to all components of Koji. Including:

- Hub
- Builder
- Web UI
- Utils

No more messagebus plugin

PR: https://pagure.io/koji/pull-request/1043

The messagebus plugin has been dropped. The protonmsg plugin is still available.

Simple mode for mergerepos

PR: https://pagure.io/koji/pull-request/1066

External repos now have a `merge_mode` option. Valid values are either `koji` (the old way) or `simple` (a new alternative). This option can be set with the `--mode` option to the `add-external-repo` or `edit-external-repo` commands.

When an external repo is merged with simple mode, a number of the complex filters that Koji normally applies are skipped. This mode still honors the block list from Koji and ignores duplicate NVRAs, but otherwise it simply merges the repo in.

Multiple merge modes cannot be combined in a single tag. If a tag has two external repos with different modes, then the repo will fail to generate.

Avoid "unknown task" errors in Kojira

PR: https://pagure.io/koji/pull-request/1175

This is a bug fix for a minor race condition in Kojira that could cause errors in the log and redundant repo regens.

Full filename display for kojifiles directory indexes

PR: https://pagure.io/koji/pull-request/1156

This is simply a change to the default httpd configuration for serving /mnt/koji. It adds `NameWidth=*` to `IndexOptions` so that long filenames are fully displayed.

Broader support for target/source/scratch tests in channel policy

PR: https://pagure.io/koji/pull-request/962

It is now possible to write channel policy rules based on build target, source, and scratch options for task types other than `build`.

Longer Build Target names

PR: https://pagure.io/koji/pull-request/925

Build target names can now be up to 256 characters, the same length restriction as for tag names.

2.6.2 Koji 1.16.2 Release Notes

Koji 1.16.2 is a bugfix release for Koji 1.16. The purpose of this release is address *CVE-2018-1002161*.

See also:

- *Koji 1.16.1 Release Notes*
- *Koji 1.16.0 Release notes*

Issues fixed in 1.16.2

- Issue 1183 – CVE-2018-1002161

2.6.3 Koji 1.16.1 Release Notes

Koji 1.16.1 is a point release for Koji 1.16. The major changes include:

- Allow target info to be read for different type tasks in channel policy.
- Create symlinks for builds imported onto non-default volumes.
- Fix RPMdiff issues found in Koji 1.16.0.

Please see: *Koji 1.16.0 Release notes*

Issues fixed in 1.16.1

- Issue 847 – spin-livecd failed with "Could not resolve host"
- Issue 932 – Fix use_host_resolv with new mock version
- Issue 1010 – koji fails runroot because of *UnicodeDecodeError*
- Issue 998 – cancel build doesn't work for images
- Issue 994 – rpmdiff calculate wrong results
- Issue 1025 – missing default volume symlink for imported builds affected by volume policy
- Issue 1007 – decode_args() might result in –package parameter missing in runroot command
- Issue 150 – no target info in channel policy for non-rpm tasks
- PR: 973 – Check empty arches before spawning dist-repo
- Issue 958 – Notification for tagBuildBypass is writing message untagged from, expected message tagged into
- Issue 968 – Default enable python3 on RHEL8
- Issue 916 – *clone-tag* doesn't preserve tagging order
- Issue 949 – cli: [rpminfo] KeyError: 'license' for external RPM
- Issue 876 – koji clone-tag raises "UnboundLocalError"
- Issue 945 – Koji build fail due to ambiguous python shebang

2.6.4 Koji 1.16.0 Release notes

Migrating from Koji 1.15

For details on migrating see *Migrating to Koji 1.16*

Security Fixes

CVE-2018-1002150 - distRepoMove missing access check

This release includes the fix for *CVE-2018-1002150*.

Client Changes

CLI commands to manage notifications

PR: https://pagure.io/koji/pull-request/688

The change adds new cli sub-commands:

- list-notifications
- add-notification
- remove-notification
- edit-notification

Previously this functionality was only available through the web ui or by making direct api calls.

Add –old-chroot option to runroot command

PR: https://pagure.io/koji/pull-request/823

This option causes the runroot handler to pass the same-named option to the mock command. This complements the existing `--new-chroot` option.

If neither `--old-chroot` or `--new-chroot` is given, then mock will follow its default behavior. This default varies across mock versions. For newer versions of mock, `--new-chroot` is the default (uses a systemd nspawn container).

Fix runroot output on py3

PR: https://pagure.io/koji/pull-request/828

The runroot command should now work under python3.

Honor runroot –quiet

PR: https://pagure.io/koji/pull-request/806

The `--quiet` option was added to the runroot command in version 1.15, but it only took effect when the `--watch` option was given. Now it is honored in all cases.

Drop old ssl code

PR: https://pagure.io/koji/pull-request/498

The old `koji.ssl` module has been removed, and the `use_old_ssl` option has been removed from client code.

Because these files (which were originally from Plague) were the only parts of Koji that were licensed as GPLv2+, Koji is now simply licensed as LGPLv2.

Builder Changes

Configure install timeout for imagefactory

PR: https://pagure.io/koji/pull-request/841

Previously the install timeout parameter for imagefactory was set to a fixed value of 7200 by Koji. Now it can be controlled by setting the `oz_install_timeout` option in `kojid.conf`.

A value of `0` will disable the timeout.

Record log timestamps

PR: https://pagure.io/koji/pull-request/777

If the `log_timestamps` option is enabled in `kojid.conf`, then the builder will record a separate timestamp file for each log file in a build.

The filename for the timestamp file is generated by taking the name of the log file and appending `-ts.log`. So `build.log` will have timestamp data in `build.log-ts.log`.

The format of the timestamp log is plain text with each line showing a numeric timestamp and a line offset.

Builder option: chroot_tmpdir

PR: https://pagure.io/koji/pull-request/787

The new `chroot_tmpdir` option controls which directory within buildroots is used for various temporary data by the Koji builder daemon. Previously this was hardcoded to `/builddir/tmp`, which created problems with modern versions of mock.

The default value is `/chroot_tmpdir`.

Add internal_dev_setup option to runroot config

PR: https://pagure.io/koji/pull-request/824

The `internal_dev_setup` config option for the runroot builder plugin controls whether the mock option of the same name is set for runroot tasks.

System Changes

Add option to configure DB port

PR: https://pagure.io/koji/pull-request/884

The hub now accepts a `DBPort` option in `hub.conf`, which specifies which port the hub should use when connecting to the database.

Split debuginfo for dist repos

PR: https://pagure.io/koji/pull-request/914

Dist repos can now be generated with debuginfo files split into a separate repo. The behavior is controlled by passing the `--split-debuginfo` option to the `dist-repo` subcommand.

When this option is in effect, the main repo will be in the normal location. The debuginfo repo will be in the `debug` subdirectory. So, you will see a directory structure like:

```
Packages/
repodata/
debug/
debug/repodata
```

Regardless of the split, all the rpms are located in the top level `Packages` directory.

Notifications in [un]tagBuildBypass

PR: https://pagure.io/koji/pull-request/691

Previously the `tagBuildBypass` and `untagBuildBypass` calls did not trigger notifications. Now they will do so by default. The call now accepts a `notify` option (defaults to True) which controls the behavior.

Track history for host data

PR: https://pagure.io/koji/pull-request/778

Koji now tracks changes to host data similarly to the way it tracks changes for other data. This includes

- enabled state
- arches
- capacity
- description & comment
- channels

The `list-history` cli command now supports `--host` and `--channel` options to select history entries for a host or channel.

The versioned host data is stored in the `host_config` and `host_channels` tables.

Fix block-group functionality

PR: https://pagure.io/koji/pull-request/678

The `block-group` command and its underlying api call now actually work.

Strict option for archive listing calls

PR: https://pagure.io/koji/pull-request/734
PR: https://pagure.io/koji/pull-request/748

The `list_archives`, `get_archive_file()`, and `list_archive_files()` hub functions now accept a strict option, which defaults to False. When the option is True, the call will raise an exception if there is no match.

Search build by source

PR: https://pagure.io/koji/pull-request/765

The `listBuilds()` api call now supports a source option. This is treated as a glob pattern and matched against the `source` field of the build.

Option to ignore tags in kojira

PR: https://pagure.io/koji/pull-request/695

Kojira now supports an `ignore_tags` option. This is treated as a space-separated list of glob patterns. Tags that match are ignored by kojira (it will not generate newRepo tasks for them).

Improve kojira throughput

PR: https://pagure.io/koji/pull-request/797

Kojira should be much more responsive in triggering `newRepo` tasks.

Drop migrateImage call

PR: https://pagure.io/koji/pull-request/632

The `migrateImage` call hub call has been removed.

This call was added in version 1.8 (April 2013) as a one-time tool for migrating images from the old model (no build entry) to the new model (image build type). It was only available if the EnableImageMigration option was set on the hub.

2.6.5 Koji 1.15.1 Release Notes

Koji 1.15.1 is a bugfix release for Koji 1.15. The most important change is the fix for *CVE-2018-1002150*.

Please see: *Koji 1.15 Release Notes*

Issues fixed in 1.15.1

- Issue 850 – CVE-2018-1002150
- Issue 846 – error occurs in SCM.get_source since subprocess.check_output is not supported by python 2.6-
- Issue 724 – buildNotification of wrapperRPM fails because of task["label"] is None
- Issue 786 – buildSRPMFromSCM tasks fail on koji 1.15
- Issue 803 – Email notifications makes build tasks fail with "KeyError: 'users_usertype'"
- Issue 742 – dict key access fail in koji_cli.commands._build_image
- Issue 811 – AttributeError: 'dict' object has no attribute 'hub.checked_md5'
- Issue 813 – cg imports fail with "Unsupported checksum type"

2.6.6 Koji 1.15 Release Notes

Updates

- *Koji 1.15.1* is a security update for Koji 1.15

Migrating from the previous release

For details on migrating see *Migrating to Koji 1.15*

Client Changes

Display license Info

PR: https://pagure.io/koji/pull-request/686

The `rpminfo` command now displays the `License` field from the rpm.

Keytabs for GSSAPI authentication

PR: https://pagure.io/koji/pull-request/708

Previously keytabs were only supported by the older kerberos auth method, which is not available on Python 3. Now the gssapi method supports them as well.

Add krb_canon_host option

PR: https://pagure.io/koji/pull-request/653

This release adds a `krb_canon_host` option that tells Koji clients to use the dns canonical hostname for kerberos auth.

This option allows kerberos authentication to work in situations where the hub is accessed via a cname, but the hub's credentials are under its canonical hostname.

If specified, this option takes precedence over the older option named `krb_rdns`. That option caused Koji clients to perform a reverse name lookup for kerberos auth.

When configuring kojiweb (in web.conf), the option is named `KrbCanonHost`.

Both options only affect the older kerberos authentication path, and not gssapi.

Watch-task return code

PR: https://pagure.io/koji/pull-request/703

Previously, the `watch-task` command would return a non-zero exit status if any subtask failed, even if this did not cause the parent task to fail.

Now that we have cases where subtasks are optional, this no longer makes sense. The exit code is now based solely on the results of the top level tasks it is asked to watch.

New runroot options

PR: https://pagure.io/koji/pull-request/633

The `runroot` command now supports options similar to the various build commands. These new options are:

```
--nowait              Do not wait on task
--watch               Watch task instead of printing runroot.log
--quiet               Do not print the task information
```

New watch-logs options

PR: https://pagure.io/koji/pull-request/625

The `watch-logs` command now supports the following new options:

```
--mine      Watch logs for all your tasks
--follow    Follow spawned child tasks
```

Web UI changes

Archive component display

PR: https://pagure.io/koji/pull-request/610

Previously, the web UI only displayed component lists for image builds. However, new build types can also have component lists.

Now the interface will display components for any archive that has them.

Display license Info

PR: https://pagure.io/koji/pull-request/686

The `rpminfo` page now displays the `License` field from the rpm.

Show suid bit

PR: https://pagure.io/koji/pull-request/617

The web UI will now display the setuid bit when displaying rpm/archive file contents.

Builder changes

Alternate tmpdir for mock chroots

PR: https://pagure.io/koji/pull-request/602

Recent versions of mock (1.4+) default to `use_nspawn=True`, which results in /tmp being a fresh tmpfs mount on every run. This means the /tmp directory no longer persists outside of the mock invocation.

Now, the builder will use /builddir/tmp instead of /tmp for persistent data.

Store git commit hash

PR: https://pagure.io/koji/pull-request/674

In Koji, for builds from an SCM, the source is specified as an scm url. For git urls, the revision in that url can be anything that git will recognize, including:

- a sha1 ref
- an abbreviated sha1 ref
- a branch name
- a tag
- HEAD

With this change:

- the revision is replaced with the full sha1 ref for git urls
- the scm url is stored in build.source
- the original scm url is saved in build.extra

Previously, this source url was not properly stored for rpm builds. It appeared in the task parameters, but the build.source field remained blank. If a symbolic git ref (e.g. HEAD) was given in the url, the underlying sha1 value was only recorded in the task logs.

System changes

Volume policy support

PR: https://pagure.io/koji/pull-request/622

Koji has for many years had the ability to split its storage across multiple volumes. However, there is no automatic process for placing builds onto volumes other than the primary. To do so often requires a lot of manual work from an admin.

This feature:

- adds a volume policy check to the key import pathways
- adds an applyVolumePolicy call to apply the policy to existing builds

The hub consults the volume policy at various points to determine where a build should live. This allows admins to make rules like:

- all kernel builds go to the volume named kstore
- all builds built from the epel-7-build tag go to the volume named epel7
- all builds from the osbs content generator go to the volume named osbs

The default policy places all builds on the default volume.

See also: *Storage Volumes*

Messagebus plugin changes

PR: https://pagure.io/koji/pull-request/537

There are two notable changes to the messagebus plugin this release:

Deferred sending

Similar to the current behavior of the protonmsg plugin, messages are queued up during hub calls and only sent out during the `postCommit` callback.

This avoids sending messages about failed calls, which can be confusing to message consumers (e.g. build state change messages about a build that does not exist because it failed to import).

Test mode

The plugin now looks for a boolean `test_mode` option. If it is true, then the messages are still queued up, but not actually sent. This makes it possible to enable the plugin in test environments without having to set up a separate message bus.

Protonmsg plugin changes

PR: https://pagure.io/koji/pull-request/657
PR: https://pagure.io/koji/pull-request/651

There are two changes to how the protonmsg plugin handles rpmsign events:

1. The arch of the rpm is included in messages
2. The message are omitted when the sigkey is empty

No notifications for disabled users or hosts

PR: https://pagure.io/koji/pull-request/615

Koji will no longer send out email notifications to disabled users or to users corresponding to a host.

Replace pycurl with requests

PR: https://pagure.io/koji/pull-request/601

All uses of the pycurl library have been replaced with calls to python-requests, so pycurl is no longer required.

Drop importBuildInPlace call

PR: https://pagure.io/koji/pull-request/606

The deprecated `importBuildInPlace` call has been dropped.

This call was an artifact of a particular bootstrap event that happened a long time ago. It was never really documented or recommended for use.

2.6.7 Koji 1.14 Release Notes

Migrating from Koji 1.13

For details on migrating see *Migrating to Koji 1.14*

Client Changes

Fail fast option for builds

PR: https://pagure.io/koji/pull-request/432

When builders are configured with `build_arch_can_fail = True` then the failure of a single buildArch task does not immediately cause the build to fail. Instead, the remaining buildArch tasks are allowed to complete, at which point the build will still fail.

Sometimes developers would rather a build fail immediately, so we have added the `--fail-fast` option to the build command, which overrides this setting. The option only has an effect if the builders are configured to fail slow.

Custom Lorax templates

PR: https://pagure.io/koji/pull-request/419

Koji now supports custom Lorax templates for the `spin-livemedia` command. The command accepts two new options:

```
--lorax_url=URL       The URL to the SCM containing any custom lorax
                      templates that are to be used to override the default
                      templates.
--lorax_dir=DIR       The relative path to the lorax templates directory
                      within the checkout of "lorax_url".
```

The Lorax templates must come from an SCM, and the `allowed_scms` rules apply.

When these options are used, the templates will be fetched and an appropriate `--lorax-templates` option will be passed to the underlying livemedia-creator command.

Allow profiles to request a specific python version

PR: https://pagure.io/koji/pull-request/566

On platforms with python3 available, the Koji client is built to execute with the python3 binary. However, there are a few client features that do not work under python3, notably old-style (non-gssapi) Kerberos authentication.

If this issue is affecting you, you can set `pyver=2` in your Koji configuration. This can be done per profile. When Koji sees this setting at startup, it will re-execute itself under the requested python binary.

New list-builds command

PR: https://pagure.io/koji/pull-request/526

The command line now has a `list-builds` command that has similar functionality to the builds tab of the web interface.

```
Usage: koji list-builds [options]
(Specify the --help global option for a list of other help options)

Options:
  -h, --help            show this help message and exit
  --package=PACKAGE     List builds for this package
  --buildid=BUILDID     List specific build from ID or nvr
  --before=BEFORE       List builds built before this time
  --after=AFTER         List builds built after this time
  --state=STATE         List builds in this state
  --type=TYPE           List builds of this type.
  --prefix=PREFIX       Only list packages starting with this prefix
  --owner=OWNER         List builds built by this owner
  --volume=VOLUME       List builds by volume ID
  -k FIELD, --sort-key=FIELD
                        Sort the list by the named field
  -r, --reverse         Print the list in reverse order
  --quiet               Do not print the header information
```

New block-group command

PR: https://pagure.io/koji/pull-request/509

The `block-group` command allows admins to block package group entries without having to resort to the `call` command.

```
Usage: koji block-group <tag> <group>
(Specify the --help global option for a list of other help options)

Options:
  -h, --help  show this help message and exit
```

Exit codes for some commands

PR: https://pagure.io/koji/pull-request/558
PR: https://pagure.io/koji/pull-request/559

Several more commands will now return a non-zero exit code when an error occurs:

- the various image building commands
- the `save-failed-tree` command (provided by a plugin)

Easier for scripts to use activate_session

PR: https://pagure.io/koji/pull-request/493

In Koji 1.13.0, it became possible for scripts to `import koji_cli.lib` and gain access to the `activate_session` function that the command line tool uses to authenticate.

In this release, this function has been made easier for scripts to use:

- the options argument can now be a dictionary
- less options need to be specified

Builder changes

Normalize paths for scms

PR: https://pagure.io/koji/pull-request/591

For many years, kojid has supported the `allowed_scms` option (see: *Source Control Configuration*) for controlling which scms can be used for building. In 1.13.0, Koji added the ability to explicitly block a host:path pattern.

Unfortunately, 1.13.0 did not normalize the path before checking the pattern, making it possible for users to use equivalent paths to route around the block patterns.

Now, Koji will normalize these paths before the `allowed_scms` check.

Graceful reload

PR: https://pagure.io/koji/pull-request/565

For a long time kojid has handled the USR1 signal by initiating a graceful restart. This change exposes that in the systemd service config (and the init script on older platforms).

Now, `service kojid reload` will trigger the same sort of restart that the `restart-hosts` command accomplishes, but only for the build host you run it on. When this happens, kojid will:

- stop taking new tasks
- wait for current tasks to finish
- restart itself once all its tasks are completed

Friendlier runroot configuration

PR: https://pagure.io/koji/pull-request/539
PR: https://pagure.io/koji/pull-request/528

Two changes make it easier to write a configuration for the runroot plugin.

The `path_subs` option is now more forgiving about whitespace:

- leading and trailing whitespace is ignored for each line
- blank lines are ignored

The `[pathNN]` sections are no longer required to have sequential numbers. Previously, the plugin expected a sequence like `[path0]`, `[path1]`, `[path2]`, etc, and would stop looking for entries if the next number was missing. Now, any set of distinct numbers is valid and all `[pathNN]` sections will be processed.

System changes

Deprecations

PR: https://pagure.io/koji/pull-request/554
PR: https://pagure.io/koji/pull-request/597

The following features are deprecated and will be removed in a future release:

- the `importBuildInPlace` rpc call
- the `use_old_ssl` client configuration option (and the underlying `koji.compatrequests` library)

Removed calls

PR: https://pagure.io/koji/pull-request/497
PR: https://pagure.io/koji/pull-request/507

The deprecated `buildFromCVS` hub call has been removed. It was replaced by the `buildSRPMFromCVS` call many years ago and has been deprecated since version 1.6.0.

