Packaging and developing with flatpak

Alexander Larsson
Red Hat

David King
Red Hat

What is Flatpak?

- App installation system
- Desktop apps
- Cross distribution
- Bundling
- Runtimes
- Sandboxed
- OSTree

Lets reimplement flatpak

- Just the core
  - remote-add
  - install/update
  - run
- Uses git and common unix tools
- Not production quality!
- Good working model for understanding

flatpak remote-add flathub \
    https://flathub.org/repo/flathub.flatpakrepo

```
# First time initialization of installation
mkdir app runtime exports
git init repo

# Add the git url as a named remote
git -C repo remote add flathub  \
      https://github.com/flathub.git
```

flatpak install flathub org.gnome.gedit

```
# Pull a branch from the remote (PULL)
git -C repo fetch flathub app/gedit

# Check out a copy of the current revision (DEPLOY)
CURRENT=$(git -C repo rev-parse flathub/app/gedit)
mkdir -p app/gedit/
git clone --shared --branch app/gedit repo app/gedit/$CURRENT
echo flathub > app/gedit/$CURRENT/origin

# Atomically switch which rev is current
ln -sf $CURRENT app/gedit/active

# Copy all desktop files and icons to a global directory
cp -r app/gedit/active/export/* exports/
```

flatpak update org.gnome.gedit

```
# We always update from the same remote
# as we installed from
REMOTE=`cat app/gedit/active/origin`

# Remember old version
OLD=$(readlink app/gedit/active)

# ... same as install $REMOTE app/gedit here ...

# Remove old deployment
rm -rf $OLD
```

Installing/updating runtimes

Same as app, but with “runtime”, not “app” in branch and directory names

flatpak run org.gnome.gedit

```
# Create new root and mount tmpfs on it
mkdir root && mount -t tmpfs tmpfs root
# Create root dirs
(cd root;
  mkdir usr app tmp dev proc var;
  ln -s usr/lib lib)
mount -o bind,ro runtime/gnome/active/files root/usr
mount -o bind,ro app/gedit/active/files root/app
# Mount proc, sys, etc..
# Run the app in the chroot
chroot root /app/bin/gedit
```

Important differences

- OSTree naturally handles multiple checkouts
- Deploys are hardlinked into repo
- Namespaces, seccomp, etc, not just chroot
- “Chroot” setup more complex
- “Chroot” only exists inside namespace

...and lots of features, details and polish

Layout of an installation

- Multiple installations
  - User: `~/.local/share/flatpak`
  - System: `/var/lib/flatpak`
  - Custom
- Contents:
  - `repo/`
  - `app/`
  - `runtime/`
  - `exports/`
  - `appstream/`

Layout of an application

- Deployed as:
  - `app/com.spotify.Client/x86_64/stable/`
    - `d6760516...`
    - `active → d6760516...`
- Contains
  - `metadata`
  - `files/`
  - `export/`
  - `deploy`

Application identifier

- Important to get right
- Unique identifier for application
- Reverse dns-style: org.foo.App
- If using dbus, must be same as dbus name
- Defines the name of the desktop file (*$appid*.desktop)
- All exported files must have this prefix

Branches

- One application can have multiple branches
  - Flathub: `stable`
  - Gnome nightly builds: `master`
- Major version for runtimes
- Can be parallel installed
- Only one “current”
  - Exports files
  - Change with: `flatpak make-current`
- Share application data
- Alternative: Separate app id

Refs

- OSTree full branch names
- app/$APPID/$ARCH/$BRANCH
  - app/org.pitivi.Pitivi/x86_64/stable
- runtime/$ID/$ARCH/$BRANCH
  - runtime/org.gnome.Platform/x86_64/3.28
- Can be shortened in CLI
  - org.pitivi.Pitivi
  - org.pitivi.Pitivi//stable
  - etc

What is a runtime?

