A Project Report

On

Using Koji Build System to Atuomate building RPM Packages

BY

Tanay Sodha

2022B4A70720P

SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS OF

BITS F221: Practice School 1

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

PILANI CAMPUS

(July 2024)

SUBMITTED FOR

END SEMESTER EVALUATION

BASED ON PROJECT ACTIVITIES UNDERTAKEN AT

Centre for Development of Advanced Computing (CDAC), Pune

ABSTRACT

Fedora is a popular Linux Distribution that is often used for OS development. In the Mid Semester Report, we saw the making of a Fedora based OS from scratch for RISC-V architecture. For this minimal OS, we require only certain RPM packages in a local repository so that it is easy to install on an individual machine. In order to do so, we must rebuild Source RPM files.

Fedora uses the RPM package manager. RPM packages are basically programs or apps that can be run on the PC. When creating an Operating System, they are to be built from their source codes, which can be tedious when done manually. The Koji Build System provides an interface that helps automate the process of building Source RPM packages while taking care of several issues on the way. We try to use Koji and understand its functionality.

CONTENTS

1. Packages and Package Management System

In Linux, a package is a bundled collection of software, including applications, libraries, and system tools. Each package typically contains the compiled code, configuration files, dependencies, and metadata necessary for the software to function correctly. Packages simplify the distribution, installation, and updating of software by providing a standardized format.

A package management system is a set of tools and protocols used to automate the process of installing, upgrading, configuring, and removing software packages. These systems handle dependencies, ensuring that all required software components are present and compatible.

Popular package management systems include:

- APT (Advanced Package Tool): Used by Debian-based distributions like Ubuntu. It manages .deb packages.
- YUM (Yellowdog Updater, Modified) and DNF (Dandified YUM): Used by Red Hat-based distributions like Fedora and CentOS. They manage .rpm packages.
- Pacman: Used by Arch Linux, managing .pkg.tar.xz packages.

Importance

- **Simplifies Software Installation**: Users can easily install software with a single command.
- **Dependency Management**: Automatically resolves and installs dependencies required by software packages.
- **Security**: Ensures that software is obtained from trusted sources and can apply security patches automatically.
- **Consistency**: Maintains a consistent environment by managing software versions and configurations.
- **Updates**: Facilitates easy updates of software packages, ensuring systems have the latest features and security fixes.

Ways to Install Packages:

1. Use a Package Management System
2. Install a package from Source using .cofigure and make commands
3. Build a source rpm file into an rpm file using rpmbuild

2. Koji Build System

The Koji build system is an open-source software project used for building and managing software packages. It was initially developed for the Fedora Project, but it can be used for other distributions and projects as well. Here are the key features and components of the Koji build system:

Key Features:
1. Build Automation: Koji automates the process of building software packages from source code. It handles the compilation and packaging processes, ensuring consistent and repeatable builds.

2. Distributed Builds: Koji supports distributed builds across multiple build servers. This helps in scaling the build process and utilizing resources efficiently.

3. Package Management: It provides tools for managing software packages, including creating, tracking, and distributing them.

4. Version Control Integration: Koji integrates with version control systems (like Git) to fetch source code for builds. This ensures that the correct version of the source code is used for each build.

5. Security: Koji includes mechanisms for secure builds, such as running builds in isolated environments (e.g., chroots) and using digital signatures to verify the integrity of packages.

6. Web Interface: It offers a web-based interface for monitoring and managing builds. This interface provides access to build logs, package repositories, and build status.

7. CLI Tools: Koji also provides command-line tools for interacting with the build system, which is useful for scripting and automation.

Components:
1. Koji Hub: The central server that coordinates builds and maintains the database of build information. It handles requests from clients and build tasks from builders.

2.Koji Builders: These are the machines that perform the actual building of software packages. They run build tasks assigned by the Koji Hub.

3. Koji CLI and Web Interface: Tools for users to interact with the Koji system. The CLI (command-line interface) is used for scripting and automation, while the web interface provides a graphical view of the build system.