The `add_db_logger` function has been removed from the koji library, along with the `log_messages` table in the db. This extraneous call has never been used in Koji.

Dropped mod_python support

PR: https://pagure.io/koji/pull-request/508

Koji no longer supports mod_python. This option has been deprecated since mod_wsgi support was added in version 1.7.0.

See also: *Migrating to Koji 1.7*

Large integer support

PR: https://pagure.io/koji/pull-request/571

Koji uses xmlrpc for communications with the hub, and unfortunately the baseline xmlrpc standard only supports 32-bit signed integers. This results in errors when larger integers are encountered, typically when a file is larger than 2 GiB.

Starting with version 1.14.0, Koji will emit `i8` tags when encoding large integers for xmlrpc. Integers below the limit are still encoded with the standard `int` tag. The only time this makes a difference is when Koji would previously have raised an `OverflowError`.

The `i8` tag comes from the ws-xmlrpc spec. Python's xmlrpc decoder has for many years accepted and understood this tag, even though its encoder would not emit it.

Previous versions of Koji worked around such size issues by converting large integers to strings in a few targeted places. Those targeted workarounds have been left in place on the hub for the sake of backward compatibility.

Test mode for protonmsg plugin

PR: https://pagure.io/koji/pull-request/538

The `protonmsg` plugin now accepts a boolean `test_mode` configuration option. When this option is enabled, the plugin will not actually send messages, but will instead log them (at the DEBUG level).

This option allows testing environments to run with the plugin enabled, but without requiring a message bus to be set up for that environment.

Handling of debugsource rpms

PR: https://pagure.io/koji/pull-request/524

Koji will now treat rpms ending in `-debugsource` the same way that it does other debuginfo rpms. Such rpms are:

- omitted from Koji's normal yum repos
- listed separately when displaying builds
- not downloaded by default in the `download-build` command

Added kojifile component type for content generators

PR: https://pagure.io/koji/pull-request/506

Content generator imports now accept entries with type equal to `kojifile` in the component lists for buildroots and images/archives. This type provides a more reliable way to reference archive that come from Koji.

See: *Example metadata*.

2.6.8 Koji 1.13 Release Notes

Migrating from Koji 1.12

For details on migrating see *Migrating to Koji 1.13*

Client Changes

Python 3 client support

PR: https://pagure.io/koji/pull-request/417

The koji command and core library now support Python 3 (as well as 2). The default spec now produces both *python2-koji* and *python3-koji* subpackages. The *koji* package still contains the (now much smaller) `/usr/bin/koji` file.

Some older features are not supported by the Python 3 client

- the *use_old_ssl* option is not supported, python-requests must be used
- the old kerberos auth mechanism is not supported, use gssapi instead

CLI Plugins

PR: https://pagure.io/koji/pull-request/199

The command line interface now has basic plugin support. The primary use case is for plugins to be able to add new subcommands. For details see: *New command for CLI*

list-channels CLI command

PR: https://pagure.io/koji/pull-request/442

The new *list-channels* command lists the known channels for the system.

```
Usage: koji list-channels
(Specify the --help global option for a list of other help options)

Options:
  -h, --help  show this help message and exit
  --quiet     Do not print header information
```

hostinfo CLI command

PR: https://pagure.io/koji/pull-request/399
Issue: https://pagure.io/koji/issue/364

The new `hostinfo` command shows basic information about a build host, similar to the web interface.

```
Usage: koji hostinfo [options] <hostname> [<hostname> ...]
(Specify the --help global option for a list of other help options)

Options:
  -h, --help  show this help message and exit
```

Enhancements to restart-hosts

PR: https://pagure.io/koji/pull-request/472

The *restart-hosts* command is used by admins to safely restart the build hosts after a configuration change.

Because multiple restarts can conflict, the command will now exit with a error if a restart is already underway (can be overridden with –force).

There are now options to limit the restart to a given channel or arch.

The command now has a timeout option, which defaults to 24hrs.

User-Agent header

PR: https://pagure.io/koji/pull-request/393
Issue: https://pagure.io/koji/issue/392

Previously the Koji client library reported a confusingly out-of-date value in the `User-Agent` header. Now it simply reports the major version.

raise error on non-existing profile

PR: https://pagure.io/koji/pull-request/375
Issue: https://pagure.io/koji/issue/370

If the requested client profile is not configured, the library will raise an error, rather than proceeding with default values.

See also: *Koji Profiles*

Changes to the Web interface

Build Log Display

PR: https://pagure.io/koji/pull-request/471

The build info pages now display the log files for a build (instead of linking directly to the directory on the download server). This works for all builds, including those imported by content generators.

Builder changes

Configuring mock chroot behavior

PR: https://pagure.io/koji/pull-request/400
Issue: https://pagure.io/koji/issue/398

Koji now supports using mock's –new-chroot option on a per-tag basis. For details see: *Tuning mock's behavior per tag*

pre/postSCMCheckout callbacks

The callback interface is used by plugins to hook into various Koji operations. With this release we have added callbacks in the builder daemon for before and after source checkout: `preSCMCheckout` and `postSCMCheckout`.

Extended allowed_scms format

PR: https://pagure.io/koji/pull-request/421

The allowed_scms option now accepts entries like:

```
!host:repository
```

to explicitly block a host:repository pattern.

See also: *Source Control Configuration*

System changes

mod_auth_gssapi required

PR: https://pagure.io/koji/pull-request/444

On modern platforms, both koji-hub and koji-web now require mod_auth_gssapi instead of mod_auth_kerb.

Longer tag names

PR: https://pagure.io/koji/pull-request/388
Issue: https://pagure.io/koji/issue/369

Previously, tag names were limited to 50 characters. They are now limited to 256 characters.

For releases before 1.13, see the migration guides:

Migrations

2.7 Migrations

2.7.1 Migrating to Koji 1.17

You should consider the following changes when migrating to 1.17:

DB Updates

This release some minor schema changes

- the `tag_external_repos` table has a new `merge_mode` column
- the `build_target.name` column is now a TEXT field rather than VARCHAR

As in previous releases, we provide a migration script that updates the database.

```
# psql koji koji  </usr/share/doc/koji/docs/schema-upgrade-1.16-1.17.sql
```

Other changes

There are numerous other changes in 1.17 that should not have a direct impact on migration. For details see: *Koji 1.17.0 Release notes*

2.7.2 Migrating to Koji 1.16

You should consider the following changes when migrating to 1.16:

DB Updates

This release has schema changes to support tracking history for hosts.

- new table: `host_config`
- some fields from the `host` table have moved to `host_config`
- the `host_channels` table now has versioning data like the other versioned tables

As in previous releases, we provide a migration script that updates the database.

```
# psql koji koji  </usr/share/doc/koji/docs/schema-upgrade-1.15-1.16.sql
```

Other changes

There are numerous other changes in 1.16 that should not have a direct impact on migration. For details see: *Koji 1.16.0 Release notes*

2.7.3 Migrating to Koji 1.15

The update from to 1.15 is comparatively light from a migration perspective.

DB Updates

There are no schema updates in 1.15.

Other changes

There are numerous other changes in 1.15 that should not have a direct impact on migration. For details see: *Koji 1.15 Release Notes*

2.7.4 Migrating to Koji 1.14

You should consider the following changes when migrating to 1.14:

DB Updates

The schema updates this time are minor

- dropped unused `log_messages` table
- new standard entries in the `archivetypes` table

As in previous releases, we provide a migration script that updates the database.

```
# psql koji koji  </usr/share/doc/koji/docs/schema-upgrade-1.13-1.14.sql
```

Dropped mod_python support

Koji's support for mod_python has been deprecated for many years. If you are still relying on mod_python, you will need to switch to mod_wsgi.

See: *Migrating to Koji 1.7*

Other changes

There are numerous other changes in 1.14 that should not have a direct impact on migration. For details see: *Koji 1.14 Release Notes*

2.7.5 Migrating to Koji 1.13

The 1.13 release of Koji includes several changes that you should consider when migrating.

DB Updates

We have increased the length limit for tag names and there is a minor schema change to support this.

As in previous releases, we provide a migration script that updates the database.

```
# psql koji koji  </usr/share/doc/koji/docs/schema-upgrade-1.12-1.13.sql
```

Packaging changes

Because the CLI and base library now support both python2 and python3, the core libs and most of the cli code have moved to separate packages for each major Python version:

- python2-koji
- python3-koji

The main koji package still contains the (now much smaller) koji script, and requires either python2-koji or python3-koji, depending on whether python3 support is enabled.

The CLI now also supports plugins, and two commands (runroot and save-failed-tree) have moved to the *python[23]-koji-cli-plugins* subpackages. If you need these subcommands, you may need to explicitly install the appropriate koji-cli-plugins package.

Other changes

There are numerous other changes in 1.13 that should not have a direct impact on migration. For details see: *Koji 1.13 Release Notes*

2.7.6 Migrating to Koji 1.12

The 1.12 release of Koji includes a several changes that you should consider when migrating.

DB Updates

There is a minor update to support the dist-repos feature:

- The `repo` table now has a `dist` column

Additionally, the schema explicitly adds the `image` permission to the permissions table, correcting an old oversight.

As in previous releases, we provide a migration script that updates the database.

```
# psql koji koji  </usr/share/doc/koji/docs/schema-upgrade-1.11-1.12.sql
```

Command line changes

The `import-sig` command now now supports a `--write` option to immediately write out a signed copy.

The `write-signed-rpm` command previously (and confusingly) only accepted nvrs as arguments (i.e. builds not rpms). Now it accepts either nvras or nvrs (rpms or builds).

The `clone-tag` command has been refactored. It supports many more options and should execute much faster than before.

The new `dist-repo` command creates rpm repos suitable for distribution.

The new `save-failed-tree` command allows the a task owner (or admin) to download information from the buildroot of a failed build. This feature requires the `save_failed_tree` plugin to be enabled on the hub and builders.

Configuration

The only configuration changes are for the `save-failed-tree` plugins (hub and builder). Each has its own configuration file. See *Plugins*

The hub accepts a new `CheckClientIP` option (default True) to indicate whether authentication credentials should be tied to the client's IP address. (For some proxy setups, this may need to be set to False).

RPC API Changes

New rpc calls:

`listPackagesSimple` handles a limited subset of the functionality provided by the `listPackages` call

`distRepo` triggers generation of a distribution repo

Changes to calls:

- repo related calls (e.g. `repoInfo` now include a boolean `dist` field
- the `editTag2` call can now remove tag_extra data if the `remove_extra` keyword argument is used
- the listTaskOutput call supports a new `all_volumes` keyword argument. If true, the results are extended to deal with files in same relative paths on different volumes.
- the `getTaskResult` call takes an optional boolean `raise_fault` argument
- the `taskWaitResults` call takes an optional `canfail` argument to indicate subtasks which can fail without raising an exception

2.7.7 Migrating to Koji 1.11

The 1.11 release of Koji includes a several changes that you should consider when migrating.

DB Updates

There are a number of new tables and columns to support content generators. Here is a summary:

- The `btype` table tracks the known btypes [LINK] in the system
- The `build_types` table links builds to their btype(s)
- The `content_generator` table tracks the known content generators in the system

- The `cg_users` table tracks which users have access to which content generators
- The `buildroot` table now tracks more generic buildroots
- The `standard_buildroot` table tracks data for "normal" koji buildroots
- Several tables now have an `extra` column that stores json data
- There are several new entries in the `archivetypes` table
- The `image_listing` table has been replace by the more general `archive_rpm_components` table
- The new `archive_components` complements this and tracks non-rpm components

As in previous releases, we provide a migration script that updates the database.

```
# psql koji koji  </usr/share/doc/koji-1.11.0/docs/schema-upgrade-1.10-1.11.sql
```

Note: prior to this release, we had some interim update scripts:

- schema-update-cgen.sql
- schema-update-cgen2.sql

Most users should not need these scripts. The new schema upgrade script includes those changes.

Command line changes

The `help` command now shows a categorized list of commands.

The `hello` command now reports the authentication type.

Several commands support new arguments. Here are the notable changes:

`add-tag`

- `--extra` : Set an extra option for tag at creation

`watch-task`

- Supports several new task selection options

`download-build`

- `--rpm` : Used to download a particular rpm by name

`runroot`

- `--new-chroot` : Run command with the –new-chroot (systemd-nspawn) option to mock

And there are five new commands

- `assign-task`
- `import-cg`
- `grant-cg-access`
- `revoke-cg-access`
- `spin-livemedia`

Client configuration options

The command line and several other tools support the following new configuration options:

- `use_old_ssl` : Use the old ssl code instead of python-requests
- `no_ssl_verify` : Disable certificate verification for https connections
- `upload_blocksize` : Override the blocksize for uploads
- `krb_rdns` : Use the fqdn of the server when authenticating via kerberos

The `ca` option is deprecated and no longer required for ssl authentication (`serverca` is still required).

Even if not using ssl authentication, the `serverca` option, if specified, is used to verify the certificate of the server.

Other Configuration changes

The Koji web interface supports the following new configuration options:

- `KrbRDNS` : Use the fqdn of the server when authenticating via kerberos
- `LoginDisabled` : Hide the login link at the top of the page

RPC API Changes

New rpc calls:

- `CGImport` : Used by content generators
- `getBuildType` : Returns typeinfo for a build
- `listBTypes` : List the known btypes for the system
- `addBType` : Adds a new btype
- `grantCGAccess` : Grants a user content generator access
- `revokeCGAccess` : Revokes content generator access

Changes to calls

- Several information calls now return additional fields
- `getRPMDeps` returns optional deps
- `listTasks` supports new selection options
- `getLoggedInUser` includes an authtype field

2.7.8 Migrating to Koji 1.10

The 1.10 release of Koji includes a few changes that you should consider when migrating.

DB Updates

The new `tag_extra` table tracks extra data for tags.

There is a new entry in the `channels` table and some additions and updates to the `archivetypes` table.

As in previous releases, we provide a migration script that updates the database.

```
# psql koji koji  </usr/share/doc/koji-1.10.0/docs/schema-upgrade-1.9-1.10.sql
```

Command line changes

A few commands support new arguments

maven-build

- `--ini` : Pass build parameters via a .ini file
- `--section` : Get build parameters from this section of the .ini

wrapper-rpm

- `--ini` : Pass build parameters via a .ini file
- `--section` : Get build parameters from this section of the .ini

import

- `--link` : Attempt to hardlink instead of uploading

list-tagged

- `--latest-n`: Only show the latest N builds/rpms

list-history

- `--watch` : Monitor history data

edit-tag

- `--extra` : Set tag extra option

list-tasks

- `--user` : Only tasks for this user
- `--arch` : Only tasks for this architecture
- `--method` : Only tasks of this method
- `--channel` : Only tasks in this channel
- `--host` : Only tasks for this host

download-build

- `--task-id` : Interpret id as a task id

And there are three new commands

- `image-build-indirection`
- `maven-chain`
- `runroot`

Other Configuration changes

The Koji web interface can now treat `extra-footer.html` as a Cheetah template. This behavior can be enabled by setting the `LiteralFooter` option to `False` in the kojiweb config.

RPC API Changes

The `readTaggedBuilds` and `readTaggedRPMS` now treat an integer value for the optional latest argument differently. Before it was simply treated as a boolean flag, which if true caused the call to return only the latest build for each package. Now, if the value is a positive integer N, it will return the N latest builds for each package. The behavior is unchanged for other values.

New rpc calls: `chainMaven`, `buildImageIndirection`, and `mergeScratch`

2.7.9 Migrating to Koji 1.9

The 1.9 release of Koji includes a few changes that you should consider when migrating.

DB Updates

ImageFactory support introduced some new archive types. These have been added to the `archivetypes` table. The inaccurate `vmx` entry has been removed.

As in previous releases, we provide a migration script that updates the database.

```
# psql koji koji  </usr/share/doc/koji-1.9.0/docs/schema-upgrade-1.8-1.9.sql
```

Command line changes

The command line interface handles configuration files a little differently. Old configs should work just fine, but now there are new options and enhancements.

In addition to the main configuration files, the koji cli now checks for `/etc/koji.conf.d` and `~/.koji/config.d` directories and loads any `*.conf` files contained within. Also if the user specifies a directory with the `-c/--config` option, then that directory will be processed similarly.

The command line supports a new `-p/--profile` option to select alternate configuration profiles without having to link or rename the koji executable.

The new `image-build` command is used to generate images using ImageFactory. The older spin-appliance command is now deprecated.

The `mock-config` command no longer requires a name argument. You can still specify one if you want to override the default choice. It also supports new options. The `--latest` option causes the resulting mock config to reference the `latest` repo (a varying symlink). The `--target` option allows generating the config from a target name.

Other command line changes include: * a new `download-logs` command * the `list-groups` command now accepts event args * the `taginfo` command now reports the list of comps groups for the tag * the fast upload feature is now used automatically if the server supports it

Other Configuration changes

There are also some minor configuration changes in other parts of Koji.

In `kojid` the time limit for rpm builds is now configurable via the `rpmbuild_timeout` setting in `kojid.conf`. The default is 24 hours.

The `koji-gc` tool supports two new configuration options. The `krbservice` option allows you to specify the kerberos service for authentication, and the `email_domain` option allows you to specify the email domain for sending gc notices.

The messagebus hub plugin now supports `timeout` and `heartbeat` options for the message bus connection.

RPC API Changes

Most of these changes are extensions, though some of the host-only call changes are incompatible.

The `tagHistory` call accepts a new named boolean option (`active`) to select only active/inactive entries. It also now reports the additional fields `maven_build_id` and `win_build_id` if builds are maven or win builds respectively.

New rpc calls: `buildImageOz`, `host.completeImageBuild`, and `host.evalPolicy`.

The host-only calls `host.moveImageBuildToScratch` and `host.importImage` no longer accept the `rpm_results` argument. The rpm results can be embedded in the regular `results` argument.

2.7.10 Migrating to Koji 1.8

The 1.8 release of Koji refactors how images (livecd and appliance) are stored in the database and on disc. These changes will require a little extra work when updating.

There have also been some changes to the command line.

Finally, kojira accepts some new options.

DB Schema Updates

Previous to 1.8, images were stored in separately from other builds, both in the database and on disc. The new schema adds new tables: `image_builds`, `image_listing`, and `image_archives`.

The following tables are now obsolete: `imageinfo` and `imageinfo_listing`. However you should not drop these tables until you have migrated your image data.

As in previous releases, we provide a migration script that updates the database.

```
# psql koji koji  </usr/share/doc/koji-1.8.0/docs/schema-upgrade-1.7-1.8.sql
```

Note that the SQL script does not (and can not) automatically migrate your old image data to the new tables. After applying the schema changes, you can migrate old images using the `migrateImage` hub call. This method is necessary because the new schema requires each image to have a name, version, and release value. The values for name and version cannot be automatically guessed.

Migrating your old images

If you have old images, you can migrate them to the new system using the `migrateImage` hub call. This call requires admin privilege and must also be enabled with the `EnableImageMigration` configuration option in `hub.conf`.

The signature of the call is:

```
migrateImage(old_image_id, name, version)
```

This call can made from the command line:

```
# koji call migrateImage 45 my_livecd 1.1
```

Cleaning up

After you have migrated any necessary images to the new system, you may want to remove the old database tables and filesystem directories. This step is *optional*. If you want to leave the old data around, it will not affect Koji.

Before you take any of the following actions, please *make sure* that you have migrated any desired images.

Removing the old data is simply a matter of dropping tables and deleting files.

```
koji=> DROP TABLE imageinfo_listing;
koji=> DROP TABLE imageinfo;
# rm -rf /mnt/koji/images
```

Command line changes

For clarity and consistency, all of the `-pkg` commands have been renamed to `-build` commands.

```
latest-pkg -> latest-build
move-pkg -> move-build
tag-pkg -> tag-build
untag-pkg -> untag-build
```

For backwards compatibility, the old commands names are also recognized.