- Like a minimal distribution
- Only app dependencies
- An application specifies the runtime to use
- Multiple runtimes can be used in parallel
- Also supplies a SDK

What is in a runtime?

- libc
- Basic library dependencies
- Basic unix tools
- Security updates

What is a runtime not?

- Another name for packages
- Something you should create
- A way to avoid bundling dependencies

How do I chose a runtime?

- Depends on your application
- Generally
  - Larger runtime
    - Shorter supported lifespan
    - Need to bundle less
  - Smaller runtime
    - Longer supported lifespan
    - May need to bundle more
- org.freedesktop.Platform runtime
  - 1.6 is current
  - 1.8 in progress
- org.gnome.Platform runtimes:
  - 3.24, 3.26, 3.28 (based on freedesktop 1.6)
  - master (based on 1.8)
- org.kde.Platform runtimes:
  - 5.9, 5.10, 5.11 (based on freedesktop 1.6)

AppData / AppStream

- Freedesktop standard
- Describes your application
  - Name
  - Description
  - Version
  - Screenshots
  - Etc
- Combined into one ostree branch
- Used by gnome-software, flathub.org, etc

AppStream sample

```
<components version="0.8" origin="flatpak">
  <component type="desktop">
    <id>org.gnome.Characters.desktop</id>
    <translation type="gettext">org.gnome.Characters</translation>
    <name>GNOME Characters</name>
    <name xml:lang="sv">Tecken</name>
    <summary>Character map application</summary>
    <description><p>Characters is …</p> </description>
    <icon height="64" width="64">64x64/org.gnome.Characters.png</icon>
    <categories> <category>Utility</category> </categories>
    <keywords> <keyword>emoji</keyword> </keywords>
    <project_license>BSD-3-Clause</project_license>
    <url type="homepage">https://wiki.gnome.org/Design/Apps/CharacterMap</
url>
    <screenshots>
        <image>https://gnome.org/character-map/screenshot.png</image>
    </screenshots>
  </component>
```

flatpak build

- Similar to flatpak run, but for builds
- Uses build-directory instead of deployment
- /app is writable
- Create a build-directory using:
  - flatpak build-init <dirname> \
    <app-id> <sdk> <runtime> <runtime-version>
- Typically used multiple time each build
- End with flatpak build-finish
- Export to repo with build-export

Flatpak build example

```
$ flatpak build-init dir org.the.app \
 org.gnome.Platform org.gnome.Sdk 3.28
$ # extract sources to build
$ flatpak build dir ./configure --prefix=/app
$ flatpak build dir make
$ flatpak build dir make install
$ flatpak build-finish dir --socket=x11
$ flatpak build-export dir /the/repo dir
# Test the app:
$ flatpak build dir sh
```

Automating builds

- Manually using flatpak build is very tedious
- Most applications are built in the same way
- Enter flatpak-builder:
  - flatpak-builder *[--repo=repo]* <build-dir> <manifest>
- Manifests are json or yaml
- Describes a set of sources to download/prepare
- Automates calls to flatpak build
- Caches build steps

```
app-id: org.gnome.Dictionary
runtime: org.gnome.Platform
runtime-version: '3.28'
sdk: org.gnome.Sdk
command: gnome-dictionary
finish-args:
- "--socket=x11"
- "--share=network"
build-options:
  cflags: "-O2 -g"
cleanup:
- "/include"
- "*.a"
modules:
- name: gnome-dictionary
  buildsystem: autotools
  sources:
  - type: archive
    url: https://gnome.org/.../gnome-dictionary-3.28.0.tar.xz
    sha256: efb36377d46eff9291d3b8fec37ba...4716121
```

flatpak-builder phases

- Download sources
- build-init app dir
- For each ${module}
  - Create build dir
  - Extract and prepare sources
  - Build
    - ./autogen.sh
    - ./configure --prefix=/app ${config-opts}
    - make ${make-args}
    - make install ${make-install-args}
  - Post install
  - Extract debug info
  - Remove build dir
  - Collect file list for module
- Cleanup
  - Local
  - Global
- build-finish app dir
- Write resolved manifest.json
- Optional: build-export app dir to repository

