Application Containers

Introdução Engenharia Informática

Mário Antunes

October 20, 2025

Exercises

Practical Exercises: Flatpak & AppImage

Objective: This workshop will guide you through the fundamentals of application packaging. You will start with a simple "Hello World" and progress to packaging a complete Python GUI application with its dependencies.

---

0. Setup: Configure Your Workbench (Revised)

1. **Update your system:**

   ```
   $ sudo apt update && sudo apt upgrade -y
   ```

2. **Install tools:**

   ```
   $ sudo apt install wget flatpak flatpak-builder \
   python3 python3-pip python3-venv
   ```

3. **Add Flathub:** This gives you access to the runtimes and SDKs needed for building.

   ```
   $ flatpak remote-add --if-not-exists \
   $ flathub https://flathub.org/repo/flathub.flatpakrepo
   ```

4. **Install `appimagetool`:** We will download the tool and place it in `~/.local/bin`, which is in your user's default `$PATH`.

   ```
   $ mkdir -p ~/.local/bin
   $ wget -O appimagetool \
   "https://github.com/AppImage/AppImageKit/\
   releases/download/continuous/appimagetool-x86_64.AppImage"
   $ chmod +x appimagetool
   $ mv appimagetool ~/.local/bin/
   ```

5. **Apply the PATH change: Log out and log back in.** To verify it's working, open a new terminal and type:

   ```
   $ appimagetool --version
   ```

---

1. "Hello World" 🌍

Let's package a simple shell script.

1.A: Flatpak "Hello World"

Flatpak uses a "manifest" file to define the build.

1. Create a directory for this exercise:

```
$ mkdir ex1-flatpak && cd ex1-flatpak
```

2. Create the application script, `hello.sh`:

```
#!/bin/sh
echo "Hello from a Flatpak Sandbox!"
```

3. Create the manifest file, `pt.ua.deti.iei.HelloWorld.yml`:

```
app-id: pt.ua.deti.iei.HelloWorld
runtime: org.freedesktop.Platform
runtime-version: '25.08'
sdk: org.freedesktop.Sdk
command: hello.sh

modules:
  - name: hello-module
    buildsystem: simple
    build-commands:
      # Installs the script into the sandbox's /app/bin/ folder
      - install -Dm755 hello.sh /app/bin/hello.sh
    sources:
      # Tells the builder to find 'hello.sh' in our project dir
      - type: file
        path: hello.sh
```

4. **Build the package:** This command builds and installs your.

```
$ flatpak-builder --user --install --install-deps-from=flathub \
--force-clean build-dir pt.ua.deti.iei.HelloWorld.yml
```

5. **Run and Cleanup:**

```
$ flatpak run pt.ua.deti.iei.HelloWorld
$ flatpak uninstall --user pt.ua.deti.iei.HelloWorld
```

`cd ..` to exit the directory.

1.B: AppImage "Hello World"

AppImage works by bundling an entire directory (`AppDir`).

1. Create a directory for this exercise:

```
$ mkdir ex1-appimage && cd ex1-appimage
```

2. Create the `AppDir` and the main `AppRun` script:

```
$ mkdir -p HelloWorld.AppDir
$ echo '#!/bin/sh' > HelloWorld.AppDir/AppRun
$ echo 'echo "Hello from an AppImage!"' >> HelloWorld.AppDir/AppRun
$ chmod +x HelloWorld.AppDir/AppRun
$ touch HelloWorld.AppDir/icon.png
```

3. Create a file named `HelloWorld.AppDir/hello.desktop` and fill it:

```
[Desktop Entry]
Name=Hello
Exec=AppRun
Icon=icon
Type=Application
Categories=Utility;
```

4. **Build the package:** Use the `appimagetool` you installed in Step 0. If necessary change the `ARCH` varible to `arm64`

```
$ ARCH=x86_64 appimagetool HelloWorld.AppDir
```

This will create `Hello-x86_64.AppImage`.

5. **Run and Cleanup:**

```
$ chmod +x Hello-x86_64.AppImage
$ ./Hello-x86_64.AppImage

# Cleanup
$ rm -rf Hello-x86_64.AppImage
```

`cd ..` to exit the directory.

---

2. Python CLI App: ASCII Tree 🌳

Let's package a simple Python CLI app. We'll create a `pytree.py` script that recursively lists directories in a tree format.

2.A: Run with Virtual Environment (Venv)

First, let's run the app natively to confirm it works.

