Red Hat Summit

Packaging software

for Red Hat Enterprise Linux

Carl George
Principal Software Engineer
Red Hat

Scott McBrien
Technical Contributor
Red Hat

Red Hat

RPMs

Lab: initialize

Open the link below and click the Launch → button.

red.ht/rpm

What is RPM?

- Package format used by Fedora, CentOS, RHEL, and other operating systems
- Used by packaged managers such as DNF to manage software

Why package with RPM?

- Easily install, reinstall, remove, and upgrade software
- Query and verify installed packages
- Metadata to describe package properties and relationships with other packages
- Digitally signed packages to validate authenticity
- Distribute packages in DNF repositories
- Pristine sources to ease future maintenance

What is an RPM package?

- Special archive containing files and metadata
- Two types:
  - A binary RPM contains the files to be installed on the target system
  - A source RPM (SRPM) contains source code and instructions to build a binary RPM

What is a spec file?

- Recipe for building the package
- Preamble that defines metadata about the package
- Body with several sections for various stages of the build process
- Conditionals for flexibility between operating systems, architectures, etc.

RPM macros

- Variables for text substitution in the spec file
  - Syntax: `%example` or `%{example}`
- Some macros accept parameters to influence the output
- Can be defined inside the spec file or on the system:
  - `/usr/lib/rpm/macros.d/macros.*`
  - `/etc/rpm/macros.*`
  - `~/.rpmmacros`
- Can be conditional to only expand when the macro is defined, e.g. `%{?dist}`
- Can be explored outside of the build process:
  - `rpm --eval '%example'` ⟶ evaluate a specific macro
  - `rpm --showrc` ⟶ print all defined macros

Common macros

- Filesystem paths:
  - `%{_bindir}` → `/usr/bin`
  - `%{_datadir}` → `/usr/share`
  - `%{_sysconfdir}` → `/etc`
- Operating system properties:
  - `%{rhel}` → `9`
  - `%{dist}` → `.el9`
- Build process helpers:
  - `%autosetup` → extract source code archives and apply patches
  - `%configure` → `./configure` with packaging-specific options
  - `%make_build` → `make` with packaging-specific options
  - `%make_install` → `make install` with packaging-specific options

Common macros

- Python helpers:
  - `%py3_build` → `python3 setup.py build` with packaging-specific options
  - `%py3_install` → `python3 setup.py install` with packaging-specific options
- CMake helpers:
  - `%cmake` → `cmake` with packaging-specific options
  - `%cmake_build` → `cmake --build` with packaging-specific options
  - `%cmake_install` → `cmake --install` with packaging-specific options
- Meson helpers:
  - `%meson` → `meson` with packaging-specific options
  - `%meson_build` → `meson compile` with packaging-specific options
  - `%meson_install` → `meson install` with packaging-specific options

Packaging workspace setup

- The `rpmdevtools` package includes the `rpmdev-setuptree` command
- Running that command on a system will create the following directories:
  - `~/rpmbuild/BUILD`
  - `~/rpmbuild/RPMS`
  - `~/rpmbuild/SOURCES`
  - `~/rpmbuild/SPECS`
  - `~/rpmbuild/SRPMS`

Lab: workspace setup

Your first challenge is to set up your packaging workspace.

Click the Start button and follow the on screen instructions.

Once you have completed the instructions, click the Next button.

Spec file preamble

- `Name` → name of the package, should match the spec file name
- `Version` → version of the software being packaged
- `Release` → package release, used to distinguish between different builds of the same software version
- These properties form a useful identifier know as the NVR, some examples:
  - `gawk-5.1.0-6.el9`
  - `tzdata-2023c-1.el9`
  - `virt-what-1.25-1.el9`
- `Epoch` → optional integer used to override normal version-release sorting order

Spec file preamble

- `Summary` → short one line summary of the package
- `License` → identifier for the license of the software being packaged
- `URL` → URL for more information about the software
- `Source` → file name or URL of a file needed to build the software, such as the source code archive or default config files (can be used multiple times)
- `Patch` → file name or URL of patch to apply to the source code (can be used multiple times)
- `BuildArch` → defaults to the build system architecture, can be set to noarch for packages with no architecture-dependent files

Spec file preamble

- `BuildRequires` → other packages needed to build this package
- `Requires` → other packages needed to install this package
- `Conflicts` → other packages that cannot be installed at the same time
- `Obsoletes` → used to replace one package with another
- `Provides` → allows other packages to refer to this package by another name
- `Recommends` → weak requires, installed by default but can be removed
- `Supplements` → reverse recommends