Important module options

- Buildsystem
  - autotools, cmake, cmake-ninja, meson, simple
- build-commands (especially for simple)
- config-opts
- make-args / make-install-args
- no-autogen / rm-configure
- build-options (cflags, env, build-args)
- post-install
- cleanup
- Modules

See `man flatpak-manifest` for more

Source types

- Main sources
  - archive
  - git
  - bzr
  - svn
- Build tweaks
  - patch
  - file
  - script
  - shell (runs code)
- Special
  - dir
  - extra-data

flatpak-builder layout

- All data is stored in `.flatpak-builder`:
  - `downloads/`
    - Downloaded files and archives
  - `git/`
    - Mirrored git repos
  - `build/$module/`
    - This is where sources are extracted and built
  - `cache/`
    - Build cache
  - `ccache/`
    - Optional ccache (enable with --ccache)
- All builds are run as `/run/build/$module`

Debugging a failed build

- Enter sandbox and rebuild:
  - flatpak-builder --run <build-dir> <manifest> sh
  - $ cd /run/build/module
  - # Edit .flatpak-builder/build/module/…
  - $ make
  - $ gdb ./the-app
- Other useful options:
  - --keep-build-dirs
  - --force-clean
  - --stop-at=MODULE
  - --install

Debugging applications

- Install SDK and debug info
  - `flatpak install \`
    `org.gnome.Sdk//3.28 \`
    `org.gnome.Sdk.Debug//3.28 \`
    `org.the.App.Debug`
- Run shell in SDK:
  - `flatpak run -d --command=sh org.the.App`
  - `$ gdb /app/bin/the-app`

Sandboxing

- Everything is sandboxed
- Except
  - Per-app data
    - ~/.var/app/$appid
  - session dbus
    - App can own its app id on the bus
    - Others can talk to app
- Anything above this must be requested
- Specified in finish-args

Common sandboxing options

- X11 access: "--socket=x11", “--share=ipc"
- Wayland access: "--socket=wayland"
- Network access: "--share=network"
- Sound support: “--socket=pulseaudio”
- OpenGL rendering: “--device=dri”
- Filesystem access:
  - --filesystem=host
  - --filesystem=home
  - --filesystem=~/subdir
  - --filesystem=xdg-downloads
  - ...
- See `man flatpak-build-finish` for rest

Sandboxing dconf

- Giving it required access:

```
"--filesystem=xdg-run/dconf",
"--filesystem=~/.config/dconf:ro",
"--talk-name=ca.desrt.dconf",
"--env=DCONF_USER_CONFIG_DIR=.config/dconf"
```

- There are plans for a sandbox aware dconf

Portals – xdg-desktop-portal

xdg-desktop-portal:

- Document
- File chooser
- Open URI/File
- Compose Email
- Print
- NetworkMonitor
- Notification
- Screenshot
- Screencast
- Proxy resolver
- Inhibit Lock/Screensaver

flatpak portal:

- Spawn
  - restart, sandbox
  - flatpak-spawn

Breaking out of the sandbox

- --filesystem=host
  - /run/host/usr
  - /run/host/etc
- --allow=devel
  - ptrace
  - perf
- Launch commands on host
  - --talk-name=org.freedesktop.Flatpak
  - org.freedesktop.Flatpak.Development.HostCommand
  - flatpak-spawn --host

Sdk extensions

- Some development tools are not in the Sdk
  - Golang
  - Rust
  - Mono
  - Java
- Use SDK extensions:

```
sdk-extensions:
  - org.freedesktop.Sdk.Extension.rust-stable
build-options:
  append-path: /usr/lib/sdk/rust-stable/bin
```