A new command has been added, `remove-pkg`.

Several commands have been modified to support images.

The `spin-livecd` and `spin-appliance` commands now require additional arguments. These arguments specify the name and version to use for the image.

New kojira options

The following options are new to kojira:

```
max_delete_processes
max_repo_tasks_maven
```

Previously, kojira ran as a single process and repo deletions could potentially slow things down (particularly for Maven-enabled repos). Now kojira spawns a separate process to handle these deletions. The `max_delete_processes` determines how many such processes it will launch at one time.

When Maven-enabled repos are in use, they can potentially take a very long time to regenerate. If a number of these pile up it can severely slow down regeneration of non-Maven repos. The `max_repo_tasks_maven` limits how many Maven repos kojira will attempt to regenerate at once.

Also the following kojira option has been removed:

```
prune_batch_size
```

2.7.11 Migrating to Koji 1.7

The 1.7 release of Koji contains changes that will require a little extra work when updating. These changes are:

- DB schema updates to support storage volumes
- The change from mod_python to mod_wsgi

- The introduction of a separate configuration file for koji-web
- Changes to url options

DB Schema Updates

The 1.7 release adds two new tables to the database. The `volume` table tracks the names of available storage volumes, and the `tag_updates` table tracks changes to tags that are not easily calculated from other tables. There is also a new field in the `build` table, `volume_id`, which indicates which volume a build is stored on.

As in previous releases, we provide a migration script that updates the database.

```
# psql koji koji  </usr/share/doc/koji-1.7.0/docs/schema-upgrade-1.5-1.7.sql
```

mod_python and mod_wsgi

Koji now defaults to using mod_wsgi to interface with httpd. Support for mod_python is _deprecated_ and will disappear in a future version of Koji. Koji administrators can opt to stay on mod_python for now, but some minor configuration changes will be required.

Migrating to mod_wsgi

The mod_wsgi package is now required for both koji-hub and koji-web. Folks running RHEL5 can find mod_wsgi in EPEL.

You will need to adjust your http config for both koji-hub and koji-web. Our example config files default to mod_wsgi. To adapt your existing config, you will need to:

- **For both the koji-hub and koji-web/scripts directories:**
  - add `Options ExecCGI`
  - change `SetHandler` from mod_python to wsgi-script
- Ensure that the koji-web Alias points to wsgi_publisher.py
- If you have not already, migrate all koji-hub PythonOptions to hub.conf
- Migrate all koji-web PythonOptions to web.conf (see later section)

Staying on mod_python

Support for mod_python is _deprecated_ and will disappear in a future version of Koji.

While we have made efforts to maintain mod_python compatibility, there are a few configuration changes you will need to make.

The koji-hub http config should continue to function without modification.

The koji-web http config will, at minimum, require the following changes:

- Ensure that the koji-web `Alias` points to wsgi_publisher.py
- Change koji-web's `PythonHandler` setting to wsgi_publisher

Our example http configurations contain commented examples of mod_python configuration.

Even if you stay on mod_python, we recommend that you migrate away from using PythonOptions and place your configuration in web.conf and hub.conf.

Web Configuration

Starting with version 1.7, koji-web uses a separate configuration file, rather than PythonOptions embedded in the httpd config. The location of the new file is:

```
/etc/kojiweb/web.conf
```

The web.conf file is an ini-style configuration file. Options should be placed in the [web] section. All previous options accepted via PythonOptions are accepted in web.conf. Please see the example web.conf file.

Custom Config File Location

The location of web.conf can be specified in the httpd configuration. To specify the location under mod_wsgi, use:

```
SetEnv koji.web.ConfigFile /path/to/web.conf
```

Under mod_python, use:

```
PythonOption koji.web.ConfigFile /path/to/web.conf
```

If you opt to stay on mod_python, the server will continue to process the old PythonOptions. To ease migration, it does so by default unless the koji.web.ConfigFile PythonOption is specified. In order to use web.conf under mod_python, you *must* specify koji.web.ConfigFile in your http config.

We strongly recommend moving to web.conf. The server will issue a warning at startup if web.conf is not in use.

Changes to url options

The pkgurl option has been removed from the koji command line tool and from the build daemon (kojid). The url for packages is deduced from the topurl option, which should point to the top of the /mnt/koji tree.

Any config files that specify pkgurl (e.g. ~/.koji/config, /etc/koji.conf, or /etc/kojid/kojid.conf) will need to be adjusted.

Similarly, the kojiweb config options KojiPackagesURL, KojiMavenURL, and KojiImagesURL have been dropped in favor of the new option KojiFilesURL.

Additional Notes

Split Storage

Apart from the schema changes, no other migration steps are required for the split storage feature. By default, builds are stored in the normal location.

Web Themes

Using the old method (httpd aliases for koji static content) should continue to work. For (brief) instructions on the new method, see the README file under koji-web/static/themes.

2.8 Koji CVEs

2.8.1 CVE-2018-1002161

SQL injection in multiple remote calls

FAQ for CVE-2018-1002161

Following are answers to some questions regarding CVE-2018-1002161 for Koji. If you haven't already, you should read the *announcement*.

If you have questions not covered here or in the announcement, please ask them on the koji-devel mailing list.

> https://lists.fedorahosted.org/archives/list/koji-devel@lists.fedorahosted.org/

Q: Does this issue affect Koji clients or builders?

> The issue only affects the Koji hub.

Q: Which versions of Koji are affected?

> All previous versions of Koji are affected, except for the legacy-py24 branch because it contains no hub code.

Q: Where are the fixed versions?

> For Koji 1.11, 1.11.1 and higher include the fix
> For Koji 1.12, 1.12.2 and higher include the fix
> For Koji 1.13, 1.13.2 and higher include the fix
> For Koji 1.14, 1.14.2 and higher include the fix
> For Koji 1.15, 1.15.2 and higher include the fix
> For Koji 1.16.2 and higher include the fix
>
> You can find all of these versions on our releases page:
>
> https://pagure.io/koji/releases

Q: What about older versions?

> We have only backported the fix to Koji versions released in the past few years. If you are still using a very old version of Koji, we strongly recommend that you shut it down and migrate to a newer version.

Q: What can be done with this exploit?

> The attacker can directly manipulate the database as they see fit. This would, among other things, allow them to gain the admin permission within Koji. They could destroy or corrupt the database, add new builds, replace existing builds, or any number of other things.

Q: Can the attacker execute arbitrary code?

> On the hub, not that we know of.
>
> However, they could create arbitrary tasks, which would be run by the build hosts.

Q: Where can I get more help?

> You can ask questions on the koji-devel mailing list (koji-devel@fedorahosted.org).
>
> For real time communication, we have the #koji IRC channel on Freenode. The best time to ask would be during the Koji devel team "office hours", which are held each Tuesday and Thursday from 10-11am eastern time.

Summary

This is a critical security bug.

Multiple xmlrpc call handlers in Koji's hub code contain SQL injection bugs. By passing carefully constructed arguments to these calls, an unauthenticated user can issue arbitrary SQL commands to Koji's database. This gives the attacker broad ability to manipulate or destroy data.

There is no known workaround. All Koji admins are encouraged to update to a fixed version as soon as possible.

Bug fix

Note: because code fixes can take time to deploy, we recommend that all admins shut down their Koji hub instances until the fix can be applied.

We are releasing updates for several recent versions of Koji to fix this bug. The following releases all contain the fix:

- 1.16.2
- 1.15.2
- 1.14.2
- 1.13.2
- 1.12.2
- 1.11.1

Note: the legacy-py24 branch is unaffected since it is client-only (no hub).

For users who have customized their Koji code, we recommend rebasing your work onto the appropriate update release. If this is not feasible, the patch should be very easy to apply. Please see issue #1183 for the code details.

As with all changes to hub code, you must restart httpd for the changes to take effect.

Links

Fixed versions can be found at our releases page:

> https://pagure.io/koji/releases

Questions and answers about this issue

> *FAQ for CVE-2018-1002161*

2.8.2 CVE-2018-1002150

Dist repo call missing authorization check allowing filesystem manipulation

FAQ for CVE-2018-1002150

Following are answers to some questions regarding CVE-2018-1002150 for Koji. If you haven't already, you should read the *announcement*.

If you have questions not covered here or in the announcement, please ask them on the koji-devel mailing list.

> https://lists.fedorahosted.org/archives/list/koji-devel@lists.fedorahosted.org/

Q: Does this issue affect Koji clients or builders?

The issue only affects the Koji hub.

Q: How can I tell if I've been attacked?

We don't know of any exploits in the wild. However, to be safe, we will release an intrusion detection document in a few days.

Q: Where are the fixed versions?

Koji versions before 1.12.0 are unaffected
For Koji 1.12, 1.12.1 and higher includes the fix
For Koji 1.13, 1.13.1 and higher includes the fix
For Koji 1.14, 1.14.1 and higher includes the fix
For Koji 1.15, 1.15.1 and higher includes the fix
Koji 1.16.0 and higher will include the fix

You can find all of these versions on our releases page:

https://pagure.io/koji/releases

Q: What about versions before 1.12.0?

Koji versions before 1.12.0 are unaffected (they don't have the dist-repo feature). However, it would be wise to update your system to the current version.

Q: What can be done with this exploit?

The attacker can trick Koji into moving files around. These can be almost any file that the httpd user can write. The attacker could use this to corrupt Koji's file store or to reveal any secret files that the httpd user can read.

Q: Can the attacker execute arbitrary code?

Not that we know of.

Q: Where can I get more help?

You can ask questions on the koji-devel mailing list (koji-devel@fedorahosted.org).

For real time communication, we have the #koji IRC channel on Freenode. The best time to ask would be during the Koji devel team "office hours", which are held each Tuesday and Thursday from 10-11am eastern time.

Summary

This is a critical security bug.

From versions 1.12.0 to 1.15.0, the Koji hub did not perform proper access checks for the hub.distRepoMove call. By passing carefully constructed arguments to the call, an unauthenticated user can trick Koji into moving content around that it should not. This could result in corrupting any files that the httpd process can write to, or revealing any files that the httpd process can read. If the user can authenticate (at any privilege level), then they can use this mechanism to replace a file with one that they have uploaded.

Workaround

We strongly recommend that all Koji admins implement this workaround immediately. This workaround will effectively disable dist-repo functionality.

Because use of the hub.distRepoMove call requires a valid dist repo that exists on disk, exploitation can be blocked by ensuring that there are none. There are many ways this might be done. We recommend the following:

1. Move the repos-dist directory to another location (if it exists)
2. Replace it with a plain text file warning of the situation. Do not skip this step.

For example:

```
$ cd /mnt/koji
$ mv repos-dist repos-dist.old
$ echo "DO NOT REMOVE. CVE-2018-1002150" > repos-dist
$ ls -l /mnt/koji/repos-dist
-rw-r--r--. 1 root root 32 Mar 19 14:35 /mnt/koji/repos-dist
```

When applying this workaround, make sure to take both steps. If you do not, then the system will recreate the directory if anyone creates a new dist repo.

Bug fix

Note: because code fixes can take time to deploy, we strongly recommend that all admins apply the above workaround first. The workaround can be easily undone once the fix is in place.

We are releasing updates for each affected version of Koji to fix this bug. The following releases all contain the fix:

- 1.15.1
- 1.14.1
- 1.13.1
- 1.12.1

Versions prior to 1.12.0 are not vulnerable because they do not have the dist-repo feature. Also, the legacy-py24 branch is unaffected since it is client-only (no hub).

For users who have customized their Koji code, we recommend rebasing your work onto the appropriate update release. If this is not feasible, the patch should be very easy to apply. Please see issue #850 for the code details.

As with all changes to hub code, you must restart httpd for the changes to take effect.

Links

Fixed versions can be found at our releases page:

https://pagure.io/koji/releases

Questions and answers about this issue

FAQ for CVE-2018-1002150

2.8.3 CVE-2017-1002153

Koji 1.13.0 does not properly validate SCM paths.

Summary

Koji 1.13.0 does not properly validate SCM paths, allowing an attacker to work around blacklisted paths for build submission.

Bug fix

Koji versions 1.14.0 and forward contain the fix.

This bug was tracked as issue#563

Links

Fixed versions can be found at our releases page:

https://pagure.io/koji/releases

2.9 Runs Here

2.9.1 List of locations that Koji is currently in use

Please add yourself here if you run a local version of Koji. Please also describe what you are using it for.

#	Project	Site Name	Used for
0	Grid and HPC Operations Center of A.U.Th	Not-Public	Building packages for local fabric needs. Maintenance of RPM repositories. Monitoring, messaging and information system software for LCG.
1	pc V Pharmacy	Not-public	Compiling proprietary software across Fedora and RHEL. Both the software and any dependencies that are not available in Fedora and/or RHEL that may also be proprietary.
2	CERN	*	CERN no longer runs their own Koji, but does make daily use of the one run by the Scientific Computing Center of A.U.Th.
3	fedora.danny.cz	the hub is not public yet	http://sharkcz.livejournal.com/3988.html
4	Ergo Project	Koji Web Interface	Fast-Track repositories for Enterprise Linux and Fedora, FTBFS, Blog posts on Koji
5	NetNix Estonia OÜ	Internal	Packaging commercial and in-house software for RHEL/CentOS/Fedora support. Multiple instances installed, locally and customer sites.
6	Kolab Systems AG	Not Public (yet)	Packaging for ISV products
7	Lebanon Evangelical School for Boys and Girls	http://koji.lesbg.com/koji	Used to provide configuration RPMS for our three schools
8	Messinet Secure Services	http://messinet.com/koji	Building and packaging to support local infrastructure as well as incorporate software not packaged at Fedora or RPMFusion. Most packaged software is available to all. See Messinet Secure Services Fedora RPM Repository for details.
9	ACOSS France	Not-public	Building and packaging on top of RHEL and CentOS 6 for newer software version.
10	Virtual Bridges, Inc.	Not-public	Building VERDE LEAF virtualization platform/distribution.
11	Caltech	http://koji.hep.caltech.edu/koji/	multi-university CERN-based project
12	Spacewalk	http://koji.spacewalkproject.org/koji/	building Spacewalk for RHEL 5, 6, and Fedora 14, 15
13	National Nanotechnology Network Grid	http://koji.ngrid.ru/koji/	GridNNN middleware
14	TomTom	non-public	used to build rpms that aren't available in the RHEL repos or EPEL
15	Amazon	non-public	used to build RPMS for the Amazon Linux AMI
16	Open Science Grid	http://koji.chtc.wisc.edu/koji/	Used to build RPMs for the Open Science Grid software project. Software built here is deployed across 100+ universities and labs across the U.S.
17	RussianFedora	http://koji.russianfedora.ru	Building add-on packages for the RFRemix.
18	Scientific Linux	Not-public (yet)	Building Scientific Linux.
19	Katello	http://koji.katello.org/koji/	Building packages for Katello project
20	NIWA	non-public	Building packages for climate and forecasting project
21	Xenith Consulting Limited	Not-Public	Banking, Telecommunications and Embedded Systems
22	Abril Comunicações	non-public	Media / Entertainment Company. Building RPM packages for the Digital products
23	The FireServer Project	http://www.fireserver	Building packages for FireServer Project

2.10 Koji Server Bootstrap

2.10.1 Bootstrapping a new Koji build environment

These are the steps involved in populating a new Koji server with packages so that it can be used for building. This assumes that the Koji hub is up, appropriate authentication methods have been configured, the Koji repo administration daemon (`kojira`) is properly configured and running, and at least one Koji builder (`kojid`) is properly configured and running. All koji cli commands assume that the user is a Koji *admin*. If you need help with these tasks, see the ServerHowTo .

- Download all source rpms and binary rpms for the arches you're interested in
- Import all source rpms

```
$ koji import /path/to/package1.src.rpm /path/to/package2.src.rpm ...
```

If the files are on the same volume as /mnt/koji, you can use `koji import --link`, which hardlinks the files into place, avoiding the need to upload them to the hub and **very significantly** increasing import speed. When using `--link`, you must run as root. It is **highly** recommended that you use `--link`.

- Import all binary rpms using the same method as above
- Create a new tag

```
$ koji add-tag dist-foo
```

- Tag all of the packages you just imported into the tag you just created

You can use `koji list-untagged` to get a list of all of the packages you just imported.

```
$ koji list-pkgs --quiet | xargs koji add-pkg --owner <kojiuser> dist-foo
$ koji list-untagged | xargs -n 1 koji call tagBuildBypass dist-foo
```

We call the *tagBuildBypass* method instead of using `koji tag-build` because it doesn't require the builders to process *tagBuild* tasks one at a time, but does the tagging directly. This will save a significant amount of time, especially when tagging a large number of packages.

- Create a build tag with the desired arches, and the previously created tag as a parent

```
$ koji add-tag --parent dist-foo --arches "i386 x86_64 ppc ppc64" dist-foo-build
```

- Create a build target that includes the tags you've already created

```
$ koji add-target dist-foo dist-foo-build
```

- Create a *build* group associated with your build tag

```
$ koji add-group dist-foo-build build
```

- Populate the *build* group with packages that will be installed into the minimal buildroot

You can find out what the current build group for Fedora is by running `koji -s https://$ARCH.koji.fedoraproject.org/kojihub list-groups f17-build` against the Fedora Koji instance for your $ARCH. This is probably a good starting point for your minimal buildroot.

```
$ koji add-group-pkg dist-foo-build build pkg1
$ koji add-group-pkg dist-foo-build build pkg2
```

If you want to fully duplicate Fedora's group data for a tag, then it would be easier to do it in bulk – export Fedora's data and import it to your build tag.

```
$ koji -s https://$ARCH.koji.fedoraproject.org/kojihub show-groups --comps f17-build >
↪ comps.xml
$ koji import-comps comps.xml dist-foo-build
```

- regenerate the repo

```
$ koji regen-repo dist-foo-build
```

- Wait for the repo to regenerate, and you should now be able to run a build successfully.

2.11 Server How To

2.11.1 Setting Up a Koji Build System

The Koji components may **live** on separate resources as long as all resources are able to communicate. This document will cover how to setup each service individually, however, all services may **live** on the same resource.

2.11.2 Knowledge Prerequisites

- Basic understanding of SSL and authentication via certificates and/or Kerberos credentials
- Basic knowledge about creating a database in PostgreSQL and importing a schema
- Working with psql
- Basic knowledge about Apache configuration
- Basic knowledge about dnf/yum/createrepo/mock - else you'll not be able to debug problems!
- Basic knowledge about using command line
- Basic knowledge about RPM building
- Simple usage of the Koji client
- For an overview of yum, mock, Koji (and all its subcomponents), mash, and how they all work together, see the excellent slides put together by Steve Traylen at CERN.

2.11.3 Package Prerequisites

On the server (koji-hub/koji-web)

- httpd
- mod_ssl
- postgresql-server
- mod_wsgi

On the builder (koji-builder)

- mock
- setarch (for some archs you'll require a patched version)
- rpm-build
- createrepo

2.11.4 A note on filesystem space

Koji will consume copious amounts of disk space under the primary KojiDir directory (as set in the kojihub.conf file - defaults to `/mnt/koji`). However, as koji makes use of mock on the backend to actually create build roots and perform the builds in those build roots, it might come to a surprise to users that a running koji server will consume large amounts of disk space under `/var/lib/mock` and `/var/cache/mock` as well. Users should either plan the disk and filesystem allocations for this, or plan to modify the default mock build directory in the kojid.conf file. If you change the location, ensure that the new directories are owned by the group "mock" and have 02755 permission.

2.11.5 Koji Authentication Selection

Koji primarily supports Kerberos and SSL Certificate authentication. For basic koji command line access, plain user/pass combinations are possible. However, kojiweb does **not** support plain user/pass authentication and once either Kerberos or SSL Certificate authentication is enabled for kojiweb, the plain user/pass method will stop working entirely. For this reason we encourage skipping the plain user/pass method altogether and properly configuring either Kerberos or SSL Certification authentication from the start.

The decision on how to authenticate users will affect all other actions you take in setting up koji. For this reason it is a decision best made up front.