1. Create a project directory:

```
$ mkdir ex2-pytree && cd ex2-pytree
```

2. Create the `pytree.py` script:

```
#!/usr/bin/env python3
import os
import sys

def tree(startpath):
    """Prints a directory tree."""
    for root, dirs, files in os.walk(startpath):
        # Don't visit .venv or __pycache__
        if '.venv' in dirs:
            dirs.remove('.venv')
        if '__pycache__' in dirs:
            dirs.remove('__pycache__')

        level = root.replace(startpath, '').count(os.sep)
        indent = '    ' * (level - 1) + '   ' if level > 0 else ''

        print(f'{indent}  {os.path.basename(root)}/')

        subindent = '    ' * level + '   '
        for f in files:
            print(f'{subindent}  {f}')

if __name__ == "__main__":
    # Use current directory or a specified path
    path = sys.argv[1] if len(sys.argv) > 1 else '.'
    tree(os.path.abspath(path))
```

Make it executable: `chmod +x pytree.py`

3. **Run the app:**

```
$ ./pytree.py
# Try it on another directory
$ ./pytree.py /tmp
```

`cd ..` to exit the directory.

2.B: Package `pytree` as a Flatpak

1. Create a project directory:

   ```
   $ mkdir ex2-flatpak && cd ex2-flatpak
   ```

2. Copy the `pytree.py` file from the previous exercise:

   ```
   $ cp ../ex2-pytree/pytree.py .
   ```

3. Create the manifest `pt.ua.deti.iei.pytree.yml`:

   ```
   app-id: pt.ua.deti.iei.pytree
   runtime: org.gnome.Platform
   runtime-version: '48'
   sdk: org.gnome.Sdk
   command: pytree.py

   modules:
     - name: pytree
       buildsystem: simple
       build-commands:
         - install -Dm755 pytree.py /app/bin/pytree.py
       sources:
         - type: file
           path: pytree.py
   ```

   Note: We use `org.gnome.Platform` because it includes Python by default.

4. **Build and Install:**

   ```
   $ flatpak-builder --user --install --install-deps-from=flathub \
   --force-clean build-dir pt.ua.deti.iei.pytree.yml
   ```

5. **Run and Cleanup:**

   ```
   $ flatpak run pt.ua.deti.iei.pytree

   # It runs inside a sandbox, so it only sees itself!
   # Let's give it access to our home directory to test it:
   $ flatpak run --filesystem=home pt.ua.deti.iei.pytree ~/

   $ flatpak uninstall --user pt.ua.deti.iei.pytree
   ```

   `cd ..` to exit.

2.C: Package `pytree` as an AppImage

1. Create a project directory:

   ```
   $ mkdir ex2-appimage && cd ex2-appimage
   ```

2. Create the `AppDir`:

   ```
   $ mkdir -p Pytree.AppDir
   $ cd Pytree.AppDir
   ```

3. **Download and extract portable Python:** Change the python URL if you use another architecture (such as arm or amr64).

   ```
   $ wget "https://github.com/niess/python-appimage/releases/\
   download/python3.10/python3.10.19-cp310-cp310-manylinux_2_28_x86_64.AppImage" \
   -O python.AppImage
   $ chmod +x python.AppImage
   $ ./python.AppImage --appimage-extract
   $ mv squashfs-root/* .
   $ rm -rf python.AppImage python* squashfs-root/
   ```

4. **Copy your script:**

   ```
   $ cp ../../ex2-pytree/pytree.py usr/bin/
   ```

5. **Update the `AppRun` entrypoint:** Update the `AppRun` file, only the last line requires updates. This script will execute your `pytree.py` using the *bundled* Python.

   ```
   #! /bin/bash
   # If running from an extracted image, then export ARGV0 and APPDIR
   if [ -z "${APPIMAGE}" ]; then
       export ARGV0="$0"

       self=$(readlink -f -- "$0") # Protect spaces (issue 55)
       here="${self%/*}"
       tmp="${here%/*}"
       export APPDIR="${tmp%/*}"
   fi

   # Resolve the calling command (preserving symbolic links).
   export APPIMAGE_COMMAND=$(command -v -- "$ARGV0")

   # Export TCl/Tk
   export TCL_LIBRARY="${APPDIR}/usr/share/tcltk/tcl8.6"
   export TK_LIBRARY="${APPDIR}/usr/share/tcltk/tk8.6"
   export TKPATH="${TK_LIBRARY}"

   # Export SSL certificate
   export SSL_CERT_FILE="${APPDIR}/opt/_internal/certs.pem"

   # Call Python
   "$APPDIR/opt/python3.10/bin/python3.10" "$APPDIR/usr/bin/pytree.py" "$@"
   ```

   Make it executable: `chmod +x AppRun`

6. Create a file named `PyTree.AppDir/pytree.desktop` and fill it. Create also and empty `icon.png` file: `touch icon.png`.