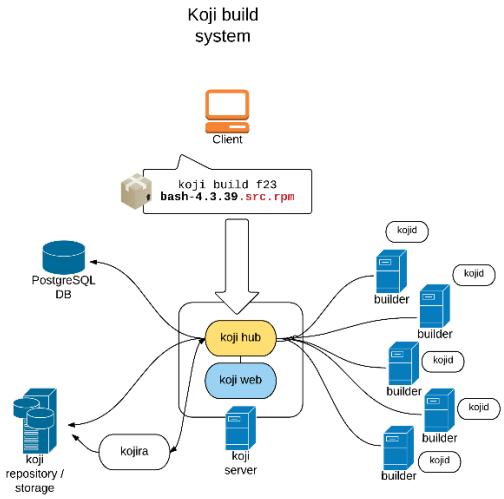
4. Database: Koji uses a database to store information about builds, packages, and other metadata. PostgreSQL is commonly used as the backend database.

5. Content Delivery Network (CDN): For distributing built packages to end users. This ensures that the packages are available globally with minimal latency.

Workflow:
1. Source Code Submission: Developers submit source code to the version control system.
2. Build Request: A build request is submitted to Koji, specifying the source code and build parameters.
3. Build Assignment: Koji Hub assigns the build task to an available builder.
4. Build Execution: The builder compiles the source code and creates the package.
5. Package Storage: The built package is stored in the Koji repository.
6. Package Distribution: The package is made available through the CDN for users to download.

Use Cases:
- Linux Distribution Builds: Used by projects like Fedora and CentOS for building and managing software packages.
- Continuous Integration/Continuous Deployment (CI/CD): Automating the build and release process for software projects.
- Package Management: Ensuring consistent and secure delivery of software packages to users.

Koji is a robust and scalable system for managing complex build processes, making it ideal for large-scale software projects and distributions.



3. The Code

The Koji Build System also provides a python API that helps us interact with several koji servers and perform operations. As a part of the project I wrote a code that does the following:

Logs in to the official fedora project server, gets information on the latest build of certain packages, and builds it on the local server (CDAC in this case)

The following sections describe the documentation of the code.

A flowchart of the workflow of the code is also attatched at the end.

1. Login & Profiles

```
import koji
import os
import subprocess

KOJIURL_fedora = "https://koji.fedoraproject.org/kojihub"
user = 'tanay'

fedora_session = koji.ClientSession(KOJIURL_fedora, opts={
    'user': user,
    'krb_rdns': False,
    'krb_principal': 'host/koji.fedoraproject.org',
    'krb_keytab': '~/.ssh/id_rsa',
})



KOJIURL_cdac = "http://riscv.pune.cdac.in/kojihub"

cdac_session = koji.ClientSession(KOJIURL_cdac)
cert = '~/.koji/tanay.pem'

SERVERCA = os.path.expanduser('~/.koji/koji_ca_cert.crt')
CLIENTCA = os.path.expanduser('~/.koji/koji_ca_cert.crt')
CLIENTCERT = os.path.expanduser(cert)

cdac_session.ssl_login(CLIENTCERT,CLIENTCA,SERVERCA)
```

The code begins by logging in to the respective Koji Profiles. We require to login into the fedoraproject instance and the cdac instance. The fedora project instance can be logged into via a Kerberos ticket authentication.

https://docs.fedoraproject.org/en-US/package-maintainers/Installing_Packager_Tools/#kerberos_ticket contains details of doing so from the command line.

Follow the steps to generate an SSH key pair, and ensure ~/.ssh/id_rsa is the path to the public key, which is also authenticated on the fedoraproject account profile.

To login into the CDAC instance which uses an SSL authentication, invoke the ssllogin method as shown and enter paths to your Server and Client CA Certicicate and the User Certificate, which can be obtained while setting up the koji instance.

Now for any function in the API call list**, fedora_session.function()** and **cdac_session.function()** can be used to obtain information, and perform tasks.

2. Reading from files

Obtain the list of names of all packages that need to be built, presumably from another server and write it, one package a line, into a text file called 'build.txt'. Now it might happen that we don't want to build some files from this list that we copied from somewhere, or the list is too large to search for and delete some unwanted packages manually. We can add the names of those packages to another text file called 'ignore.txt'

The following python code prepares a list 'package_name' that contains the names of all packages that need to be built after reading both the files:

```
try:
    with open('build.txt') as f:
        build_pkgs = [line.strip() for line in f.readlines()]
except FileNotFoundError:
    print("Error: 'build.txt' not found.")
    build_pkgs = []


try:
    with open('ignore.txt') as g:
        ignore_pkgs = [line.strip() for line in g.readlines()]
except FileNotFoundError:
    print("Error: 'ignore.txt' not found.")
    ignore_pkgs = []


package_name = []
for pkg in build_pkgs:
    if(pkg not in ignore_pkgs):
        package_name.append(pkg)
```