For Kerberos authentication a working Kerberos environment (the user is assumed to either already have this or know how to set it up themselves, instructions for it are not included here) and the Kerberos credentials of the initial admin user will be necessary to bootstrap the user database.

For SSL authentication SSL certificates for the xmlrpc server, for the various koji components, and one for the admin user will need to be setup (the user need not know how to create certificate chains already, we include the instructions for this below).

Setting up SSL Certificates for authentication

Certificate generation

Create the `/etc/pki/koji` directory and copy-and-paste the ssl.cnf listed here, and save it in the new directory. This configuration file is used along with the `openssl` command to generate the SSL certificates for the various koji components.

`ssl.cnf`

```
HOME                    = .
RANDFILE                = .rand

[ca]
default_ca              = ca_default

[ca_default]
```

(continues on next page)

(continued from previous page)

```
dir                     = .
certs                   = $dir/certs
crl_dir                 = $dir/crl
database                = $dir/index.txt
new_certs_dir           = $dir/newcerts
certificate             = $dir/%s_ca_cert.pem
private_key             = $dir/private/%s_ca_key.pem
serial                  = $dir/serial
crl                     = $dir/crl.pem
x509_extensions         = usr_cert
name_opt                = ca_default
cert_opt                = ca_default
default_days            = 3650
default_crl_days        = 30
default_md              = sha256
preserve                = no
policy                  = policy_match

[policy_match]
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

[req]
default_bits            = 1024
default_keyfile         = privkey.pem
default_md              = sha256
distinguished_name      = req_distinguished_name
attributes              = req_attributes
x509_extensions         = v3_ca # The extensions to add to the self signed cert
string_mask             = MASK:0x2002

[req_distinguished_name]
countryName                     = Country Name (2 letter code)
countryName_default             = AT
countryName_min                 = 2
countryName_max                 = 2
stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = Vienna
localityName                    = Locality Name (eg, city)
localityName_default            = Vienna
0.organizationName              = Organization Name (eg, company)
0.organizationName_default      = My company
organizationalUnitName          = Organizational Unit Name (eg, section)
commonName                      = Common Name (eg, your name or your server\'s␣
↪hostname)
commonName_max                  = 64
emailAddress                    = Email Address
emailAddress_max                = 64

[req_attributes]
challengePassword               = A challenge password
challengePassword_min           = 4
challengePassword_max           = 20
```

(continues on next page)

(continued from previous page)

```
unstructuredName                = An optional company name

[usr_cert]
basicConstraints                = CA:FALSE
nsComment                       = "OpenSSL Generated Certificate"
subjectKeyIdentifier            = hash
authorityKeyIdentifier          = keyid,issuer:always

[v3_ca]
subjectKeyIdentifier            = hash
authorityKeyIdentifier          = keyid:always,issuer:always
basicConstraints                = CA:true
```

Although it is not required, it is recommended that you edit the default values in the `[req_distinguished_name]` section of the configuration to match the information for your own server. This will allow you to accept most of the default values when generating certificates later. The other sections can be left unedited.

Generate CA

The CA is the Certificate Authority. It's the key/cert pair used to sign all the other certificate requests. When configuring the various koji components, both the client CA and the server CA will be a copy of the CA generated here. The CA certificate will be placed in the `/etc/pki/koji` directory and the certificates for the other components will be placed in the `/etc/pki/koji/certs` directory. The `index.txt` file which is created is a database of the certificates generated and can be used to view the information for any of the certificates simply by viewing the contents of `index.txt`.

```
cd /etc/pki/koji/
mkdir {certs,private,confs}
touch index.txt
echo 01 > serial
openssl genrsa -out private/koji_ca_cert.key 2048
openssl req -config ssl.cnf -new -x509 -days 3650 -key private/koji_ca_cert.key \
-out koji_ca_cert.crt -extensions v3_ca
```

The last command above will ask you to confirm a number of items about the certificate you are generating. Presumably you already edited the defaults for the country, state/province, locale, and organization in the `ssl.cnf` file and you only needed to hit enter. It's the organizational unit and the common name that we will be changing in the various certs we create. For the CA itself, these fields don't have a hard requirement. One suggestion for this certificate is to use the FQDN of the server.

If you are trying to automate this process via a configuration management tool, you can create the cert in one command with a line like this:

```
openssl req -config ssl.cnf -new -x509 \
-subj "/C=US/ST=Oregon/L=Portland/O=IT/CN=koji.example.com" \
-days 3650 -key private/koji_ca_cert.key -out koji_ca_cert.crt -extensions v3_ca
```

Generate the koji component certificates and the admin certificate

Each koji component needs its own certificate to identify it. Two of the certificates (kojihub and kojiweb) are used as server side certificates that authenticate the server to the client. For this reason, you want the common name on both of those certs to be the fully qualified domain name of the web server they are running on so that clients don't complain

about the common name and the server not being the same. You can set the OU for these two certificates to be kojihub and kojiweb for identification purposes.

For the other certificates (kojira, kojid, the initial admin account, and all user certificates), the cert is used to authenticate the client to the server. The common name for these certs should be set to the login name for that specific component. For example the common name for the kojira cert should be set to kojira so that it matches the username. The reason for this is that the common name of the cert will be matched to the corresponding user name in the koji database. If there is not a username in the database which matches the CN of the cert the client will not be authenticated and access will be denied.

When you later use `koji add-host` to add a build machine into the koji database, it creates a user account for that host even though the user account doesn't appear in the user list. The user account created must match the common name of the certificate which that component uses to authenticate with the server. When creating the kojiweb certificate, you'll want to remember exactly what values you enter for each field as you'll have to regurgitate those into the /etc/koji-hub/hub.conf file as the ProxyDNs entry.

When you need to create multiple certificates it may be convenient to create a loop or a script like the on listed below and run the script to create the certificates. You can simply adjust the number of kojibuilders and the name of the admin account as you see fit. For much of this guide, the admin account is called `kojiadmin`.

```
#!/bin/bash
# if you change your certificate authority name to something else you will
# need to change the caname value to reflect the change.
caname=koji

# user is equal to parameter one or the first argument when you actually
# run the script
user=$1

openssl genrsa -out private/${user}.key 2048
cat ssl.cnf | sed 's/insert_hostname/'${user}'/'> ssl2.cnf
openssl req -config ssl2.cnf -new -nodes -out certs/${user}.csr -key private/${user}.
↪key
openssl ca -config ssl2.cnf -keyfile private/${caname}_ca_cert.key -cert ${caname}_ca_
↪cert.crt \
    -out certs/${user}.crt -outdir certs -infiles certs/${user}.csr
cat certs/${user}.crt private/${user}.key > ${user}.pem
mv ssl2.cnf confs/${user}-ssl.cnf
```

Generate a PKCS12 user certificate (for web browser)

This is only required for user certificates.

```
openssl pkcs12 -export -inkey private/${user}.key -in certs/${user}.crt \
    -CAfile ${caname}_ca_cert.crt -out certs/${user}_browser_cert.p12
```

When generating certs for a user, the user will need the `${user}.pem`, the `${caname}_ca_cert.crt`, and the `${user}_browser_cert.p12` files which were generated above. The ${user}.pem file would normally be installed as `~/.fedora.cert`, the `${caname}_ca_cert.crt` file would be installed as both `~/.fedora-upload-ca.cert` and `~/.fedora-server-ca.cert`, and the user would import the `${user}_brower_cert.p12` into their web browser as a personal certificate.

Copy certificates into ~/.koji for kojiadmin

You're going to want to be able to send admin commands to the kojihub. In order to do so, you'll need to use the newly created certificates to authenticate with the hub. Create the kojiadmin user then copy the certificates for the koji CA and the kojiadmin user to `~/.koji`:

```
kojiadmin@localhost$ mkdir ~/.koji
kojiadmin@localhost$ cp /etc/pki/koji/kojiadmin.pem ~/.koji/client.crt   # NOTE: It␣
↪is IMPORTANT you use the PEM and NOT the CRT
kojiadmin@localhost$ cp /etc/pki/koji/koji_ca_cert.crt ~/.koji/clientca.crt
kojiadmin@localhost$ cp /etc/pki/koji/koji_ca_cert.crt ~/.koji/serverca.crt
```

Note: See /etc/koji.conf for the current system wide koji client configuration. Copy /etc/koji.conf to ~/.koji/config if you wish to change the config on a per user basis.

Setting up Kerberos for authentication

The initial configuration of a kerberos service is outside the scope of this document, however there are a few specific things required by koji.

DNS

The koji builders (kojid) use DNS to find the kerberos servers for any given realm.

```
_kerberos._udp    IN SRV  10 100 88 kerberos.EXAMPLE.COM.
```

The trailing dot denotes DNS root and is needed if FQDN is used.

Principals and Keytabs

It should be noted that in general you will need to use the fully qualified domain name of the hosts when generating the keytabs for services.

You will need the following principals extracted to a keytab for a fully kerberized configuration, the requirement for a host key for the koji-hub is currently hard coded into the koji client.

host/kojihub@EXAMPLE.COM Used by the koji-hub server when communicating with the koji client

HTTP/kojiweb@EXAMPLE.COM Used by the koji-web server when performing a negotiated Kerberos authentication with a web browser. This is a service principal for Apache's mod_auth_gssapi.

koji/kojiweb@EXAMPLE.COM Used by the koji-web server during communications with the koji-hub. This is a user principal that will authenticate koji-web to Kerberos as "koji/kojiweb@EXAMPLE.COM". Koji-web will proxy the mod_auth_gssapi user information to koji-hub (the `ProxyPrincipals` koji-hub config option).

koji/kojira@EXAMPLE.COM Used by the kojira server during communications with the koji-hub

compile/builder1@EXAMPLE.COM Used on builder1 to communicate with the koji-hub

2.11.6 PostgreSQL Server

Once the authentication scheme has been setup your will need to install and configure a PostgreSQL server and prime the database which will be used to hold the koji users.

Configuration Files

- `/var/lib/pgsql/data/pg_hba.conf`
- `/var/lib/pgsql/data/postgresql.conf`

Install PostgreSQL

Install the `postgresql-server` package:

```
# yum install postgresql-server
```

Initialize PostgreSQL DB:

The following commands will initialize PostgreSQL and will start the database service

```
root@localhost$ postgresql-setup initdb
root@localhost$ systemctl enable postgresql
root@localhost$ systemctl start postgresql
```

Setup User Accounts:

The following commands will setup the `koji` account and assign it a password

```
root@localhost$ useradd koji
root@localhost$ passwd koji
```

Setup PostgreSQL and populate schema:

The following commands will:

- create the koji user within PostgreSQL
- create the koji database within PostgreSQL
- set a password for the koji user
- create the koji schema using the provided `/usr/share/doc/koji*/docs/schema.sql` file from the `koji` package.

```
root@localhost$ su - postgres
postgres@localhost$ createuser --no-superuser --no-createrole --no-createdb koji
postgres@localhost$ createdb -O koji koji
postgres@localhost$ psql -c "alter user koji with encrypted password 'mypassword';"
postgres@localhost$ logout
root@localhost$ yum -y install koji
root@localhost$ su - koji
```

(continues on next page)

(continued from previous page)

```
koji@localhost$ psql koji koji < /usr/share/doc/koji*/docs/schema.sql
koji@localhost$ exit
```

Note: When issuing the command to import the psql schema into the new database it is important to ensure that the directory path /usr/share/doc/koji*/docs/schema.sql remains intact and is not resolved to a specific version of koji. In test it was discovered that when the path is resolved to a specific version of koji then not all of the tables were created correctly.

Note: When issuing the command to import the psql schema into the new database it is important to ensure that you are logged in as the koji database owner. This will ensure all objects are owned by the koji database user. Upgrades may be difficult if this was not done correctly.

Authorize Koji-web and Koji-hub resources

Note: In this example, Koji-web and Koji-hub are running on localhost.

`/var/lib/pgsql/data/pg_hba.conf` These settings need to be valid and inline with other services configurations. Please note, the first matching auth line is used so this line must be above any other potential matches. Add:

```
#TYPE   DATABASE    USER    CIDR-ADDRESS      METHOD
host    koji        koji    127.0.0.1/32      trust
host    koji        koji     ::1/128          trust
```

It may also be necessary to add an entry for your machine's external IP address:

```
host    koji        koji    $IP_ADDRESS/32    trust
```

You can also use UNIX socket access. The DBHost variable must be unset to use this method. Add:

```
local   koji        apache                            trust
local   koji        koji                              trust
```

Note: To enforce password based logins to the database, change `trust` to `md5`.

```
#TYPE   DATABASE    USER    CIDR-ADDRESS      METHOD
host    koji        koji    127.0.0.1/32      md5
host    koji        koji     ::1/128          md5
host    koji        koji    $IP_ADDRESS/32    md5
```

Make auth changes live: You must reload the PostgreSQL configuration for these changes to become active.

```
root@localhost$ systemctl reload postgresql
```

Bootstrapping the initial koji admin user into the PostgreSQL database

The initial admin user must be manually added to the user database using sql commands. Once added and given admin privilege, you may add additional users and change privileges of those users via the koji command line tool's administrative commands.

However, if you decided to use the simple user/pass method of authentication, then any password setting/changing must be done manually via sql commands as there is no password manipulation support exposed through the koji tools.

The sql commands you need to use vary by authentication mechanism.

Set User/Password Authentication

```
root@localhost$ su - koji
koji@localhost$ psql
koji=> insert into users (name, password, status, usertype) values ('admin-user-name',
↪ 'admin-password-in-plain-text', 0, 0);
```

Kerberos authentication

The process is very similar to user/pass except you would replace the first insert above with this:

```
root@localhost$ su - koji
koji@localhost$ psql
koji=> insert into users (name, krb_principal, status, usertype) values ('admin-user-
↪name', 'admin@EXAMPLE.COM', 0, 0);
```

SSL Certificate authentication

There is no need for either a password or a Kerberos principal, so this will suffice:

```
root@localhost$ su - koji
koji@localhost$ psql
koji=> insert into users (name, status, usertype) values ('admin-user-name', 0, 0);
```

Give yourself admin permissions

The following command will give the user admin permissions. In order to do this you will need to know the ID of the user.

```
koji=> insert into user_perms (user_id, perm_id, creator_id) values (<id of user␣
↪inserted above>, 1, <id of user inserted above>);
```

Note: If you do not know the ID of the admin user, you can get the ID by running the query:

```
koji=> select * from users;
```

You can't actually log in and perform any actions until kojihub is up and running in your web server. In order to get to that point you still need to complete the authentication setup and the kojihub configuration. If you wish to access koji via a web browser, you will also need to get kojiweb up and running.

Set Database To Listen On All Addresses

The koji-hub service will attempt to connect to the database server in the manner you configure. If you use the system hostname, then the database will need to be available on that address. To configure this please perform the following:

1. Edit /var/lib/pgsql/data/postgresql.conf
2. Set listen_address

```
listen_addresses = '*'
```

3. Reload the postgresql service

   :: systemctl restart postgresql

2.11.7 Koji Hub

Koji-hub is the center of all Koji operations. It is an XML-RPC server running under mod_wsgi in the Apache httpd. koji-hub is passive in that it only receives XML-RPC calls and relies upon the build daemons and other components to initiate communication. Koji-hub is the only component that has direct access to the database and is one of the two components that have write access to the file system.

Configuration Files

- `/etc/koji-hub/hub.conf`
- `/etc/httpd/conf/httpd.conf`
- `/etc/httpd/conf.d/kojihub.conf`
- `/etc/httpd/conf.d/ssl.conf` (when using ssl auth)

Install koji-hub

Install the `koji-hub` package along with mod_ssl:

```
# yum install koji-hub mod_ssl
```

Required Configuration

/etc/httpd/conf/httpd.conf

The apache web server has two places that it sets maximum requests a server will handle before the server restarts. The xmlrpc interface in kojihub is a python application, and processes can sometimes grow outrageously large when it doesn't reap memory often enough. As a result, it is strongly recommended that you set both instances of MaxRequestsPerChild in httpd.conf to something reasonable in order to prevent the server from becoming overloaded and crashing (at 100 the httpd processes will grow to about 75MB resident set size before respawning).

```
<IfModule prefork.c>
...
MaxRequestsPerChild  100
</IfModule>
<IfModule worker.c>
...
MaxRequestsPerChild  100
</IfModule>
```

/etc/httpd/conf.d/kojihub.conf

The koji-hub package provides this configuration file. You will need to modify it based on your authentication type. Instructions are contained within the file and should be simple to follow.

/etc/httpd/conf.d/ssl.conf

If using SSL you will also need to add the needed SSL options for apache. These options should point to where the certificates are located on the hub.

```
SSLCertificateFile /etc/pki/koji/certs/kojihub.crt
SSLCertificateKeyFile /etc/pki/koji/private/kojihub.key
SSLCertificateChainFile /etc/pki/koji/koji_ca_cert.crt
SSLCACertificateFile /etc/pki/koji/koji_ca_cert.crt
SSLVerifyClient require
SSLVerifyDepth  10
```

/etc/koji-hub/hub.conf

This file contains the configuration information for the hub. You will need to edit this configuration to point Koji Hub to the database you are using and to setup Koji Hub to utilize the authentication scheme you selected in the beginning.

```
DBName = koji
DBUser = koji
DBPass = mypassword
DBHost = db.example.com
KojiDir = /mnt/koji
LoginCreatesUser = On
KojiWebURL = http://kojiweb.example.com/koji
```

If kojihub is running on the same server as the koji db, then DBHost should be set to 127.0.0.1

Authentication Configuration

/etc/koji-hub/hub.conf

If using Kerberos, these settings need to be valid and inline with other services configurations.

```
AuthPrincipal host/kojihub@EXAMPLE.COM
AuthKeytab /etc/koji.keytab
ProxyPrincipals koji/kojiweb@EXAMPLE.COM
HostPrincipalFormat compile/%s@EXAMPLE.COM
```

If using SSL auth, these settings need to be valid and inline with other services configurations for kojiweb to allow logins.

ProxyDNs should be set to the DN of the kojiweb certificate. The exact format depends on your mod_ssl version.

For mod_ssl < 2.3.11 use:

```
DNUsernameComponent = CN
ProxyDNs = /C=US/ST=Massachusetts/O=Example Org/OU=kojiweb/CN=example/
↪emailAddress=example@example.com
```

However, for mod_ssl >= 2.3.11 use:

```
DNUsernameComponent = CN
ProxyDNs = CN=example.com,OU=kojiweb,O=Example Org,ST=Massachusetts,C=US
```

Note: More details on this format change, including handling of special characters, can be found in the Apache mod_ssl documentation. See LegacyDNStringFormat there.

Koji filesystem skeleton

Above in the `kojihub.conf` file we set KojiDir to `/mnt/koji`. For certain reasons, if you change this, you should make a symlink from `/mnt/koji` to the new location (note: this is a bug and should be fixed eventually). However, before other parts of koji will operate properly, we need to create a skeleton filesystem structure for koji as well as make the file area owned by apache so that the xmlrpc interface can write to it as needed.

```
cd /mnt
mkdir koji
cd koji
mkdir {packages,repos,work,scratch,repos-dist}
chown apache.apache *
```

SELinux Configuration

If running in Enforcing mode

- you will need to allow apache to connect to the postgreSQL server
- you will need to allow apache to write some files to disk

Even if you are not currently running in Enforcing mode, it is still recommended to configure the SELinux settings so that there are no future issues with SELinux if Enforcing mode is enabled later on.

```
root@localhost$ setsebool -P httpd_can_network_connect_db=1 allow_httpd_anon_write=1
root@localhost$ chcon -R -t public_content_rw_t /mnt/koji/*
```

If you've placed `/mnt/koji` on an NFS share you may also need to set `httpd_use_nfs`.

Check Your Configuration

At this point, you can now restart apache and you should have at least minimal operation. The admin user should be able to connect via the command line client, add new users, etc. It's possible at this time to undertake initial administrative steps such as adding users and hosts to the koji database.

So we will need a working client to test with.

2.11.8 Koji cli - The standard client

The koji cli is the standard client. It can perform most tasks and is essential to the successful use of any koji environment.