Renaming exports

- Exported files have to match app id
  - Icons
  - Desktop files
  - AppData files
- Require modification to the file contents
- flatpak-builder options:
  - rename-icon: wrong.png
  - copy-icon: true
  - rename-desktop-file: wrong.desktop
  - rename-appdata-file: wrong.appdata.xml

flatpak-builder-tools

- Avoid build systems that need network access
  - Npm
  - Pip
  - Ruby gems
  - Yarn
- Generates include snippet
- Pre-seeds the downloads
- Typically uses lock files

Base Apps

- Simple way to share dependencies
- Dependencies “resolved” at build-time
- Only for large/common dependencies
- In flathub:
  - io.atom.electron.BaseApp
  - org.electronjs.Electron2.BaseApp
  - io.qt.qtwebkit.BaseApp
- To use:
  - `base: io.atom.electron.BaseApp`

Bisecting

```
$ flatpak remote-info --log flathub org.gnome.frogr

...

History:

 Subject: Update git URL after migration to gitlab.gnome.org (cc75ab90)

 Date: 2018-05-24 16:58:35 +0000

 Commit: 7267b5ece610c0cc751a117fe0a5da92e02790ff90354491bcccf83198f3505b


 Subject: Update to frogr 1.4 and to the GNOME 3.26 runtime (cc48dcb3)

 Date: 2017-12-28 00:18:50 +0000

 Commit: b4f20b6575778f837631a56fee3a9b07f5733702a6820e6f6bbf54eca8a4c910

...

$ flatpak update --
commit=b4f20b6575778f837631a56fee3a9b07f5733702a6820e6f6bbf54eca8a4c910
org.gnome.frogr
```