3. Obtaining Information of Packages

In order to build the packages, we need to obtain information about said packages' latest build on the fedora server. More Specifically we need the Source Control Manager's URL where the Source RPM Code is present. We obtain the required information in the following manner:

```
#function to generate info on the latest build of a package
def getInfo(package, tag):
    build = fedora_session.listTaggedRPMS(tag,latest=True,package=package)
    id = build[0][1]['build_id']
    info = fedora_session.getBuild(id)
    return {'id': id, 'name': package, 'SCMURL': info['source']}

# info of all packages to be built
packages  = []

for pack in  package_name:
    packages.append(getInfo(pack,target))
```

'packages' is now a list of dictionaries that contains basic information about the packages that need to be built. This information is obtained by inspecting the API call list and noticing what each function returns.

4. Segregating Packages

Having obtained the link, we are now tempted to immediately called the cdac_session.build() function, which will begin the build process and we can be done. But there is a detail we can take care of to save time.

Each build contains several RPM files. Each RPM file is to be built and it contains an attribute called 'arch' which specifies the architecture on which we can build it. However if the 'arch' is set to 'noarch' then it need not be built as it contains non binary files that do not need compilation. Running the build command on such RPMs can lead to wastage of time and memory. We can simply download those rpm files locally and import it to obtain whats required.

We must segregate the packages into those that contain only 'noarch' rpms other than the source rpms. We try to automate the process of downloading and importing.

The following code checks if the packages satisfy our criteria. If they do we can add them to a separate list called **no_build** and the others into **to_build**. We then try to deal with both the lists separately.

```
## function that returns true if RPM of the build are only 'src' & 'noarch'
def chkBuildArt(package):
    archs = [RPM['arch'] for RPM in fedora_session.listRPMs(package['id'])]
    for arch in archs:
        if arch != 'src' and arch != 'noarch':
            return False
    return True



## segregating packages to build and to import
to_build = []
no_build = []

for package in packages:
    if(not chkBuildArt(package)):
        to_build.append(package)
    else:
        no_build.append(package)
```

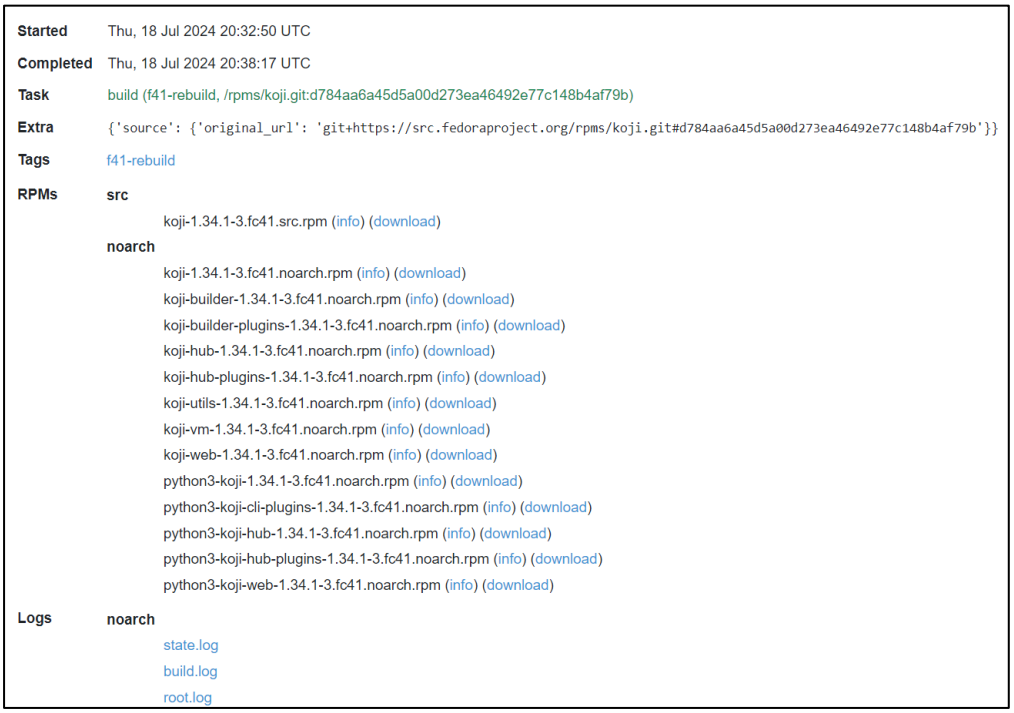
5. Building packages

For every package in to_build, we can simply call the build function in the cdac server, specify the target and Koji will take care of the rest:

```
## building packages that need to be built
for package in to_build:
    cdac_session.build(package['SCMURL'], target)
```