Ensure that your client is configured to work with your server. The system-wide koji client configuration file is `/etc/koji.conf`, and the user-specific one is in `~/.koji/config`. You may also use the `-c` option when using the Koji client to specify an alternative configuration file.

If you are using SSL for authentication, you will need to edit the Koji client configuration to tell it which URLs to use for the various Koji components and where their SSL certificates can be found.

For a simple test, all we need is the `server` and authentication sections.

```
[koji]

;url of XMLRPC server
server = http://koji-hub.example.com/kojihub

;url of web interface
weburl = http://koji-web.example.com/koji

;url of package download site
topurl = http://koji-filesystem.example.com/kojifiles

;path to the koji top directory
topdir = /mnt/koji

; configuration for Kerberos authentication

;the service name of the principal being used by the hub
;krbservice = host

; configuration for SSL athentication

;client certificate
cert = ~/.koji/client.crt

;certificate of the CA that issued the client certificate
ca = ~/.koji/clientca.crt

;certificate of the CA that issued the HTTP server certificate
serverca = ~/.koji/serverca.crt
```

The following command will test your login to the hub:

```
root@localhost$ koji moshimoshi
```

2.11.9 Koji Web - Interface for the Masses

Koji-web is a set of scripts that run in mod_wsgi and use the Cheetah templating engine to provide an web interface to Koji. koji-web exposes a lot of information and also provides a means for certain operations, such as cancelling builds.

Configuration Files

- /etc/httpd/conf.d/kojiweb.conf
- /etc/httpd/conf.d/ssl.conf
- /etc/kojiweb/web.conf

Install Koji-Web

Install the koji-web package along with mod_ssl:

```
# yum install koji-web mod_ssl
```

Required Configuration

/etc/httpd/conf.d/kojiweb.conf

The koji-web package provides this configuration file. You will need to modify it based on your authentication type. Instructions are contained within the file and should be simple to follow.

/etc/httpd/conf.d/ssl.conf

If you are using SSL you will need to add the needed SSL options for apache.

```
SSLVerifyClient require
SSLVerifyDepth  10
```

/etc/kojiweb/web.conf

You will need to edit the kojiweb configuration file to tell kojiweb which URLs it should use to access the hub, the koji packages and its own web interface. You will also need to tell kojiweb where it can find the SSL certificates for each of these components. If you are using SSL authentication, the "WebCert" line below must contain both the public **and** private key. You will also want to change the last line in the example below to a unique password.

```
[web]
SiteName = koji
# KojiTheme =

# Necessary urls
KojiHubURL = https://koji-hub.example.com/kojihub
KojiFilesURL = http://koji-filesystem.example.com/kojifiles

## Kerberos authentication options
; WebPrincipal = koji/web@EXAMPLE.COM
; WebKeytab = /etc/httpd.keytab
; WebCCache = /var/tmp/kojiweb.ccache

## SSL authentication options
; WebCert = /etc/pki/koji/koji-web.pem
; ClientCA = /etc/pki/koji/ca_cert.crt
; KojiHubCA = /etc/pki/koji/ca_cert.crt
```

(continues on next page)

(continued from previous page)

```
LoginTimeout = 72

# This must be set before deployment
#Secret = CHANGE_ME

LibPath = /usr/share/koji-web/lib
```

Filesystem Configuration

You'll need to make `/mnt/koji/` web-accessible, either here, on the hub, or on another web server altogether.

This URL will go into various clients such as: * `/etc/kojiweb/web.conf` as KojiFilesURL * `/etc/kojid/kojid.conf` as topurl * `/etc/koji.conf` as topurl

```
Alias /kojifiles/ /mnt/koji/
<Directory "/mnt/koji/">
    Options Indexes
    AllowOverride None
    # Apache < 2.4
    #   Order allow,deny
    #   Allow from all
    # Apache >= 2.4
    Require all granted
</Directory>
```

Wherever you configure this, please go back and set it correctly in `/etc/kojiweb/web.conf` now.

Web interface now operational

At this point you should be able to point your web browser at the kojiweb URL and be presented with the koji interface. Many operations should work in read only mode at this point, and any configured users should be able to log in.

2.11.10 Koji Daemon - Builder

Kojid is the build daemon that runs on each of the build machines. Its primary responsibility is polling for incoming build requests and handling them accordingly. Koji also has support for tasks other than building such as creating livecd images or raw disk images, and kojid is responsible for handling these tasks as well. The kojid service uses mock for creating pristine build environments and creates a fresh one for every build, ensuring that artifacts of build processes cannot contaminate each other. All of kojid is written in Python and communicates with koji-hub via XML-RPC.

Configuration Files

- `/etc/kojid/kojid.conf` - Koji Daemon Configuration
- `/etc/sysconfig/kojid` - Koji Daemon Switches

Install kojid

Install the `koji-builder` package:

```
# yum install koji-builder
```

Required Configuration

Add the host entry for the koji builder to the database

You will now need to add the koji builder to the database so that they can be utilized by koji hub. Make sure you do this before you start kojid for the first time, or you'll need to manually remove entries from the sessions and users table before it can be run successfully.

```
kojiadmin@localhost$ koji add-host kojibuilder1.example.com i386 x86_64
```

The first argument used after the `add-host` command should the hostname of the builder. The second argument is used to specify the architecture which the builder uses.

/etc/kojid/kojid.conf

The configuration file for each koji builder must be edited so that the line below points to the URL for the koji hub. The user tag must also be edited to point to the username used to add the koji builder.

```
; The URL for the xmlrpc server
server=http://hub.example.com/kojihub

; the username has to be the same as what you used with add-host
; in this example follow as below
user = kojibuilder1.example.com
```

The koji filesystem may also be needed over http. Set this as it was configured about.

```
# The URL for the file access
topurl=http://koji-filesystem.example.com/kojifiles
```

This item may be changed, but may not be the same as KojiDir on the `kojihub.conf` file (although it can be something under KojiDir, just not the same as KojiDir)

```
; The directory root for temporary storage
workdir=/tmp/koji
```

The root of the koji build directory (i.e., `/mnt/koji`) must be mounted on the builder. A Read-Only NFS mount is the easiest way to handle this.

```
# The directory root where work data can be found from the koji hub
topdir=/mnt/koji
```

Authentication Configuration (SSL certificates)

/etc/kojid/kojid.conf

If you are using SSL, these settings need to be edited to point to the certificates you generated at the beginning of the setup process.

```
;client certificate
; This should reference the builder certificate we created on the kojihub CA, for␣
↪kojibuilder1.example.com
; ALSO NOTE: This is the PEM file, NOT the crt
cert = /etc/kojid/kojid.pem

;certificate of the CA that issued the client certificate
ca = /etc/kojid/koji_ca_cert.crt

;certificate of the CA that issued the HTTP server certificate
serverca = /etc/kojid/koji_ca_cert.crt
```

It is important to note that if your builders are hosted on separate machines from koji hub and koji web, you will need to scp the certificates mentioned in the above configuration file from the `/etc/kojid/` directory on koji hub to the `/etc/koji/` directory on the local machine so that the builder can be authenticated.

Authentication Configuration (Kerberos)

/etc/kojid/kojid.conf

If using Kerberos, these settings need to be valid and inline with other services configurations.

```
; the username has to be the same as what you used with add-host
;user =

host_principal_format=compile/%s@EXAMPLE.COM
```

By default it will look for the Kerberos keytab in `/etc/kojid/kojid.keytab`

Note: Kojid will not attempt kerberos authentication to the koji-hub unless the username field is commented out

Source Control Configuration

/etc/kojid/kojid.conf

The *allowed_scms* setting controls which source control systems the builder will accept. It is a space-separated list of entries in one of the following forms:

```
hostname:path[:use_common[:source_cmd]]
!hostname:path
```

where

hostname is a glob pattern matched against SCM hosts.

path is a glob pattern matched against the SCM path.

use_common is boolean setting (yes/no, on/off, true/false) that indicates whether koji should also check out /common from the SCM host. The default is on.

source_cmd is a shell command to be run before building the srpm, with commas instead of spaces. It defaults to `make,sources`.

The second form (`!hostname:path`) is used to explicitly block a host:path pattern. In particular, it provides the option to block specific subtrees of a host, but allow from it otherwise

```
allowed_scms=
    !scm-server.example.com:/blocked/path/*
    scm-server.example.com:/repo/base/repos*/:no
    alt-server.example.com:/repo/dist/repos*/:no:fedpkg,sources
```

The explicit block syntax was added in version 1.13.0.

Add the host to the createrepo channel

Channels are a way to control which builders process which tasks. By default hosts are added to the ''default" channel. At least some build hosts also needs to be added to the ''createrepo" channel so there will be someone to process repo creation tasks initiated by kojira.

```
kojiadmin@localhost$ koji add-host-to-channel kojibuilder1.example.com createrepo
```

A note on capacity

The default capacity of a host added to the host database is 2. This means that once the load average on that machine exceeds 2, kojid will not accept any additional tasks. This is separate from the maxjobs item in the configuration file. Before kojid will accept a job, it must pass both the test to ensure the load average is below capacity and that the current number of jobs it is already processing is less than maxjobs. However, in today's modern age of quad core and higher CPUs, a load average of 2 is generally insufficient to fully utilize hardware.

```
koji edit-host --capacity=16 kojibuilder1.example.com
```

The koji-web interface also offers the ability to edit this value to admin accounts.

Start Kojid

Once the builder has been added to the database you must start kojid

```
root@localhost$ service kojid start
```

Check `/var/log/kojid.log` to verify that kojid has started successfully. If the log does not show any errors then the koji builder should be up and ready. You can check this by pointing your web browser to the web interface and clicking on the hosts tab. This will show you a list of builders in the database and the status of each builder.

2.11.11 Kojira - Dnf|Yum repository creation and maintenance

Configuration Files

- `/etc/kojira/kojira.conf` - Kojira Daemon Configuration
- `/etc/sysconfig/kojira` - Kojira Daemon Switches

Install kojira

Install the `koji-utils` package:

```
# yum install koji-utils
```

Required Configuration

Add the user entry for the kojira user

The kojira user requires the `repo` permission to function.

```
kojiadmin@localhost$ koji add-user kojira
kojiadmin@localhost$ koji grant-permission repo kojira
```

`/etc/kojira/kojira.conf` This needs to point at your koji-hub.

```
; The URL for the xmlrpc server
server=http://koji-hub.example.com/kojihub
```

Additional Notes

- Kojira needs read-write access to `/mnt/koji`.
- There should only be one instance of kojira running at any given time.
- It is not recommended that kojira run on the builders, as builders only should require read-only access to `/mnt/koji`.
- Kojira may need to be restarted when new tags are added in order to detect those tags correctly.

Authentication Configuration

/etc/kojira/kojira.conf

If using SSL, these settings need to be valid.

```
;client certificate
; This should reference the kojira certificate we created above
cert = /etc/pki/koji/kojira.pem

;certificate of the CA that issued the client certificate
ca = /etc/pki/koji/koji_ca_cert.crt

;certificate of the CA that issued the HTTP server certificate
serverca = /etc/pki/koji/koji_ca_cert.crt
```

If using Kerberos, these settings need to be valid.

```
; For Kerberos authentication
; the principal to connect with
principal=koji/kojira@EXAMPLE.COM
; The location of the keytab for the principal above
keytab=/etc/kojira.keytab
```

/etc/sysconfig/kojira The local user kojira runs as needs to be able to read and write to `/mnt/koji/repos/`. If the volume that directory resides on is root-squashed or otherwise unmodifiable by root, you can set `RUNAS=` to a user that has the required privileges.

Start Kojira

```
root@localhost$ service kojira start
```

Check `/var/log/kojira/kojira.log` to verify that kojira has started successfully.

2.11.12 Bootstrapping the Koji build environment

For instructions on importing packages and preparing Koji to run builds, see *Server Bootstrap*.

For instructions on using External Repos and preparing Koji to run builds, see *External Repo Server Bootstrap*.

Useful scripts and config files for setting up a Koji instance are available here.

2.11.13 Minutia and Miscellany

Please see *KojiMisc* for additional details and notes about operating a koji server.

2.12 Using the koji build system

2.12.1 Using Koji in Fedora

The Koji Build System is Fedora's RPM buildsystem. Packagers use the koji client to request package builds and get information about the buildsystem. Koji runs on top of Mock to build RPM packages for specific architectures and ensure that they build correctly.

There is also a simplified Chinese edition.

Installing Koji

Installing the Koji CLI

Everything you need to use Koji (and be a Fedora contributor) can be installed in a single step:

```
dnf install fedora-packager
```

fedora-packager provides useful scripts to help maintain and setup your koji environment. Additionally, it includes dependencies on the Koji CLI, so it will be installed when you install `fedora-packager`. The command is called `koji` and is included in the main koji package. By default the koji tool authenticates to the central server using Kerberos. However SSL and username/password authentications are available. You will need to have a valid authentication token to use many features. However, many of the read-only commands will work without authentication.

Alternatively, koji CLI is now also available via:

- Project releases tarballs Preferred way is to use your distribution's mechanism instead, as it will also contain appropriate configuration files.

- PyPi There is only client/API part and it is mostly usable for people who wants some more advanced client-side scripting in virtualenv's, so API-only access is not sufficient for them or who can profit from some utilities in e.g. basic `koji` library.
- Actual development version via Pagure's git: `git clone https://pagure.io/koji.git`

Fedora Account System (FAS2) Setup

In order to interface with the koji server, maintainers will need to run

```
/usr/bin/fedora-packager-setup
```

Each user on a system will need to run fedora-packager-setup if they wish to use Koji to build Fedora packages. Each user has their own certificates that authenticate them.

Fedora Certificates

Koji uses three certificates:

`~/.fedora.cert` (specific to the Fedora Maintainer) : This cert is generated from running `fedora-cert -n`. It should have been generated when you became maintainer. You may need to refresh it when it expires by running `fedora-cert -n` again. You can check if it has expired with `fedora-cert -v`.

the following are downloaded automatically by fedora-packager-setup and don't need to be manually setup

`~/.fedora-upload-ca.cert` (The certificate for the Certificate Authority used to sign the user keys.) : It can be manually downloaded from here or `fedora-packager-setup or fedora-cert -n` should fetch it. using the CLI is preferred. `~/.fedora-server-ca.cert` (The certificate for the Certificate Authority used to sign the build system's server keys.) : It can be downloaded manually from here or `fedora-packager-setup` should fetch it. This certificate may also be needed to let https koji URLs resolve without untrusted-CA warnings.

Warning: If you're using RHEL6, an incompatibility between RHEL6's openssl and nss causes certificates downloaded from fas to fail to work with some fedpkg tools. Bug 631000 rhel6 openssl creates PKCS#8 encoded PEM RSA private key files, nss can't read them. The cert can be made compatible using this command: *openssl x509 -in ~/.fedora.cert -text; echo; openssl rsa -in ~/.fedora.cert) > fedora.cert.new*

Warning: You can also have problem in Fedora/RHEL if you are going to use GSSAPI authentication. These distributions have changed default *rdns=false* in /etc/krb5.conf. If you encounter *requests_kerberos.exceptions.MutualAuthenticationError: Unable to authenticate <Response [200]>* error, maybe you are hitting this problem. More info in pagure issue.

Koji Config

The global local client configuration file for koji is `/etc/koji.conf`. You should not need to change this from the defaults for building Fedora packages.These will allow you to use the primary build system as well as secondary arch build systems.

The web interface

The primary interface for viewing Koji data is a web application. It is available at https://koji.fedoraproject.org/koji/ . Most of the interface is read-only, but with sufficient privileges, you can log in and perform some additional actions. For example:

- Cancel a build
- Resubmit a failed task
- Setup a notification

Those with admin privileges will find additional actions, such as:

- Create/Edit/Delete a tag
- Create/Edit/Delete a target
- Enable/Disable a build host

The web site utilizes SSL authentication. In order to log in you will need a valid SSL certificate and your web browser will need to be configured to trust the SSL cert. Instructions on how to do this are printed when running `fedora-packager-setup --with-browser-cert`.

Installing SSL Certificates in Firefox

Once you have created your FAS account, generated your certificate in the form posted in the link above and ran `fedora-packager-setup --with-browser-cert`, you will need to import it into your web browser. You can do this in Firefox by doing the following:

1. Launch Firefox and click on the **Edit** menu from the toolbar
2. Select **Preferences** in the sub-menu which appears.
3. This should open the **Preferences** window where you can switch to the **Advanced** section
4. In the **Advanced** section switch to the **Encryption** tab
5. Click on the **View Certificates** button and the Certificates window will appear
6. Switch to the **Your Certificates** tab and click on the **Import** button
7. Point to where your Fedora Certificate is located and click **Open** (fedora-packager-setup will have told you where it was saved and will have asked you to set a password for the cert)

You should now be able to see your Fedora Certificate listed under **Your Certificates** and you should be able to authenticate with the koji web interface.

Installing SSL Certificates in Chromium

Chromium uses the NSS Shared DB, you will need the nss-tools package installed.

```
pk12util -d sql:$HOME/.pki/nssdb -i fedora-browser-cert.p12
```

Notifications

Koji supports a limited number of email notifications:

- build notifications: when builds complete or fail

- tag notifications: when builds are tagged or untagged

These mails are sent to:

- the owner of the build in question
- (for tag notifications) the owner of the package for the tag
- any user who as subscribed to notifications for that package or tag

Users can manage their notification subscriptions in the web interface. To do so, they need to be logged in. The main page (Summary) will list their subscriptions at the bottom. Each entry includes an "edit" and "delete" link. Below that table is an "Add a notification" link for adding new notifications.

Starting in Koji version 1.16.0, users can also manage these subscriptions on the command line. The relevant commands are:

- add-notification
- edit-notification
- list-notifications
- remove-notification

Building with fedpkg targets

Every push is automatically tagged via git. All you have done to build the package is to run,

```
fedpkg build
```

This will trigger a build request for the branch. Easy!

It is also possible to target a specific koji tag as follows:

```
fedpkg build --target TARGET
```

for example, if building on rawhide against a special tag created by rel-eng for updating API for many packages, e.g. `dist-f14-python` you would use the following:

```
fedpkg build --target 'dist-f14-python'
```

Chained builds

Sometimes you want to make sure than one build succeeded before launching the next one, for example when you want to rebuild a package against a just rebuilt dependency. In that case you can use a chain build with:

```
fedpkg chain-build libwidget libgizmo
```

The current package is added to the end of the CHAIN list. Colons (:) can be used in the CHAIN parameter to define groups of packages. Packages in any single group will be built in parallel and all packages in a group must build successfully and populate the repository before the next group will begin building. For example:

```
fedpkg chain-build libwidget libaselib : libgizmo :
```

will cause libwidget and libaselib to be built in parallel, followed by libgizmo and then the correct directory package. If no groups are defined, packages will be built sequentially.

If a build fail, following builds are cancelled but the builds that already succeeded are pushed to the repository.

Scratch Builds

Sometimes it is useful to be able to build a package against the buildroot but without actually including it in the release. This is called a scratch build. The following section covers using koji directly as well as the fedpkg tool to do scratch builds. To create a scratch build from changes you haven't committed, do the following:

```
rpmbuild -bs foo.spec
koji build --scratch rawhide foo.srpm
```

From the latest git commit:

```
koji build --scratch rawhide 'git url'
```

Warning: Scratch builds will *not* work correctly if your .spec file does something different depending on the value of %fedora, %fc9, and so on. Macro values like these are set by the *builder*, not by koji, so the value of %fedora will be for whatever created the source RPM, and *not* what it's being built on. Non-scratch builds get around this by first re-building the source RPM.

If you have committed the changes to git and you are in the current branch, you can do a scratch build with fedpkg tool which wraps the koji command line tool with the appropriate options:

```
fedpkg scratch-build
```

if you want to do a scratch build for a specific architecture, you can type:

```
   fedpkg scratch-build-<archs>

can be a comma separated list of several architectures.
```

finally is possible to combine the scratch-build command with a specific koji tag in the form:

```
fedpkg scratch-build --target TARGET
```

fedpkg scratch-build –help or koji build –help for more information.