Spec file body

- `%description` → description of the package, can span multiple lines
- `%prep` → commands to prepare the source code for building (unpacking archives, applying patches, etc.)
- `%build` → commands to build the software
- `%install` → commands to copy the desired build artifacts into the appropriate filesystem locations relative to the buildroot
- `%check` → commands to test the software, e.g. running unit tests
- `%files` → list of files and directories that will be installed on the target system
- `%changelog` → record of changes that have happened to the package between different versions and releases

File attributes

- In `%files`, each file and directory can be preceded by an attribute
  - `%dir` ⟶ own just the directory itself, but not its content
  - `%config` ⟶ mark a file as configuration
  - `%config(noreplace)` ⟶ mark a file as configuration and prevent it from being overwritten on updates
  - `%attr(<mode>,<user>,<group>)` ⟶ set non-default permissions or ownership
- Some attributes accept relative paths to copy files into the `%{buildroot}`
  - `%license` ⟶ copy file to `/usr/share/licenses/%{name}/` and mark as license
  - `%doc` ⟶ copy file to `/usr/share/doc/%{name}/` and mark as documentation

Creating spec files

- From scratch
- Copy a similar spec file and adjust as needed
- Automatic templates from a text editor
- The `rpmdevtools` package includes the `rpmdev-newspec` command to create a new spec files from templates

Creating changelog entries

- By hand
- Copy another changelog entry and adjust as needed
- Text editor plugins
- The `rpmdevtools` package includes the `rpmdev-bumpspec` command to create new changelog entries and simultaneously adjust version and release tags

Building RPMs

- RPMs are built with the `rpmbuild` command
- This commands expects the directory structure from `rpmdev-setuptree`
- Several build modes:
  - `-bs` ⟶ build an SRPM from a spec file and sources
  - `-bb` ⟶ build an RPM from a spec file and sources
  - `-ba` ⟶ build both an SRPM and an RPM from a spec file and sources
  - `--rebuild` ⟶ build an RPM from an SRPM
- Example:
  - `rpmbuild -ba SPECS/bello.spec`

Quality checking RPMs

- `rpmlint` is a linter tool for spec files, SRPMs, and RPMs
- Identifies common packaging errors
- Ideal to resolve all errors and warnings, but not always possible
- `rpm` can query an uninstalled RPMs by using the `--package` flag
- Consider the following `rpm` flags:
  - `--info`
  - `--list`
  - `--requires`
  - `--provides`
  - `--conflicts`
  - `--changelog`

Lab: packaging bello

Your next challenge is to package bello, a “Hello World” program written in Bash.

Click the Start button and follow the on screen instructions.

Once you have completed the instructions, click the Next button.

Installing build requirements

- Build requirements in the spec file must be installed on the build host
- Can be installed manually or with `dnf builddep`

Lab: packaging cello

Your next challenge is to package cello, a “Hello World” program written in C.

Click the [Start] button and follow the on screen instructions.

Once you have completed the instructions, click the [Next] button.

Lab: packaging pello

Your next challenge is to package pello, a “Hello World” program written in Python.

Click the Start button and follow the on screen instructions.

Once you have completed the instructions, click the Next button.

Mock

- Drawbacks of using `rpmbuild` directly:
  - Build requirements must be installed on the build host
  - Build requirements that are already installed are easy to forget in the spec file
  - Can only build RPMs targeting the same operating system and release as build host
- `mock` is a tool that builds RPMs in isolated chroots
  - Uses `rpmbuild` internally
  - Build requirements are installed in the chroot, not on the build host
  - Helps identify missing build requirements
  - Can build RPMs for different operating systems and releases than the build host
  - Chroots are automatically created and removed
  - Widely used (Koji, Copr, fedpkg, and more)

Lab: building with mock

Your final challenge is to build the pello package using the `mock` tool.

Click the Start button and follow the on screen instructions.

Once you have completed the instructions, click the Next button.

Containers

Lab: initialize

Open the link below and click the Launch → button.

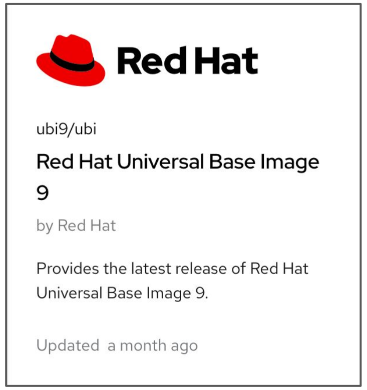
red.ht/containerize

How to build?

