UC-35

Automated Software Packaging for Linux using AppImage

IT Capstone Program

Project Sponsor: Tsai-Tien Tseng

Created by: Blair Hill, Robert Ryan, Dylan Parker, Bishwo Marhatta, and Sammi Figueroa

KENNESAW STATE UNIVERSITY

Project Description

- Automate packaging Linux software programs into AppImages with scripts written in bash script
- AppImages are a self-contained executable that is portable across different Linux distributions
- Specifically focused on packaging bioinformatics software used

Research Areas

- AppDir structure as the basis for creating an AppImage
- How the foundational software "Appimagetool" is used to turn AppDirs into AppImages
- Higher-order software that uses "Appimagetool" under the hood to collect and package software
- Linux directory structure and common libraries
- How to find dependencies for a piece of software to include in the AppImage

```
AppDir/
├── AppRun
├── seaview.desktop
├── seaview.png
└── usr
    ├── bin
    │   ├── clustalo
    │   ├── example.nxs
    │   ├── Gblocks
    │   ├── muscle
    │   ├── PhyML-3.1_linux64
    │   ├── seaview
    │   ├── seaview.html
    │   └── treerecs
    ├── lib
    └── share
        ├── applications
        │   └── seaview.desktop
        └── icons

6 directories, 12 files
```

Figure 1. Sample AppDir structure (Seaview)

```
Generating squashfs...
Parallel mksquashfs: Using 4 processors
Creating 4.0 filesystem on seaview-x86_64.AppImage, block size 131072.
[=============================================================================-] 169/169 100%

Exportable Squashfs 4.0 filesystem, gzip compressed, data block size 131072
        compressed data, compressed metadata, compressed fragments,
        compressed xattrs, compressed ids
        duplicates are removed
Filesystem size 8011.77 Kbytes (7.82 Mbytes)
        42.78% of uncompressed filesystem size (18728.24 Kbytes)
Inode table size 1030 bytes (1.01 Kbytes)
        46.88% of uncompressed inode table size (2197 bytes)
Directory table size 603 bytes (0.59 Kbytes)
        55.78% of uncompressed directory table size (1081 bytes)
Number of duplicate files found 2
Number of inodes 50
Number of files 39
Number of fragments 10
Number of symbolic links  1
Number of device nodes 0
Number of fifo nodes 0
Number of socket nodes 0
Number of directories 10
Number of ids (unique uids + gids) 1
Number of uids 1
        root (0)
Number of gids 1
        root (0)
Embedding ELF...
Marking the AppImage as executable...
Embedding MD5 digest
Success
```

Figure 2. Script creating an AppImage

Methodology

- Study documentation of AppImage, related projects, and Linux
- Implement and test bash scripts
- Study open-source project code for solutions
- Iterate on our scripts

Preliminary Results

- Our initial automation script utilized outside tools like “Appimage-builder”
- Created a dependency on Debian-based Linux distribution's APT software package manager and external software development
- Shifted focus and created a script that was devoid of external dependencies

Conclusion

- Creating a solution that relies on outside software is risky and unsustainable in the long term
- Automating AppImage creation is possible

Future Potential

- Our final script solution automates AppImage creation of fully installed software
- Automating the installation of software would be difficult but theoretically possible
- Further automation could be achieved with retrieval using curl and git clones

Intellectual Merit

- Automation accelerates the initial creation of many AppImages at once
- Automation simplifies AppImage maintenance when a new version of the software is released
- Local(ex. USB) or remote(ex. server) repositories of AppImages streamline setting up new research machines
- Process could be expanded to other subsets or a wider domain of software applications

Abstract

Often many scientists and researchers use different versions of Linux to run software which can cause dependency errors, and makes in difficult to create a functioning workstation. Many of the software that Kennesaw State University staff and students use requires specific libraries, language packages, and system permissions to run correctly. Once the user finally has the software installed, future installation dependencies can break other software in which they are also reliant on a different release.

One of the many answers to this issue is a software packaging solution called AppImage. AppImage creates a functional, portable, and quick solution that bakes software and dependencies into one executable. This allows a novice user to take the portable software to any Linux system and execute without the need to then find all the necessary items that may not be on the current system. This software packaging also makes it possible for developers to package the exact versions of dependencies with software that eliminates complicated matrices from the base system.

Currently, many users and administrators use pre-created AppImages that were painstakingly created by manual packaging and are difficult to reproduce. Through our capstone project, we have set out in creating a functioning BASH script that automates packaging at scale. This script allows the common user to present most software and creates functional AppImages. We anticipate this will greatly enable the Bioinformatics and Linux community in consuming software.