Build Failures

If your package fails to build, you will see something like this:

```
420066 buildArch kernel-2.6.18-1.2739.10.9.el5.jjf.215394.2.src.rpm,
ia64): open (build-1.example.com) -> FAILED: BuildrootError:
error building package (arch ia64), mock exited with status 10
```

You can figure out why the build failed by looking at the log files. If there is a build.log, start there. Otherwise, look at init.log.

Logs can be found via the web interface in the Task pages for the failed task. Alternatively the koji client can be used to view the logs via the `watch-logs` command. See the help output for more details.

Advanced use of Koji

We've tried to make Koji self-documenting wherever possible. The command line tool will print a list of valid commands and each command supports –help. For example:

```
$ koji help

Koji commands are:
build                Build a package from source
cancel-task          Cancel a task
help                 List available commands
latest-build         Print the latest rpms for a tag
latest-pkg           Print the latest builds for a tag
[...]
```

```
$ koji build --help

usage: koji build [options]  tag URL
(Specify the --help global option for a list of other help options)

options:
-h, --help            show this help message and exit
--skip-tag            Do not attempt to tag package
--scratch             Perform a scratch build
--nowait              Don't wait on build
[...]
```

Using koji to generate a mock config to replicate a buildroot

koji can be used to replicate a build root for local debugging

```
koji mock-config --help
Usage: koji mock-config [options] name
(Specify the --help global option for a list of other help options)

Options:
  -h, --help            show this help message and exit
  --arch=ARCH           Specify the arch
  --tag=TAG             Create a mock config for a tag
  --task=TASK           Duplicate the mock config of a previous task
  --buildroot=BUILDROOT
                        Duplicate the mock config for the specified buildroot
                        id
  --mockdir=DIR         Specify mockdir
  --topdir=DIR          Specify topdir
  --topurl=URL          url under which Koji files are accessible
  --distribution=DISTRIBUTION
                        Change the distribution macro
  -o FILE               Output to a file
```

for example to get the latest buildroot for dist-f12-build run

```
koji mock-config --tag dist-f12-build --arch=x86_64 --topurl=https://kojipkgs.
↪fedoraproject.org/ dist-f12
```

you will need to pass in –topurl=https://kojipkgs.fedoraproject.org/ to any mock-config command to get a working mock-config from fedoras koji.

Tuning mock's behavior per tag

Few options for mock can be configured per-tag. These options are stored in tag info's *extra* field. Extra values can be checked via *koji taginfo* command. Example for forcing *dnf* usage in specific build environment follows:

```
koji edit-tag dnf-fedora-tag -x mock.package_manager=dnf
```

- *mock.package_manager* - If this is set, it will override mock's default package manager. Typically used with *yum* or *dnf* values.
- *mock.new_chroot* - 0/1 value. If it is set, *–new-chroot* or *–old-chroot* option is appended to any mock call. If it is not set, mock's default behavior is used.

Using Koji to control tasks

List tasks:

```
koji list-tasks
```

List only tasks requested by you:

```
koji list-tasks --mine
```

requeue an already-processed task: general syntax is: koji resubmit [options] taskID

```
koji resubmit 3
```

Building a Package with the command-line tool

Instead of using the fedpkg target, you can also directly use the command_line tool, koji.

To build a package, the syntax is:

```
$ koji build <build target> <git URL>
```

For example:

```
$ koji build dist-f14 'git url'
```

The koji build command creates a build task in Koji. By default the tool will wait and print status updates until the build completes. You can override this with the –nowait option.

NOTE: For fedora koji, the git url MUST be based on pkgs.fedoraproject.org. Other arbitrary git repos cannot be used for builds.

Koji tags and packages organization

Terminology

In Koji, it is sometimes necessary to distinguish between a package in general, a specific build of a package, and the various rpm files created by a build. When precision is needed, these terms should be interpreted as follows:

- Package: The name of a source rpm. This refers to the package in general and not any particular build or subpackage. For example: kernel, glibc, etc.

- Build: A particular build of a package. This refers to the entire build: all arches and subpackages. For example: kernel-2.6.9-34.EL, glibc-2.3.4-2.19.
- RPM: A particular rpm. A specific arch and subpackage of a build. For example: kernel-2.6.9-34.EL.x86_64, kernel-devel-2.6.9-34.EL.s390, glibc-2.3.4-2.19.i686, glibc-common-2.3.4-2.19.ia64

Tags and targets

Koji organizes packages using tags. In Koji a tag is roughly a collection of packages:

- Tags support inheritance
- Each tag has its own list of valid packages (inheritable)
- Package ownership can be set per-tag (inheritable)
- When you build you specify a target rather than a tag

A build target specifies where a package should be built and how it should be tagged afterwards. This allows target names to remain fixed as tags change through releases.

Koji commands for tags

Targets

You can get a full list of build targets with the following command:

```
$ koji list-targets
```

You can see just a single target with the –name option:

```
$ koji list-targets --name dist-f14

Name                              Buildroot                        Destination
-------------------------------------------------------------------------------
↪-------
dist-f14                       dist-f14-build                  dist-f14
```

This tells you a build for target dist-f14 will use a buildroot with packages from the tag dist-f14-build and tag the resulting packages as dist-f14.

Watch out: You probably don't want to build against dist-rawhide. If Fedora N is the latest one out, to build to the next one, choose dist-f{N+1}.

Tags

You can get a list of tags with the following command:

```
$ koji list-tags
```

Packages

As mentioned above, each tag has its own list of packages that may be placed in the tag. To see that list for a tag, use the list-pkgs command:

```
$ koji list-pkgs --tag dist-f14
```

The first column is the name of the package, the second tells you which tag the package entry has been inherited from, and the third tells you the owner of the package.

Latest Builds

To see the latest builds for a tag, use the latest-pkg command:

```
$ koji latest-pkg --all dist-f14
```

The output gives you not only the latest builds, but which tag they have been inherited from and who built them.

Category:Package Maintainers

2.12.2 Koji XMLRPC API

All features supported by command-line client are also accessible by XMLRPC API. You can get listing of all available calls, arguments and basic help via calling *koji list-api* command. This call will also provide you API extensions provided by plugins in that particular koji instance.

Because of the data Koji routinely deals with, we use the following extensions to the xmlrpc standard:

- We use the `nil` extension to represent null values (e.g. None in Python). Koji's library handles this automatically. If you are using a different library, you may need to explicitly enable this (e.g. enabling allow_none in Python's own xmlrpc library).
- We represent large integers with the `i8` tag. This standard is borrowed from Apache's *ws-xmlrpc <https://ws.apache.org/xmlrpc/types.html>* implementation. Python's own xmlrpc library understands this tag, even thought it will not emit it.

2.13 Koji Profiles

This document describes how to work with koji profiles.

2.13.1 Command Line Interface

Koji client allows connecting to multiple koji instances from CLI by using profiles. The default profile is given by executable file name, which is 'koji'.

To change koji profile, you can:

- run koji with –profile=$profile_name argument
- change executable file name by symlinking $profile_name -> koji

2.13.2 Configuration Files

Configuration files are located in following locations:

- /etc/koji.conf
- /etc/koji.conf.d/*.conf

- ~/.koji/config.d/*.conf
- user-specified config

Koji reads them, looking for [$profile_name] sections.

2.13.3 Using Koji Profiles in Python

Instead of using koji module directly, get profile specific module by calling:

```
>>> mod = koji.get_profile_module($profile_name)
```

This module is clone of koji module with additional profile specific tweaks.

Profile configuration is available via:

```
>>> mod.config
```

Example

Print configuration:

```
import koji

fedora_koji = koji.get_profile_module("koji")
ppc_koji = koji.get_profile_module("ppc-koji")

for i in (fedora_koji, ppc_koji):
    print("PROFILE: %s" % i.config.profile)
    for key, value in sorted(i.config.__dict__.items()):
        print("    %s = %s" % (key, value))
    print("")
```

Use ClientSession:

```
import koji

koji_module = koji.get_profile_module("koji")
client = koji_module.ClientSession(koji_module.config.server)
print(client.listTags())
```

2.13.4 TODO

- consider using pyxdg for user config locations

2.14 Plugins

Following plugins are available in default koji installation.

2.14.1 Runroot

Plugin for running any command in buildroot.

2.14.2 Save Failed Tree Plugin

In some cases developers want to investigate exact environment in which their build failed. Reconstructing this environment via mock needn't end with exactly same structure (due to builder settings, etc.). In such case this plugin can be used to retrieve tarball with complete mock tree.

Additional feature is that some paths from buildroot can be left out from tarball. Feature can be configured via */etc/kojid/plugins/save_failed_tree.conf* file. Currently only field filters.paths is used and it consists of globs (standard python's fnmatch is used) separated by whitespaces.

```
[filters]
paths = /etc/*.keytab /tmp/secret_data
```

Warning: For security reasons, currently all `/tmp/krb5cc*` and `/etc/*.keytab` files are removed from tarball. If we found some other dangerous pieces, they can be added to this blacklist.

Special task method is created for achieving this which is called `SaveFailedTree`. This task can be created via CLI: `koji save-failed-tree <taskID>`. Additional options are:

`--full`
directs koji to create tarball with complete tree.

`--nowait`
exit immediately after creating task

`--quiet`
don't print any information to output

After task finishes, one can find the tarball on relevant task web page (URL will be printed to stdout until `--quiet` is used.

Plugin allow to save trees only for tasks defined in config `/etc/koji-hub/plugins/save_failed_tree.conf`. Option `allowed_methods` contains list of comma-delimited names of tasks. Default configuration contains line: `allowed_methods = buildArch`. Anybody is allowed to create this type of task (and download tarball).

Warning: Don't forget that this type of task can generate huge amount of data, so use it wisely.

TODO

- Separate volume/directory on hub
- garbage collector + policy for retaining generated tarballs

2.15 Storage Volumes

2.15.1 Introduction

Since version 1.7.0, Koji has had the ability to store builds on multiple volumes.

Each build in Koji has a volume field that indicates which volume it is stored on. The default volume is named `DEFAULT` and corresponds to the original paths under `/mnt/koji` that predate this feature.

Additional volumes correspond to paths under `/mnt/koji/vol/NAME` (where NAME is the name of the volume). All builds associated with such a volume will be stored under this directory. The expectation is that this directory will map to a different file system.

Hint:

If `koji-1.13.0-1` is on the DEFAULT volume, its path will be:
/mnt/koji/packages/koji/1.13.0/1

If `koji-1.13.0-1` is on the volume named "test", its path will be:
/mnt/koji/vol/test/packages/koji/1.13.0/1

2.15.2 Constructing the path

If you are using the Koji python api, things should **just work**.

```
>>> import koji
>>> mykoji = koji.get_profile_module('mykoji')
>>> opts = mykoji.grab_session_options(mykoji.config)
>>> session = mykoji.ClientSession(mykoji.config.server, opts)
>>> binfo = session.getBuild('test-1-1')
>>> binfo['volume_name']
'DEFAULT'
>>> mykoji.pathinfo.build(binfo)
'/mnt/koji/packages/test/1/1'
>>> binfo = session.getBuild('fake-1.0-21')
>>> binfo['volume_name']
'vol3'
>>> mykoji.pathinfo.build(binfo)
'/mnt/koji/vol/vol3/packages/fake/1.0/21'
```

If you are constructing these paths yourself, then you may need to do a little work.

- Look for volume information in build data. Hub calls that return build data include `volume_name` and `volume_id` fields in their return.
- If the volume is `DEFAULT`, then the path is the "normal" path.
- Otherwise, you need to insert `/vol/<volume_name>` after the top directory (normally /mnt/koji).

2.15.3 Symlinks on default volume

For backwards compatibility, Koji maintains symlinks on the default volume for builds on other volumes.

```
$ file /mnt/koji/packages/fake/1.0/21
/mnt/koji/packages/fake/1.0/21: symbolic link to ../../../vol/vol3/packages/fake/1.0/
↪21
```

2.15.4 Adding a new volume

The new volume directory should initially contain a packages/ subdirectory, and the permissions should be the same as the default packages directory.

Assuming you do use a mount for a vol/NAME directory, you will want to ensure that the same mounts are created on all systems that interface with `/mnt/koji`, such as builders that run createrepo tasks, hosts running kojira or similar maintenance, and any hosts that rely on the topdir option rather than the topurl option.

Once you have the directory set up, you can tell Koji about it by running `koji add-volume NAME`. This call will fail if the hub can't find the directory.

2.15.5 Moving builds onto other volumes

By default, all builds live on the DEFAULT volume.

An admin can move a build to a different volume by using the `koji set-build-volume` command, or by using the underlying `changeBuildVolume` api call.

Moving a build across volumes will cause kojira to trigger repo regenerations, if appropriate. When the new volume is not DEFAULT, Koji will create a relative symlink to the new build directory on the default volume. Moving builds across volumes may immediately break repos (until the regen occurs), so use caution.

Consider the following example:

```
# mypkg-1.1-20 initially on default volume
$ file /mnt/koji/packages/mypkg/1.1/20
/mnt/koji/packages/mypkg/1.1/20: directory

# move it to test volume
$ koji set-build-volume test mypkg-1.1-20
$ file /mnt/koji/vol/test/packages/mypkg/1.1/20
/mnt/koji/vol/test/packages/mypkg/1.1/20: directory

# original location is now a symlink
$ file /mnt/koji/packages/mypkg/1.1/20
/mnt/koji/packages/mypkg/1.1/20: symbolic link to ../../../vol/test/packages/mypkg/1.
↪1/20
```

2.15.6 Using the volume in policy checks

Policies involving builds (e.g. gc policy, tag policy), can test a build's volume with the `volume` test. This is a pattern match test against the volume name.

2.15.7 Setting a volume policy

The Koji 1.14.0 release adds the ability to set a volume policy on the hub. This policy is used at import time to determine which volume the build should be assigned to. This provides a systematic way to distribute builds across multiple volumes without manual intervention.

There is relatively limited data available to the volume policy. Tests that are expected to work include:

- user based tests (the user performing the build or running the import)
- package based tests (e.g. `is_new_package` or `package`)
- cg match tests

- the buildtag test

The action value for the volume policy should be simply the name of the volume to use.

The default volume policy is `all :: DEFAULT`.

If the volume policy contains errors, or does not return a result, then the DEFAULT volume is used.

For more information about Koji policies see: *Defining hub policies*

2.15.8 CLI commands

add-volume adds a new volume (directory must already be set up)

list-volumes prints a list of known volumes

set-build-volume moves a build to different volume

2.15.9 API calls

addVolume(name, strict=True) Add a new storage volume in the database

applyVolumePolicy(build, strict=False) Apply the volume policy to a given build

changeBuildVolume(build, volume, strict=True) Move a build to a different storage volume

getVolume(volume, strict=False) Lookup the given volume

listVolumes() List storage volumes

removeVolume(volume) Remove unused storage volume from the database

2.16 How to write Koji plugins

2.16.1 Writing Koji plugins

Depending on what you are trying to do, there are different ways to write a Koji plugin.

Each is described in this file, by use case.

Adding new task types

Koji can do several things, for example build RPMs, or live CDs. Those are types of tasks which Koji knows about.

If you need to do something which Koji does not know yet how to do, you could create a Koji Builder plugin.

Such a plugin would minimally look like this:

```
from koji.tasks import BaseTaskHandler

class MyTask(BaseTaskHandler):
    Methods = ['mytask']
    _taskWeight = 2.0

    def handler(self, arg1, arg2, kwarg1=None):
        self.logger.debug("Running my task...")
```

(continues on next page)

(continued from previous page)

```
        # Here is where you actually do something
```

A few explanations on what goes on here:

- Your task needs to inherit from `koji.tasks.BaseTaskHandler`
- Your task must have a `Methods` attribute, which is a list of the method names your task can handle.
- You can specify the weight of your task with the `_taskWeight` attribute. The more intensive (CPU, IO, ...) your task is, the higher this number should be.
- The task object has a `logger` attribute, which is a Python logger with the usual `debug`, `info`, `warning` and `error` methods. The messages you send with it will end up in the Koji Builder logs (`kojid.log`)
- Your task must have a `handler()` method. That is the method Koji will call to run your task. It is the method that should actually do what you need. It can have as many positional and named arguments as you want.

Save your plugin as e.g `mytask.py`, then install it in the Koji Builder plugins folder: `/usr/lib/koji-builder-plugins/`

Finally, edit the Koji Builder config file, `/etc/kojid/kojid.conf`:

```
# A space-separated list of plugins to enable
plugins = mytask
```

Restart the Koji Builder service, and your plugin will be enabled.

You can try running a task from your new task type with the command-line:

```
$ koji make-task mytask arg1 arg2 kwarg1
```

Exporting new API methods over XMLRPC

Koji clients talk to the Koji Hub via an XMLRPC API.

It is sometimes desirable to add to that API, so that clients can request things Koji does not expose right now.

Such a plugin would minimally look like this:

```
def mymethod(arg1, arg2, kwarg1=None):
    context.session.assertPerm('admin')
    # Here is where you actually do something

mymethod.exported = True
```

A few explanations on what goes on here:

- Your plugin is just a method, with whatever positional and/or named arguments you need.
- You must export your method by setting its `exported` attribute to `True`
- The `context.session.assertPerm('admin')` is how you ensure that only the user with administrator privileges can use this call. Read-only methods can be (in most cases) public, so such line is not needed.

Save your plugin as e.g `mymethod.py`, then install it in the Koji Hub plugins folder: `/usr/lib/koji-hub-plugins/`

Finally, edit the Koji Hub config file, `/etc/koji-hub/hub.conf`:

```
# A space-separated list of plugins to enable
Plugins = mymethod
```

Restart the Koji Hub service, and your plugin will be enabled.

You can try calling the new XMLRPC API with the Python client library:

```
>>> import koji
>>> session = koji.ClientSession("http://koji/example.org/kojihub")
>>> session.mymethod(arg1, arg2, kwarg1='some value')
```

Ensuring the user has the required permissions

If you want your new XMLRPC API to require specific permissions from the user, all you need to do is add the following to your method:

```
from koji.context import context

def mymethod(arg1, arg2, kwarg1=None):
    context.session.assertPerm("admin")

    # Here is where you actually do something

mymethod.exported = True
```

In the example above, Koji will ensure that the user is an administrator. You could of course create your own permission, and check for that.

Running code automatically triggered on events

You might want to run something automatically when something else happens in Koji.

A typical example is to automatically sign a package right after a build finished. Another would be to send a notification to a message bus after any kind of event.

This can be achieved with a plugin, which would look minimally as follows:

```
from koji.plugin import callback

@callback('preTag', 'postTag')
def mycallback(cbtype, tag, build, user, force=False):
    # Here is where you actually do something
```

A few explanations on what goes on here:

- The `@callback` decorator allows you to declare which events should trigger your function. You can pass as many as you want. For a list of supported events, see `koji/plugins.py`.
- The arguments of the function depend on the event you subscribed to. As a result, you need to know how it will be called by Koji. You probably should use `*kwargs` to be safe. You can see how callbacks are called in the `hub/kojihub.py` file, search for calls of the `run_callbacks` function.

Save your plugin as e.g `mycallback.py`, then install it in the Koji Hub plugins folder: `/usr/lib/koji-hub-plugins`

Finally, edit the Koji Hub config file, `/etc/koji-hub/hub.conf`:

```
# A space-separated list of plugins to enable
Plugins = mycallback
```

Restart the Koji Hub service, and your plugin will be enabled.

You can try triggering your callback plugin with the command-line. For example, if you registered a callback for the `postTag` event, try tagging a build:

```
$ koji tag-build mytag mypkg-1.0-1
```

New command for CLI

When you add new XMLRPC call or just wanted to do some more complicated things with API, you can benefit from writing a new command for CLI.

Most simple command would look like this:

```
from koji.plugin import export_cli

@export_cli
def anon_handle_echo(options, session, args):
    "[info] Print arguments"
    usage = _("usage: %prog echo <message>")
    parser = OptionParser(usage=usage)
    (opts, args) = parser.parse_args(args)
    print(args[0])
```

@export_cli is a decorator which registers a new command. The command name is derived from name of the function. The function name must start with either *anon_handle_* or *handle_*. The rest of the name becomes the name of the command.