- From Scratch?
  - Ultimate customization
  - You also assume more maintenance burdens

- From Base Image?
  - Which one?
  - From who?
  - How is it managed from the provider?

Base Images from Red Hat



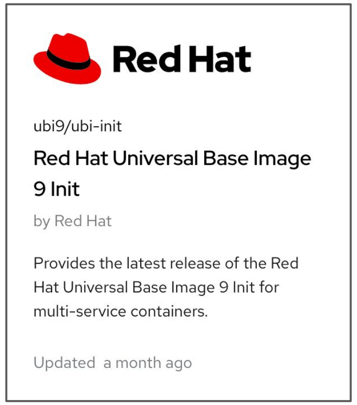
ubi9/ubi

Red Hat Universal Base Image 9

by Red Hat

Provides the latest release of Red Hat Universal Base Image 9.

Updated a month ago



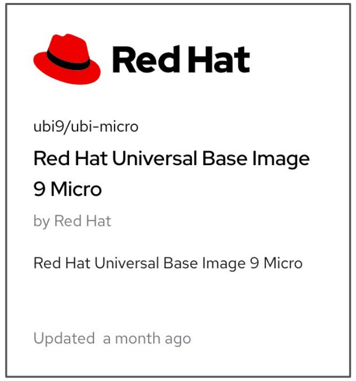
ubi9/ubi-init

Red Hat Universal Base Image 9 Init

by Red Hat

Provides the latest release of the Red Hat Universal Base Image 9 Init for multi-service containers.