6. Obtaining the download link of the RPM

We now need to handle the packages in no_build. As seen the following picture such a package will have only RPMs that have ‘src’ and ‘noarch’ as their archs:



Started Thu, 18 Jul 2024 20:32:50 UTC

Completed Thu, 18 Jul 2024 20:38:17 UTC

Task build (f41-rebuild, /rpms/koji.git:d784aa6a45d5a00d273ea46492e77c148b4af79b)

Extra {'source': {'original_url': 'git+https://src.fedoraproject.org/rpms/koji.git#d784aa6a45d5a00d273ea46492e77c148b4af79b'}}

Tags f41-rebuild

RPMs

src

koji-1.34.1-3.fc41.src.rpm (info) (download)

noarch

koji-1.34.1-3.fc41.noarch.rpm (info) (download)
koji-builder-1.34.1-3.fc41.noarch.rpm (info) (download)
koji-builder-plugins-1.34.1-3.fc41.noarch.rpm (info) (download)
koji-hub-1.34.1-3.fc41.noarch.rpm (info) (download)
koji-hub-plugins-1.34.1-3.fc41.noarch.rpm (info) (download)
koji-utils-1.34.1-3.fc41.noarch.rpm (info) (download)
koji-vm-1.34.1-3.fc41.noarch.rpm (info) (download)
koji-web-1.34.1-3.fc41.noarch.rpm (info) (download)
python3-koji-1.34.1-3.fc41.noarch.rpm (info) (download)
python3-koji-cli-plugins-1.34.1-3.fc41.noarch.rpm (info) (download)
python3-koji-hub-1.34.1-3.fc41.noarch.rpm (info) (download)
python3-koji-hub-plugins-1.34.1-3.fc41.noarch.rpm (info) (download)
python3-koji-web-1.34.1-3.fc41.noarch.rpm (info) (download)

Logs **noarch**

state.log
build.log
root.log

We can also see that each RPM has a download link that we can use to download and then import the RPM. We can of course do it manually, but we want to automate it.

The issue is that the download link is not obtainable directly via the API calls. We need to find a way to construct it.

Observe the following with respect to the details of a particular RPM and its download link:


```
{'arch': 'ppc',
 'build_id': 3429,
 'buildroot_id': None,
 'buildtime': 1172712378,
 'draft': False,
 'epoch': None,
 'external_repo_id': 0,
 'external_repo_name': 'INTERNAL',
 'extra': None,
 'id': 41279,
 'metadata_only': False,
 'name': 'libapreq2-devel',
 'payloadhash': '6fa3b2c110ee1d4eb4904d3a5cb18d22',
 'release': '0.rc2.4.fc7',
 'size': 172998,
 'version': '2.09'}
```


https://kojipkgs.fedoraproject.org//packages/koji/2.09/0.rc2.4.fc7/ppc/libapreq2-devel-2.09-0.rc2.4.fc7.ppc.rpm

We can see that the download url is obtainable by putting together certain information obtained by the getRPM function.

We therefore make a function to construct the link:

```
## function that returns download url of an RPM file
def rpmURL(id):
    rpm = fedora_session.getRPM(id)
    name = rpm['name']
    version = rpm['version']
    release = rpm['release']
    arch = rpm['arch']
    nvra = name+"-"+version+"-"+release+"."+arch
    url = "https://kojipkgs.fedoraproject.org//packages/koji/"+version+"/"+release+"/"+arch+"/"+nvra+".rpm"
    return url
```

Note that now it is enough to know the RPM ID to construct the url from the function.