In the first case, the command will not automatically authenticate with the hub (though the user can still override this behavior with *–force-auth* option). In the second case, the command will perform authentication by default (this too can be overridden by the user with the *–noauth* option).

The example above is very simplistic. We recommend that developers also examine the actual calls included in Koji. The built in commands live in *koji_cli.commands* and our standard cli plugins live in *plugins/cli*.

Koji provides some important functions via in the client cli library (*koji_cli.lib*) for use by cli commands. Some notable examples are:

- *activate_session(session, options)* - It is needed to authenticate against hub. Both parameters are same as those passed to handler.
- *watch_tasks(session, tasklist, quiet=False, poll_interval=60)* - It is the same function used e.g. in *build* command for waiting for spawned tasks.
- *list_task_output_all_volumes(session, task_id)* - wrapper function for *listTaskOutput* with different versions of hub.

Final command has to be saved in python system-wide library path - e.g. in */usr/lib/python3.4/site-packages/koji_cli_plugins*. Filename doesn't matter as all files in this directory are searched for *@export_cli* macros. Note, that python 3 variant of CLI is looking to different directory than python 2 one.

CLI plugins structure will be extended (made configurable and allowing more than just adding commands - e.g. own authentication methods, etc.) in future.

2.16.2 Pull requests

These plugins have to be written in python 2.6+/3.x compatible way. We are using *six* library to support this, so we will also prefer pull requests written this way. CLI (and client library) is meant to be fully compatible with python 3 from koji 1.13.

Tests are also recommended for PR. For example one see *tests/test_plugins/test_runroot_cli.py*.

2.17 Writing Koji Code

2.17.1 Getting Started Hacking on Koji

This page gives an overview of the Koji code and then describes what needs to change if you want to add a new type of task. A new task could be for a new content type, or assembling the results of multiple builds together, or something else that helps your workflow. New contributors to Koji should leave this page knowing where to begin and have enough understanding of Koji's architecture to be able to estimate how much work is still ahead of them.

2.17.2 Task Flow

A task starts with a user submitting it with the Koji client, which is a command line interface. This contacts the hub, an apache-based server application. It leaves a row in the database that represents a "free" task, one that has not been assigned to a builder. Periodically, the builders asynchronously ping the hub asking if there are any tasks available, and at some point one will be given the new task. The hub marks this in the database, and the builder begins executing the task (a build).

Upon completion, the builder uploads the results to the hub, including logs, binaries, environment information, and whatever else the task handler for the build dictated. The hub moves the results to a permanent shared storage solution, and marks the task as completed (or failed). During this whole time, the webUI can be used to check up on progress. So the flow of work is:

```
Client -> Hub -> Builder -> Hub
```

If you wanted to add a new build type or task that was tightly integrated in Koji's data model, you would need to modify the CLI, Hub, Builder, and WebUI at a minimum. Alternatively, you could do this with a plugin, which is far simpler but less flexible.

2.17.3 Component Overview

Koji is comprised of several components, this section goes into details for each one, and what you potentially may need to change. Every component is written in Python, so you will need to know that language beyond a beginner level.

Koji-client

koji-client is a command line interface that provides many hooks into Koji. It allows the user to query much of the data as well as perform actions such as adding users and initiating build requests.

Option Handling

The code is in `cli/koji`. It uses `OptionParsers` extensively with interspersed arguments disabled. That means these two commands are not interpreted the same:

```
$ koji -u admin -p password tag-build some-tag --force some-build
$ koji tag-build -u admin -p password some-tag --force some-build
```

The second one will generate an error, because -u and -p are not options for tag-build, they must show up before that because they are global options that can be used with any subcommand. There will be two `OptionParsers` used with each command. The first is used to pick up arguments to `koji` itself, and the second for the subcommand specified. When the first one executes (see `get_options()`) it will figure out the subcommand and come up with a function name based on it.

The convention is to prepend the word `handle_` before it, and change all hyphens to underscores. If a command does not require an account with Koji, the function handle will prepended with `anon_handle_` instead. The code will dynamically call the derived function handle which is where the second `OptionParser` is used to parse the remaining options. To have your code log into Koji (you're writing a handle_ function), use the `activate_session` function. All function signatures in the client code will get a session object, which is your interface to the hub.

Profiles

It is possible to run the Koji client with different configuration profiles so that you can interact with multiple Koji instances easily. The `--profile` option to the Koji command itself enables this. You should have a `~/.koji/config` already, if not just copy from `/etc/koji.conf` to get a start. The profile command accepts an argument that matches a section in that config file. So if your config file had this:

```
[Fedora]
authtype = ssl
server = https://koji.fedoraproject.org/kojihub
topdir = /mnt/koji
weburl = https://koji.fedoraproject.org/koji
#pkgurl = https://koji.fedoraproject.org/packages
cert = ~/.fedora.cert
ca = ~/.fedora-upload-ca.cert
serverca = ~/.fedora-server-ca.cert

[MyKoji]
server = https://koji.mydomain.com/kojihub
authtype = kerberos
topdir = /mnt/koji
weburl = https://koji.mydomain.com/koji
topurl = https://download.mydomain.com/kojifiles
```

you could pass Fedora or MyKoji to –profile.

Creating Tasks

Once options are processed and understood, a task needs to be created on the hub so that a builder can come along and take it. This is accomplished with the `makeTask` method (defined on the Hub, so call it on the `session` object). The name of the task should match the name given to the task handler in the builder, which is explained later on.

Be sure to process the channel, priority, background, and watch/nowatch parameters too, which should be available to most new tasks. They'll be buried in the first argument to your handler function, which captures the options passed to the base Koji command.

If the client needs to make locally-available artifacts (config files, sources, kickstarts) accessible to the builder, it must be uploaded to the hub. This is the case with uploading SRPMs or kickstarts. You can easily upload this content with the `session.uploadWrapper` method. You can create progress bars as necessary with this snippet:

```
if _running_in_bg() or task_opts.noprogress:
  callback = None
else:
  callback = _progress_callback
serverdir = unique_path('cli-image')   # create a unique path on the hub
session.uploadWrapper(somefile, serverdir, callback=callback)
```

Task Arguments

If you define a new task for Koji, you'll want the task submission output to have the options ordered usefully. This output is automatically generated, but sometimes it does not capture the more important arguments you want displayed.

```
Created task 10001810
Watching tasks (this may be safely interrupted)...
10001810 thing (noarch): free
10001810 thing (noarch): free -> closed
  0 free  0 open  1 done  0 failed

10001810 thing (noarch) completed successfully
```

In this (fake) example, you can see that "noarch" is the only option being displayed, but maybe you want something more than just the task architecture displayed, like some other options that were passed in. You can fix this behavior in `koji/__init__.py` in the _taskLabel function. Here you can define the string(s) to display when Koji receives status on a task. That is the return value.

Koji-Hub

koji-hub is the center of all Koji operations. It is an XML-RPC server running under mod_wsgi in Apache. koji-hub is passive in that it only receives XML-RPC calls and relies upon the build daemons and other components to initiate communication. koji-hub is the only component that has direct access to the database and is one of the two components that have write access to the file system. If you want to make changes to the webUI (new pages or themes), you are looking in the wrong section, there is a separate component for that.

Implementation Details

The **hub/kojihub.py** file is where the server-side code lives. If you need to fix any server problems or want to add any new tasks, you will need to modify this file. Changes to the database schema will almost certainly require code changes too. This file gets deployed to **/usr/share/koji-hub/kojihub.py**, whenever you make changes to that remember to restart **httpd**. Also there are cases where httpd looks for an existing .pyc file and takes it as-is, instead of re-compiling it when the code is changed.

In the code there are two large classes: **RootExports** and **HostExports**. RootExports exposes methods using XML-RPC for any client that connects to the server. The Koji CLI makes use of this quite a bit. If you want to expose a new API to any remote system, add your code here. The HostExports class does the same thing except it will ensure the requests are only coming from builders. Attempting to use an API exposed here with the CLI will fail. If your work requires the builders to call a new API, you should implement it here. Any other function defined in this file is inaccessible by remote hosts. It is generally a good practice to have the exposed APIs do very little work, and pass off control to internal functions to do the heavy lifting.

Database Interactions

Database interactions are done with raw query strings, not with any kind of modern ORM. Consider using context objects from the Koji contexts library for thread-safe interactions. The database schema is captured in the **docs** directory in the root of a git clone. A visualization of the schema is not available at the time of this writing.

If you plan to introduce schema changes, please update both `schema.sql` and provide a migration script if necessary.

Troubleshooting

The hub runs in an Apache service, so you will need to look in Apache logs for error messages if you are encountering 500 errors or the service is failing to start. Specifically you want to check in:

- /var/log/httpd/error_log
- /var/log/httpd/ssl_error_log

If you need more specific tracebacks and debugging data, consider changing the debugging setting in **/etc/koji-hub/hub.conf**. Be advised the hub is very verbose with this setting on, your logs will take up gigabytes of space within several days.

Kojid

kojid is the build daemon that runs on each of the build machines. Its primary responsibility is polling for incoming build requests and handling them accordingly. Essentially kojid asks koji-hub for work. Koji also has support for tasks other than building. Creating install images is one example. kojid is responsible for handling these tasks as well. kojid uses mock for building. It also creates a fresh buildroot for every build. kojid is written in Python and communicates with koji-hub via XML-RPC.

Implementation Details

The daemon runs as a service on a host that is traditionally not the same as the hub or webUI. This is a good security practice because the service runs as root, and executes untrusted code to produce builds on a regular basis. Keeping the Hub separate limits the damage a malicious package can do to the build system as a whole. For the same reason, the filesystem that the hub keeps built software on should be mounted Read-Only on the build host. It should call APIs on the hub that are exposed through the `HostExports` class in the hub code. Whenever the builder accepts a task, it forks a process to carry out the build.

An initscript/unit-file is available for kojid, so it can be stopped and started like a normal service. Remember to do this when you deploy changes!

TaskHandlers

All tasks in kojid have a `TaskHandler` class that defines what to do when the task is picked up from the hub. The base class is defined in `koji/tasks.py` where a lot of useful utility methods are available. An example is `uploadFile`, which is used to upload logs and built binaries from a completed build to the hub since the shared filesystem is read only.

The daemon code lives in `builder/kojid`, which is deployed to /usr/sbin/kojid. In there you'll notice that each task handler class has a `Methods` member and `_taskWeight` member. These must be defined, and the former is used to match the name of a waiting task (on the hub) with the task handler code to execute. Each task handler object must have a `handler` method defined, which is the entry point for the forked process when a builder accepts a task.

Tasks can have subtasks, which is a typical model when a build can be run on multiple architectures. In this case, developers should write 2 task handlers: one handles the build for exact one architecture, and one that assembles the results of those tasks into a single build, and sends status information to the hub. You can think of the latter handler as the parent task.

All task handler objects have a `session` object defined, which is the interface to use for communications with the hub. So, parent tasks should kick off child tasks using the session object's subtask method (which is part of HostExports). It should then call `self.wait` with `all=True` to wait for the results of the child tasks.

Here's a stub of what a new build task might look like:

```
class BuildThingTask(BaseTaskHandler):
  Methods = ['thing']
  _taskWeight = 0.5

  def handler(self, a, b, arches, options):
    subtasks = {}
    for arch in arches:
      subtasks[arch] = session.host.subtask(method='thingArch', a, b, arch)
    results = self.wait(subtasks.values(), all=True)
    # parse results and put rows in database
    # put files in their final resting place
    return 'Build successful'

class BuildThingArchTask(BaseTaskHandler):
  Methods = ['thingArch']
  _taskWeight = 2.0

  def handler(self, a, b, arch):
    # do the build, capture results in a variable
    self.uploadFile('/path/to/some/log')
    self.uploadFile('/path/to/binary/file')
    return result
```

Source Control Managers

If you your build needs to check out code from a Source Control Manager (SCM) such as git or subversion, you can use SCM objects defined in `koji/daemon.py`. They take a specially formed URL as an argument to the constructor. Here's an example use. The second line is important, it makes sure the SCM is in the whitelist of SCMs allowed in `/etc/kojid/kojid.conf`.

```
scm = SCM(url)
scm.assert_allowed(self.options.allowed_scms)
directory = scm.checkout('/checkout/path', session, uploaddir, logfile)
```

Checking out takes 4 arguments: where to checkout, a session object (which is how authentication is handled), a directory to upload the log to, and a string representing the log file name. Using this method Koji will checkout (or clone) a remote repository and upload a log of the standard output to the task results.

Build Root Objects

It is encouraged to build software in mock chroots if appropriate. That way Koji can easily track precise details about the environment in which the build was executed. In `builder/kojid` a BuildRoot class is defined, which provides an interface to execute mock commands. Here's an example of their use:

```
broot = BuildRoot(self.session, self.options, build_tag, arch, self.id)
```

A session object, task options, and a build tag should be passed in as-is. You should also specify the architecture and the task ID. If you ever need to pass in specialized options to mock, look in the ImageTask.makeImgBuildRoot method to see how they are defined and passed in to the BuildRoot constructor.

Troubleshooting

The daemon writes a log file to `/var/log/kojid.log`. Debugging output can be turned on in `/etc/kojid/kojid.conf`.

Koji-Web

koji-web is a set of scripts that run in mod_wsgi and use the Cheetah templating engine to provide a web interface to Koji. It acts as a client to koji-hub providing a visual interface to perform a limited amount of administration. koji-web exposes a lot of information and also provides a means for certain operations, such as cancelling builds.

The web pages are derived from Cheetah templates, the syntax of which you can read up on here. These templates are the `chtml` files sitting in `www/kojiweb`. You'll notice quickly that these templates are referencing variables, but where do they come from?

The `www/kojiweb/index.py` file provides them. There are several functions named after the templates they support, and in each one a dictionary called `values` is populated. This is how data is gathered about the task, build, archive, or whatever the page is about. Take your time with `taskinfo.chtml` in particular, as the conditionals there have gotten quite long. If you are adding a new task to Koji, you will need to extend this at a minimum. A new type of build task would require this, and possibly another that is specific to viewing the archived information about the build. (taskinfo vs. buildinfo)

If your web page needs to display the contents of a list or dictionary, use the `$printMap` function to help with that. It is often sensible to define a function that easily prints options and values in a dictionary. An example of this is in taskinfo.chtml.

```
#def printOpts($opts)
  #if $opts
  <strong>Options:</strong><br/>
  $printMap($opts, '  ')
  #end if
#end def
```

Finally, if you need to expand the drop-down menus of "method" types when searching for tasks in the WebUI, you will need to add them to the `_TASKS` list in `www/kojiweb/index.py`. Add values where appropriate to `_TOPLEVEL_TASKS` and `_PARENT_TASKS` as well so that parent-child relationships show up correctly too.

Remember whenever you update a template or index.py, you will need to deploy and restart apache/httpd!

Troubleshooting

Like the hub, this component is backed by apache, so you should follow the same techniques for debugging Koji-Web as *Koji-Hub*.

Kojira

kojira is a daemon that keeps the build root repodata updated. It is responsible for removing redundant build roots and cleaning up after a build request is completed.

2.17.4 Building and Deploying Changes

The root of the git clone for Koji code contains a `Makefile` that has a few targets to make building and deployment a little easier. Among them are:

- tarball: create a bz2 tarball that could be consumed in an rpm build
- rpm: create Koji rpms. The NVRs will be defined by the spec file, which is also in the same directory. The results will appear in a `noarch` directory.
- test-rpm: like rpm, but append the Release field with a date and time stamp for easy upgrade-deployment

2.17.5 Plugins

This section is copied from the `docs/Writing_a_plugin.md` file.

Koji supports different types of plugins, three of which are captured here. Depending on what you are trying to do, there are different ways to write a Koji plugin.

Koji Builder Plugins

Koji can do several things, for example build RPMs, or live CDs. Those are types of tasks which Koji knows about. If you need to do something which Koji does not know yet how to do, you could create a Koji Builder plugin. Such a plugin would minimally look like this:

```
from koji.tasks import BaseTaskHandler

class MyTask(BaseTaskHandler):
  Methods = ['mytask']
  _taskWeight = 2.0

def handler(self, arg1, arg2, kwarg1=None):
  self.logger.debug("Running my task...")
  # Here is where you actually do something
```

A few explanations on what goes on here:

- Your task needs to inherit from ‘koji.tasks.BaseTaskHandler‘
- Your task must have a ‘Methods‘ attribute, which is a list of the method names your task can handle.
- You can specify the weight of your task with the ‘_taskWeight‘ attribute. The more intensive (CPU, IO, . . .) your task is, the higher this number should be.
- The task object has a `logger` attribute, which is a Python logger with the usual ‘debug‘, ‘info‘, ‘warning‘ and ‘error‘ methods. The messages you send with it will end up in the Koji Builder log.
- Your task must have a ‘handler()‘ method. That is the method Koji will call to run your task. It is the method that should actually do what you need. It can have as many positional and named arguments as you want.

Save your plugin as e.g `mytask.py`, then install it in the Koji Builder plugins folder: `/usr/lib/koji-builder-plugins/`. Finally, edit the Koji Builder config file, `/etc/kojid/kojid.conf`:

```
# A space-separated list of plugins to enable
plugins = mytask
```

Restart the Koji Builder service, and your plugin will be enabled. You can try running a task from your new task type with the command-line: `$ koji make-task mytask arg1 arg2 kwarg1`

Hub Plugins

Koji clients talk to the Koji Hub via an XMLRPC API. It is sometimes desirable to add to that API, so that clients can request things Koji does not expose right now. Such a plugin would minimally look like this:

```
def mymethod(arg1, arg2, kwarg1=None):
  # Here is where you actually do something
  mymethod.exported = True
```

What's happening?

- Your plugin is just a method, with whatever positional and/or named arguments you need.
- You must export your method by setting its `exported` attribute to `True`
- The `context.session.assertPerm()` is how you ensure that the correct permissions are available.

Save your plugin as e.g 'mymethod.py', then install it in the Koji Hub plugins folder, which is `/usr/lib/koji-hub-plugins/`

Finally, edit the Koji Hub config file, `/etc/koji-hub/hub.conf`:

```
# A space-separated list of plugins to enable
Plugins = mymethod
```

Restart the Koji Hub service, and your plugin will be enabled. You can try calling the new XMLRPC API with the Python client library:

```
>>> import koji
>>> session = koji.ClientSession("http://koji/example.org/kojihub")
>>> session.mymethod(arg1, arg2, kwarg1='some value')
```

If you want your new XMLRPC API to require specific permissions from the user, all you need to do is add the following to your method:

```
from koji.context import context

def mymethod(arg1, arg2, kwarg1=None):
  context.session.assertPerm("admin")
  # Here is where you actually do something
  mymethod.exported = True
```

In the example above, Koji will ensure that the user is an administrator. You could of course create your own permission, and check for that.

Event Plugin

You might want to run something automatically when something else happens in Koji. A typical example is to automatically sign a package right after a build finished. Another would be to send a notification to a message bus after any kind of event.

This can be achieved with a plugin too, which would look minimally as follows:

```
from koji.plugin import callback

@callback('preTag', 'postTag')
def mycallback(cbtype, tag, build, user, force=False):
  # Here is where you actually do something
```

So what is this doing?

- The `@callback` decorator allows you to declare which events should trigger your function. You can pass as many as you want. For a list of supported events, see `koji/plugins.py`.
- The arguments of the function depend on the event you subscribed to. As a result, you need to know how it will be called by Koji. You probably should use `*kwargs` to be safe. You can see how callbacks are called in the `hub/kojihub.py` file, search for calls of the `run_callbacks` function.

Save your plugin as e.g `mycallback.py`, then install it in the Koji Hub plugins folder: `/usr/lib/koji-hub-plugins`

Finally, edit the Koji Hub config file, `/etc/koji-hub/hub.conf`:

```
# A space-separated list of plugins to enable
Plugins = mycallback
```

Restart the Koji Hub service, and your plugin will be enabled. You can try triggering your callback plugin with the command-line. For example, if you registered a callback for the `postTag` event, try tagging a build: `$ koji tag-build mytag mypkg-1.0-1`

2.17.6 Submitting Changes

To submit code changes for Koji, please file a pull request in Pagure.

https://pagure.io/koji/pull-requests

Here are some guidelines on producing preferable pull requests.