Updated a month ago



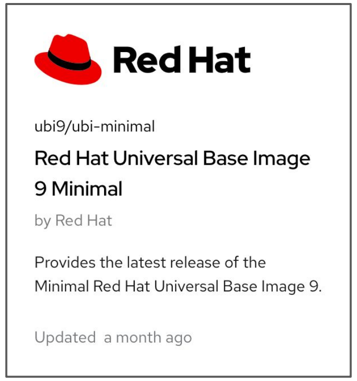
ubi9/ubi-micro

Red Hat Universal Base Image 9 Micro

by Red Hat

Red Hat Universal Base Image 9 Micro

Updated a month ago



ubi9/ubi-minimal

Red Hat Universal Base Image 9 Minimal

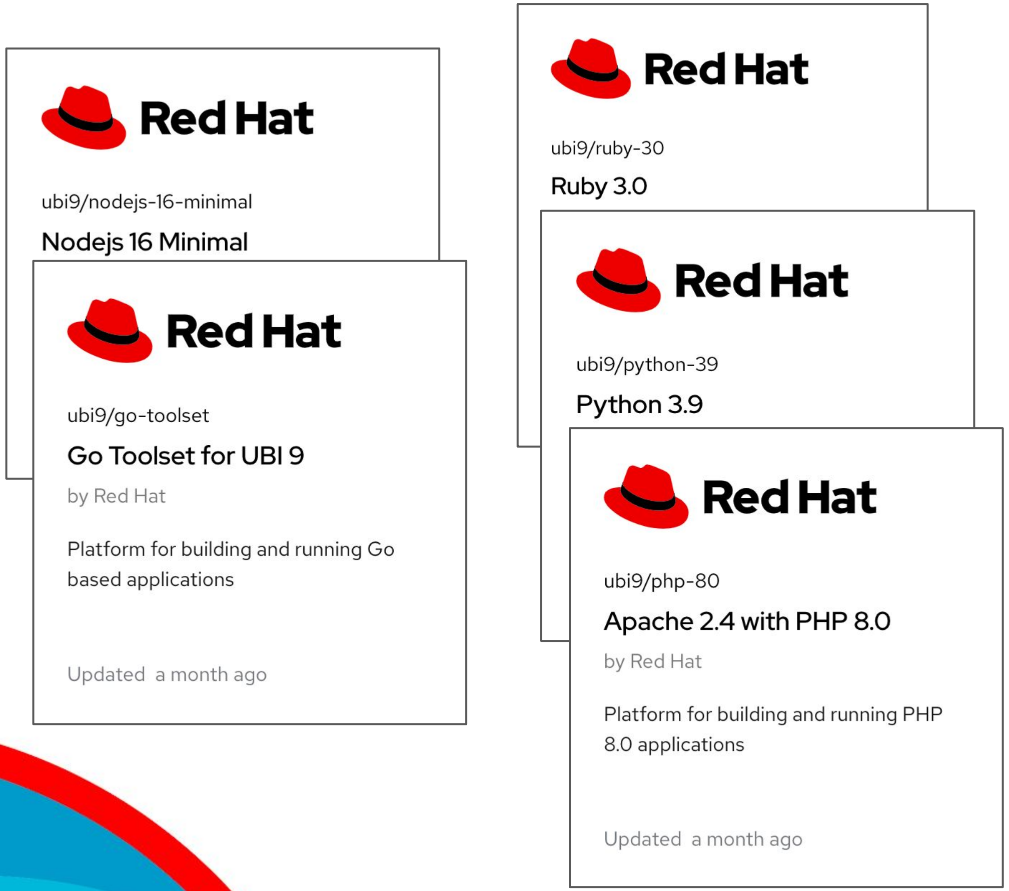
by Red Hat

Provides the latest release of the Minimal Red Hat Universal Base Image 9.

Updated a month ago

- Built using software from Red Hat Enterprise Linux
- Redistributable without a subscription
- Regularly maintained
- Available from Red Hat Container Catalog or Dockerhub
- Several options to choose from

Language runtime variants



Red Hat

ubi9/nodejs-16-minimal

Nodejs 16 Minimal

Red Hat

ubi9/go-toolset

Go Toolset for UBI 9

by Red Hat

Platform for building and running Go based applications

Updated a month ago

Red Hat

ubi9/ruby-30

Ruby 3.0

Red Hat

ubi9/python-39

Python 3.9

Red Hat

ubi9/php-80

Apache 2.4 with PHP 8.0

by Red Hat

Platform for building and running PHP 8.0 applications

Updated a month ago

- Uses UBI
- Includes base language runtime and additional Red Hat provided libraries
- Not always size-optimized

Using RPMs

- Many images include `dnf` or `microdnf`
- UBI images are configured with Red Hat UBI repos (which are a subset of RHEL)
- You can add additional repos

Making an “archive”

- Position files where they need to go
- Use multiple content sources
- Equally automatable with a Containerfile

Building with `buildah`

```
$ buildah from registry.access.redhat.com/ubi9/ubi
```

- Downloads base image
- Mounts working copy of container filesystem

Run commands with `buildah`

```
$ buildah run ubi-working-container -- dnf -y install RPM-NAME
```

- Runs command within change-rooted container environment
- –– separates host portion of command from change-rooted, in container, command

Positioning files with `buildah`

```
$ buildah run ubi-working-container -- dnf -y install RPM-NAME
```

- Runs command within change-rooted container environment
- –– separates host portion of command from change-rooted, in container, command

Positioning files with `buildah`

```
$ buildah copy ubi-working-container <file/dir> <dir-in-container>
```

- Copies content into the target container

Lab: Containerizing Applications

In this lab you will build two containerized applications

1) Install software from the UBI repos and 3rd Party repository (EPEL)

2) Position files pulled down from a github project & install UBI-provided software

Flatpaks

An alternate packaging for RHEL

- Flatpaks are supported on RHEL
- User-installable software
- Requires a redhat.com account with RHEL entitlements
- Flatpaks from Red Hat are maintained with the same rigor as our RPM packaged content

Installing `flatpak`

```
# dnf install flatpak
```

- Installs `flatpak` -- users can now use flatpak to configure repositories and add software

Enabling a repo

```
$ flatpak remote-add rhel https://flatpaks.redhat.io/rhel.flatpakrepo
```

- Adds the `flatpak` repo for a user -- users can add additional repos from 3rd party providers into their configuration as well

Authenticating

```
$ podman login registry.redhat.io
```

- The Red Hat `flatpak` repo is available to authenticated users

  e.g. those with a redhat.com customer portal account

Installing a `flatpak`

```
$ flatpak install rhel firefox
```

Running a `flatpak`

```
$ flatpak run org.mozilla.Firefox
```

- When running a flatpak, you must use the **Application ID**, which can be obtained, after install, from `flatpak list`

We would love to talk to you!

Our Red Hat User Experience team would love to talk to you about your experience using console.redhat.com, Image Builder, Red Hat Insights, or any Red Hat Product!

Sign up to give user feedback:

Or come visit the Experience Zone!

If you see us, come say hi!

Katie Riker
kriker@redhat.com

Melissa Grimes
mgrimes@redhat.com

Red Hat Summit

Thank you

in linkedin.com/company/red-hat

f facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

Red Hat