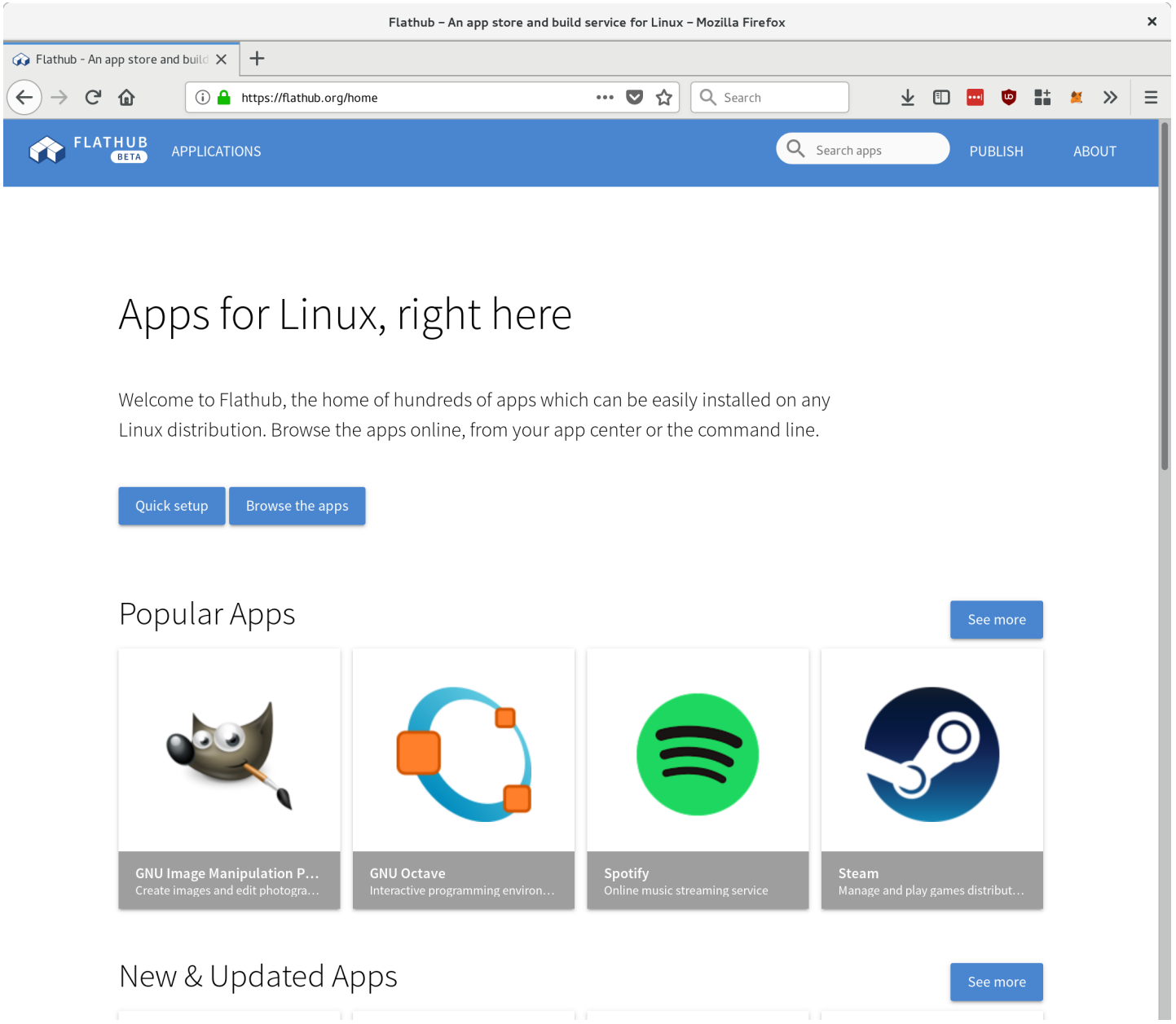
Other arches

- Build other compatible arch
  - flatpak install --arch=i386 ...
  - flatpak-builder --arch=i386 …
- With qemu-user-static installed
  - Userspace emulation
  - --arch=arm
  - --arch=aarch64

Distributing your app

- Put repository on web server
- Maintain repo with flatpak build-update-repo
  - Create static deltas
  - Update appstream branch
  - Set title, default branch
  - Prune old versions
- Sign official builds with GPG key

Alternatively - Flathub



Adding to flathub

- New apps: File PR against
  - https://github.com/flathub/flathub/tree/new-pr
- Review guidelines
  - https://github.com/flathub/flathub/wiki/Review-Guidelines
- After initial review, will be merged to new repo
  - https://github.com/flathub/org.gnome.Polari
- Commits to master are auto-built by buildbot:
  - https://flathub.org/builds/
- Request test-builds of pull requests
  - “bot, build”

Now its your turn!

Start with:

`https://people.gnome.org/~alexl/workshop-template.yaml`

Example manifests:

- `https://github.com/flathub/...`

how to play with a runtime

how to look for things to copy on flathub

Common Issues: pkg-config variable dirs

- Some pkg-config variables point to dirs in /usr
- But you want to bundle it in /app
- Solution, use PKG_CONFIG_* env vars:

```
{
    "name": "libgee",
    "build-options" : {
        "env": {
            "PKG_CONFIG_GOBJECT_INTROSPECTION_1_0_GIRDIR": "/app/share/gir-1.0",
            "PKG_CONFIG_GOBJECT_INTROSPECTION_1_0_TYPELIBDIR": "/app/lib/girepository-1.0"
        }
    },
    "sources": [
        {
            "type": "archive",
            "url": "https://download.gnome.org/sources/libgee/0.18/libgee-0.18.0.tar.xz",
            "sha256": "4ad99ef937d071b4883c061df40bfe233f7649d50c354cf81235f180b4244399"
        }
    ]
}
```

Common Issues: What is in the runtime

- Run a shell in the sdk
  - flatpak run --command=sh org.gnome.Sdk//3.28
- Run a debug shell in the sdk:
  - Flatpak run --command=sh -d org.gnome.Sdk//3.28