- Each request should be a coherent whole, e.g. a single feature or bug fix. Please do not bundle a series of unrelated changes into a single PR
- Pull requests in Pagure come from a branch in your personal fork of Koji (either in Pagure or a remote git repo). Please use an appropriately named branch for this. Do not use the master branch of your fork. Also, please be aware that Pagure will automatically update the pull request if you modify the source branch
- Your branch should be based against the current HEAD of the target branch
- Please adhere to PEP8. While much of the older code in Koji does not, we try to stick to it with new code
- Code which is imported into CLI or needed for stand-alone API calls must run in both 2.6+ and 3.x python versions. We use the python-six library for compatibility. The affected files are:
  - `cli/*`
  - `koji/__init__.py`
  - `koji/auth.py`
  - `koji/tasks.py`
  - `koji/util.py`
  - `tests/test_lib/*`

  - `tests/test_cli/*`

- Check, that unit tests are not broken. Simply run `make test` in main directory of your branch. For python3 compatible-code we have also `make test3` target.

Note that the core development team for Koji is small, so it may take a few days for someone to reply to your request.

Partial work

Pull requests are for changes that are complete and ready for inclusion, but sometimes you have partial work that you may want feedback on. Please don't submit a PR before your code is complete.

The preferred way to request early feedback is to push your changes to a your own koji fork and then send an email to koji-devel AT lists.fedorahosted.org requesting review. This approach is one step short of a PR, making it easy to upgrade to a PR once the changes are ready.

2.17.7 Unit Tests

Koji comes with a small test suite, that you should always run when making changes to the code. To do so, just run `make test` in your terminal.

You will need to install the following packages to actually run the tests.

- `findutils`
- `pyOpenSSL`
- `python-coverage`
- `python-krbV`
- `python-mock`
- `python-psycopg2`
- `python-requests`
- `python-qpid-proton`

Please note that it is currently not supported to use *virtualenv* when hacking on Koji.

Unit tests are run automatically for any commit in master branch. We use Fedora's jenkins instance for that. Details are given here: Unit tests in Fedora's Jenkins.

2.18 Koji Content Generators

A Koji Content Generator is an external service that generates content (jars, zips, tarballs, .npm, .wheel, .gem, etc) which is then passed to Koji for management and delivery to other processes in the release workflow. Content Generators can evolve independently of the Koji codebase, enabling the build process to be more agile and flexible to changing requirements and new technologies, while allowing Koji to provide stable APIs and interfaces to other processes.

Along with the content to be managed by Koji, a Content Generator will provide enough metadata to enable a reasonable level of auditing and reproducibility. The exact data provided and the format used is being discussed, but will include information like the upstream source URL, build tools used, build environment contents, and any container/virtualization technologies used.

The intention is that a team dedicated to managing a specific content type will design and maintain their own Content Generator, in coordination with the Koji developers. Once the Content Generator is ready for production use it will be

given permission to import content and metadata it produces into Koji. Policies on the Koji hub will validate imported content and metadata and ensure that it is complete and consistent.

2.18.1 Requirements for writing a Content Generator

From an implementation perspective, content generators have wide latitude in how they perform builds. To ensure sanity in the build process, we strongly recommend that administrators of Koji systems set policies about what content generators are allowed to do, and make sure that those policies are followed before the content generator is granted authorization in their Koji system.

Below are some examples of the sorts of policies that one might require. Content Generators should be designed and implemented with these requirements in mind. Please note that the list below is not complete.

Avoid Using the Host's Software

During the building process, the code should avoid using the host's installed software. The more reliance on installed software, the more risk in the future that changes (such as upgrading a builder) will break the build processes. Use mock chroots, VM guests, or containers wherever possible to insulate against changes. Isolating the build environment from the host environment makes reproducing work much easier and predictable.

Source of build environment content

The build environment must come from somewhere. In a standard Koji build, it comes from content already in Koji, or from configured external repositories.

CG authors will likely want to pull content from sources outside of Koji. Koji administrators should set a clear policy about which sources are acceptable. The use of arbitrary sources can make it difficult or impossible to reproduce build environments.

Binaries (or other compiled content) from Upstream May Not become included in output

If tools or other content downloaded from external sources are used in the build, they may not be included in CG build output, and may not be imported into Koji. In other words, output must be built from sources in the CG or Koji, not retrieved from the internet. Tools necessary to build product content can be downloaded and cached in the CG.

Log all Transformations of Content

When the content is building, as much should be logged as possible. In addition to compilation, if the content goes through other transformations, perhaps changing formats, that should be logged as well. There can be no black-box transformations of the output. Imagine having to figure out how a piece of content was built 5 years into the future to understand the motivation behind this requirement. Details of the build environment and tools used in the environment should be recorded too.

Preserve All Inputs

All inputs to a build task should be preserved either as logs, a database, or as output of the build itself.

Preserve All Outputs

Naturally the outputs of a build should be preserved too. Transient artifacts are not strictly required, but if they're not onerous to maintain, they should be included. It must not be necessary to further transform the content to make it usable.

Do Not Use Caching Mechanisms

Content Generators must build without caching mechanisms (in compilers or DNF|YUM) wherever possible. Caches make reproducing results in the future more difficult, and also introduce layers of indirection that can make debugging a build more difficult. Consider the risk of re-shipping a security flaw that is compiled in because an outdated library was cached in the Content Generator, this is why we have this requirement.

2.18.2 Metadata

Metadata will be provided by the Content Generator as a JSON file. There is a proposal of the *Content Generator Metadata* format available for review.

2.19 Koji Content Generator Metadata

This document describes the Koji Content Generator Metadata Format (version 0). This is the metadata that should be provided by a Content Generator in order for the content to be imported and managed by Koji. If you have further questions about *Content Generators*, please email koji-devel@lists.fedorahosted.org.

2.19.1 Format

Content Generator Metadata for a single build is provided as a JSON map. The map has four top-level entries:

- metadata_version: The version of the metadata format used. Currently must be 0.
- build: A map containing information about the build.
- buildroots: A list of maps, one for each environment in which build output was generated, containing information about that environment.
- output: A list of maps, one map for each file that will be imported and managed by Koji.

metadata_version

This is an integer which indicates the version of the metadata format contained in this file. It will start at 0 and be incremented as the metadata format evolves.

build

The build map contains the following entries:

- name: The name of the build.
- version: The version of the build.
- release: The release of the build.

- source: The SCM URL of the sources used in the build.
- start_time: The time the build started, in seconds since the epoch.
- end_time: The time the build was completed, in seconds since the epoch.
- owner: The owner of the build task in username format. This field is optional.
- extra: A map of extra metadata associated with the build, which must include one of:
  - typeinfo: A map whose single entry is the name of the content generator type, which is a map containing type-specific information for this build.
  - maven, win, or image: Legacy content generator type names which appear at this level instead of inside typeinfo.

buildroots

Each map in the buildroots list contains the following entries:

- id: An id for this buildroot entry. Only needs to be consistent within this file (it will be referenced by the output). Can be synthetic/generated/random.
- host: Map containing information about the host where the build was run.
  - os: The operating system that was running on the host.
  - arch: The processor architecture of the host.
- content_generator: Map containing information about the Content Generator which ran the build.
  - name: The short name of the Content Generator.
  - version: The version of the Content Generator.
- container: Map containing information about the container in which the build was run.
  - type: The type of container that was used, eg. none, directory, chroot, mock-chroot, kvm, docker
  - arch: The architecture of the container. May be different than the architecture of the host, eg. i686 container on x86_64 host.
- tools: List of maps containing information about the tools used to run the build. Each map contains:
  - name: Name of the tool used.
  - version: Version of the tool used.
- components: List of maps containing information about content installed in the build environment (if any). Each map is guaranteed to contain a **type** field, which determines what other fields are present in the map. For maps where **type = rpm**, the following fields will be present:
  - name: The rpm name.
  - version: The rpm version.
  - release: The rpm release.
  - epoch: The rpm epoch.
  - arch: The rpm arch.
  - sigmd5: The SIGMD5 tag from the rpm header.
  - signature: The signature used to sign the rpm (if any).
- For maps where **type = file**, the following fields will be present:

  - filename: The name of the file.
  - filesize: The size of the file.
  - checksum: The checksum of the file.
  - checksum_type: The checksum type used.
- For maps where **type = kojifile**, the following fields will be present:
  - filename: The name of the file.
  - filesize: The size of the file.
  - checksum: The checksum of the file.
  - checksum_type: The checksum type used.
  - nvr: Build nvr from which this file origins.
  - archive_id: ID of archive from specified build.
- The format may be extended with other types in the future.
- extra: A map containing information specific to the Content Generator that produced the files to import. For OSBS, the extra map should contain a osbs entry, which is a map with the following fields:
  - build_id: The ID of the build in OSBS.
  - builder_image_id: The ID of the image in OSBS that was used to run the build.

output

Each map in the output list contains the following entries:

- buildroot_id: The id of the buildroot used to create this file. Must match an entry in the buildroots list.
- filename: The name of the file.
- filesize: The size of the file.
- arch: The architecture of the file (if applicable).
- checksum: The checksum of the file.
- checksum_type: The checksum type used.
- type: The type of the file. Log files should use "log".
- components: If the output file is composed from other units, those should be listed here. The format is the same as the **components** field of a buildroot map.
- extra: Free-form, but should contain IDs that allow tracking the output back to the system in which it was generated (if that system retains a record of output). For docker, the extra map should contain a docker entry, which is a map with the following fields:
  - id: The ID of the docker image produced in the repo used by the build tool
  - parent_id: The parent ID of the docker image produced (if applicable).
  - repositories: A list of repository locations where the image is available.
  - digests: A map of media type (such as "application/vnd.docker.distribution.manifest.v2+json") to manifest digest (a string usually starting "sha256:"), for each available media type.

2.19.2 Example Metadata JSON

The below JSON is based loosely on the output of a docker image build.

```
{"metadata_version": 0,
 "build": {"name": "rhel-server-docker",
           "version": "7.1",
           "release": "4",
           "source": "git://git.engineering.redhat.com/users/vpavlin/tdl_templates.git
↪#a14f145244",
           "extra": {},
           "start_time": 1423148398,
           "end_time": 1423148828,
           "owner": "jdoe"},
 "buildroots": [{"id": 1,
                 "host": {"os": "rhel-7",
                          "arch": "x86_64"},
                 "content_generator": {"name": "osbs",
                                       "version": "0.2"},
                 "container": {"type": "docker",
                               "arch": "x86_64"},
                 "tools": [{"name": "docker",
                            "version": "1.5.0"}],
                 "components": [{"type": "rpm",
                                 "name": "glibc",
                                 "version": "2.17",
                                 "release": "75.el7",
                                 "epoch": null,
                                 "arch": "x86_64",
                                 "sigmd5": "a1b2c3...",
                                 "signature": "fd431d51"},
                                {"type": "rpm",
                                 "name": "openssl",
                                 "version": "1.0.1e",
                                 "release": "42.el7",
                                 "epoch": null,
                                 "arch": "x86_64",
                                 "sigmd5": "d4e5f6...",
                                 "signature": "fd431d51"},
                                {"type": "rpm",
                                 "name": "bind-libs",
                                 "version": "9.9.4",
                                 "release": "18.el7",
                                 "epoch": 32,
                                 "arch": "x86_64",
                                 "sigmd5": "987abc...",
                                 "signature": null},
                                {"type": "rpm",
                                 "name": "python-urllib3",
                                 "version": "1.5",
                                 "release": "8.el7",
                                 "epoch": null,
                                 "arch": "noarch",
                                 "sigmd5": "123hgf...",
                                 "signature": null},
                                {"type": "file",
                                 "filename": "jboss-eap-6.3.3-full-build.zip",
                                 "filesize": 12345678,
```

(continues on next page)

(continued from previous page)

```
                                "checksum": "5ec2f29c4e1c2e2aa6552836e236a158",
                                "checksum_type": "md5"}],
                "extra": {"osbs": {"build_id": 12345,
                                   "builder_image_id": 67890}}
                }],
 "output": [{"buildroot_id": 1,
             "filename": "rhel-server-docker-7.1-4.x86_64.tar.xz",
             "filesize": 34440656,
             "arch": "x86_64",
             "checksum_type": "md5",
             "checksum": "275ae42a45cfedbdb0c0a1acc0b55a1b",
             "type": "docker-image",
             "components": "",
             "extra": {"docker": {"id": "987654...",
                                  "parent_id": "a1b2c3...",
                                  "repositories": ["repository.example.com/username/
↪imagename:7.1-4",
                                                   "repository.example.com/username/
↪imagename@sha256:100000...",
                                                   "repository.example.com/username/
↪imagename@sha256:200000..."],
                                  "digests": {"application/vnd.docker.distribution.
↪manifest.v1+json": "sha256:100000...",
                                              "application/vnd.docker.distribution.
↪manifest.v2+json": "sha256:200000..."}
                                  }}},
            {"buildroot_id": 1,
             "filename": "checkout.log",
             "filesize": 85724,
             "arch": "noarch",
             "checksum_type": "md5",
             "checksum": "a1b2c3...",
             "type": "log"},
            {"buildroot_id": 1,
             "filename": "os-indirection.log",
             "filesize": 27189,
             "arch": "noarch",
             "checksum_type": "md5",
             "checksum": "d4f5g6...",
             "type": "log"}
            ]
}
```

CHAPTER

THREE

HOWTOS

Setting up and using Koji on Fedora:

- *Using the Koji build system*
- *Run Your Own Koji Build Server*
- *Building Images in Koji*
- *Defining hub policies*

CHAPTER

FOUR

KOJI ARCHITECTURE

4.1 Terminology

In Koji it is sometimes necessary to distinguish between a package in general, a specific build of a package, and the various rpm files created by a build. When precision is needed, these terms should be interpreted as follows:

Package The name of a source rpm. This refers to the package in general and not any particular build or subpackage. For example: kernel, glibc, etc.

Build A particular build of a package. This refers to the entire build: all arches and subpackages. For example: kernel-2.6.9-34.EL, glibc-2.3.4-2.19.

RPM A particular rpm. A specific arch and subpackage of a build. For example: kernel-2.6.9-34.EL.x86_64, kernel-devel-2.6.9-34.EL.s390, glibc-2.3.4-2.19.i686, glibc-common-2.3.4-2.19.ia64

CHAPTER

FIVE

KOJI COMPONENTS

Koji is comprised of several components:

5.1 Koji-Hub

koji-hub is the center of all Koji operations. It is an XML-RPC server running under mod_wsgi in Apache. koji-hub is passive in that it only receives XML-RPC calls and relies upon the build daemons and other components to initiate communication. koji-hub is the only component that has direct access to the database and is one of the two components that have write access to the file system.

5.2 Kojid

kojid is the build daemon that runs on each of the build machines. Its primary responsibility is polling for incoming build requests and handling them accordingly. Essentially kojid asks koji-hub for work. Koji also has support for tasks other than building. Creating install images is one example. kojid is responsible for handling these tasks as well. kojid uses mock for building. It also creates a fresh buildroot for every build. kojid is written in Python and communicates with koji-hub via XML-RPC.

5.3 Koji-Web

koji-web is a set of scripts that run in mod_wsgi and use the Cheetah templating engine to provide a web interface to Koji. It acts as a client to koji-hub providing a visual interface to perform a limited amount of administration. koji-web exposes a lot of information and also provides a means for certain operations, such as cancelling builds.

5.4 Koji-client

koji-client is a CLI written in Python that provides many hooks into Koji. It allows the user to query much of the data as well as perform actions such as adding users and initiating build requests.

5.5 Kojira

kojira is a daemon that keeps the build root repodata updated. It is responsible for removing redundant build roots and cleaning up after a build request is completed.

CHAPTER

SIX

PACKAGE ORGANIZATION

6.1 Tags and Targets

Koji organizes packages using tags:

- Tags are tracked in the database but not on disk
- Tags support multiple inheritance
- Each tag has its own list of valid packages (inheritable)
- Package ownership can be set per-tag (inheritable)
- Tag inheritance is more configurable
- When you build you specify a target rather than a tag

A build target specifies where a package should be built and how it should be tagged afterwards. This allows target names to remain fixed as tags change through releases. You can get a full list of build targets with the following command:

```
$ koji list-targets
```

You can see just a single target with the –name option:

```
$ koji list-targets --name dist-fc7

Name                            Buildroot                       Destination
-------------------------------------------------------------------------------
↪-------
dist-fc7                        dist-fc7-build                  dist-fc7
```

This tells you a build for target dist-fc7 will use a buildroot with packages from the tag dist-fc7-build and tag the resulting packages as dist-fc7.

You can get a list of tags with the following command:

```
$ koji list-tags
```

6.2 Package lists

As mentioned above, each tag has its own list of packages that may be placed in the tag. To see that list for a tag, use the list-pkgs command:

```
$ koji list-pkgs --tag dist-fc7

Package                 Tag                     Extra Arches     Owner
----------------------- ----------------------- ---------------- ----------------
ElectricFence           dist-fc6                                 pmachata
GConf2                  dist-fc6                                 rstrode
lucene                  dist-fc6                                 dbhole
lvm2                    dist-fc6                                 lvm-team
ImageMagick             dist-fc6                                 nmurray
m17n-db                 dist-fc6                                 majain
m17n-lib                dist-fc6                                 majain
MAKEDEV                 dist-fc6                                 clumens
[...]
```

The first column is the name of the package, the second tells you which tag the package entry has been inherited from, and the third tells you the owner of the package.

6.3 Latest Builds

To see the latest builds for a tag, use the latest-pkg command:

```
$ koji latest-pkg --all dist-fc7

Build                                   Tag                   Built by
--------------------------------------  --------------------  ----------------
ConsoleKit-0.1.0-5.fc7                  dist-fc7              davidz
ElectricFence-2.2.2-20.2.2              dist-fc6              jkeating
GConf2-2.16.0-6.fc7                     dist-fc7              mclasen
ImageMagick-6.2.8.0-3.fc6.1             dist-fc6-updates      nmurray
MAKEDEV-3.23-1.2                        dist-fc6              nalin
MySQL-python-1.2.1_p2-2                 dist-fc7              katzj
NetworkManager-0.6.5-0.3.cvs20061025.fc7  dist-fc7            caillon
ORBit2-2.14.6-1.fc7                     dist-fc7              mclasen
```

The output gives you not only the latest builds, but which tag they have been inherited from and who built them (note: for builds imported from beehive the "built by" field may be misleading).

6.4 Documentation

We've tried to make Koji self-documenting wherever possible. The command line tool will print a list of valid commands and each command supports –help. For example:

```
$ koji help

Koji commands are:
build                Build a package from source
cancel-task          Cancel a task
help                 List available commands
latest-build         Print the latest rpms for a tag
latest-pkg           Print the latest builds for a tag
[...]
```

```
$ koji build --help

usage: koji build [options]  tag URL
(Specify the --help global option for a list of other help options)

options:
-h, --help            show this help message and exit
--skip-tag            Do not attempt to tag package
--scratch             Perform a scratch build
--nowait              Don't wait on build
[...]
```

You can see administrator-only command help with –admin. Most users will never use these additional commands, but if you're setting up your own Koji system, you may find them very useful.

```
$ koji help --admin
Available commands:
        add-external-repo         Create an external repo and/or add one to a tag
        add-group                 Add a group to a tag
        add-group-pkg             Add a package to a group's package listing
[...]
```

CHAPTER

SEVEN

KOJI DEPLOYMENTS

Koji is also known to be used in many places, and we *track them on this page*. Feel free to add your entry. There is no additional obligation to you for doing so. :)

CHAPTER

EIGHT

KOJI CONTRIBUTOR GUIDES

If you're interested in submitting patches, writing documentation, or filing bugs this section is for you. In time this will be the best place to learn how to get involved.

- *Getting Started as a Developer*

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

C