7. Downloading

We can import 'subprocess' which helps us access the command line. Then a simple 'wget <url>' is enough to download the RPM using the link created above.
The following code downloads the RPM in the given directory:

```
import subprocess
download_dir = '/home/tanay/downloads/'
## function to download the rpm file
def downloadFile(url,name):
    directory = download_dir+name
    command = f"wget -q {url} -P {directory}"
    try:
        subprocess.run(command, shell=True, check=True)
    except subprocess.CalledProcessError as e:
        print(f"An error occurred: {e}")
```

8. Importing

Now that the file is downloaded we can import it using the command line interface again:

```
def importRPM(path):
    koji_command = ['koji', '-p', 'cdac', 'import']
    full_command = koji_command + [path]
    try:
        subprocess.run(full_command, check=True)

    except subprocess.CalledProcessError as e:
        print(f"Error importing {package}: {e}")
```

9. Completing the Automation

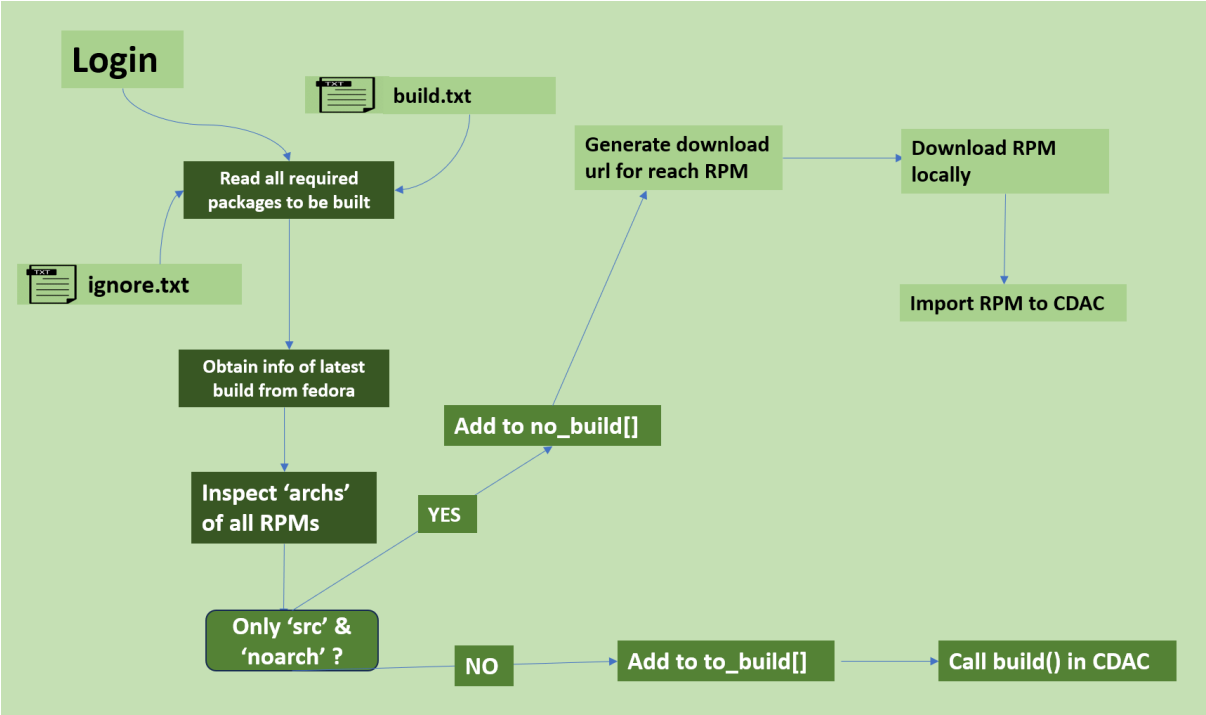
The crucial functions are created. We must run them appropriately for all RPMs in each package in no_build, and passing necessary information.

```
## importing all remaining packages
for package in no_build:
    package_name = package['name']
    upload_path = 'downloads/'+package_name
    for RPM in fedora_session.listRPMs(package['id']):
        id = RPM['id']
        nvr = RPM['nvr']
        arch = RPM['arch']
        url = rpmURL(id)
        downloadFile(url,package_name)

path=download_dir+package_name+"/"+nvr+"."+arch+".rpm"
        importRPM(path)
```

Summary

The required code is now complete. Here is a flowchart describing the flow of process:



Conclusion

At the end of my project and internship at CDAC Pune, I was successfully able to write code that provides an automation for building RPMs into the CDAC server. That code will be used for their much larger project of developing a robust Fedora OS on RISC- V architecture.

I take back with me lot of learning outcomes that I hope to make use of in the future.

References

• https://koji.fedoraproject.org/koji/

• https://koji.fedoraproject.org/koji/api

• https://pagure.io/koji/

• https://docs.pagure.org/koji/

• http://cdackoji.pune.cdac.in/koji/users