   ```
   [Desktop Entry]
   Name=PyTree
   Exec=AppRun
   Icon=icon
   Type=Application
   Categories=Utility;
   ```

7. **Build, Run, and Cleanup:**

   ```
   $ cd ..  # Go back to ex2-appimage directory
   $ ARCH=x86_64 appimagetool Pytree.AppDir

   $ chmod +x Pytree-x86_64.AppImage
   $ ./Pytree-x86_64.AppImage

   # Test it on your home directory
   $ ./Pytree-x86_64.AppImage ~/

   $ rm -rf Pytree-x86_64.AppImage
   ```

   `cd ..` *to exit.*

---

3. Python GUI App: Tic-Tac-Toe 🎮

3.A: Run with Virtual Environment (venv)

This step simulates what a user would do: download the source, extract it, and run it.

1. Create a directory and download the source:

   ```
   $ mkdir ex3-tictactoe && cd ex3-tictactoe

   $ wget "https://github.com/mariolpantunes/tictactoe/archive/refs/tags/tictactoe-1.0.tar.gz"\
   -O tictactoe-v1.0.tar.gz

   # Extract the downloaded source
   $ tar --strip-components=1 -zxvf tictactoe-v1.0.tar.gz
   ```

2. **Create and activate the `venv`:**

   ```
   $ python3 -m venv .venv
   $ source venv/bin/activate
   ```

3. **Install dependencies from the file:**

   ```
   $ pip install -r requirements.txt
   ```

4. **Run the game:**

   ```
   $ python main.py
   ```

5. **Deactivate:**

   ```
   $ deactivate
   ```

   `cd ..` to exit.

3.B: Package Tic-Tac-Toe as a Flatpak

1. Create a new directory for this build:

   ```
   $ mkdir ex3-flatpak && cd ex3-flatpak
   ```

2. **Create the manifest `pt.ua.deti.iei.tictactoe.yml`:**

```
app-id: pt.ua.deti.iei.tictactoe
runtime: org.gnome.Platform
runtime-version: "48"
sdk: org.gnome.Sdk
command: game
finish-args:
  - --share=ipc
  - --socket=x11
  - --socket=wayland
  - --device=dri
  - --env=PYTHONPATH=/app/lib/game
modules:
  - name: python-deps
    buildsystem: simple
    build-options:
      env:
        MAKEFLAGS: -j$(nproc)
    build-commands:
      - pip3 install --isolated --no-index --find-links="file://${PWD}"
        --prefix=/app pygame
    sources:
      - type: file
        url: https://pypi.io/packages/source/p/pygame/pygame-2.6.1.tar.gz
        sha256: 56fb02ead529cee00d415c3e007f75e0780c655909aaa8e8bf616ee09c9feb1f
  - name: game
    buildsystem: simple
    build-commands:
      - install -d /app/lib/game/
      - install -Dm644 minMaxAgent.py /app/lib/game/minMaxAgent.py
```

```
    - install -d /app/share/game/
    - cp -r assets /app/share/game/
    - install -Dm755 main.py /app/bin/game
    - install -Dm644 pt.ua.deti.iei.tictactoe.desktop
      /app/share/applications/pt.ua.deti.iei.tictactoe.desktop
    - install -Dm644 assets/icon.png
      /app/share/icons/hicolor/128x128/apps/pt.ua.deti.iei.tictactoe.png
  sources:
    - type: archive
      url: https://github.com/mariolpantunes/tictactoe/archive/refs/tags/tictactoe-1.0.zip
      sha256: 4210c04451ae8520770b0a7ab61e8b72f0ca46fbf2d65504d7d98646fda79b5a
```

4. **Build and Install:**

```
$ flatpak-builder --user --install --install-deps-from=flathub \
--force-clean build-dir pt.ua.deti.iei.tictactoe.yml
```

5. **Run and Cleanup:**

```
$ flatpak pt.ua.deti.iei.tictactoe
$ flatpak uninstall --user pt.ua.deti.iei.tictactoe
```

`cd ..` to exit.

3.C: Package Tic-Tac-Toe as an AppImage

1. Create a build directory:

```
$ mkdir ex3-appimage && cd ex3-appimage
```

2. **Download the game source:**

```
$ wget "https://github.com/mariolpantunes/tictactoe/archive/refs/tags/tictactoe-1.0.tar.gz"\
-O tictactoe-v1.0.tar.gz
```

3. Create the `AppDir`:

```
$ mkdir -p TTT.AppDir
$ cd TTT.AppDir
```

4. **Download and extract portable Python:**

```
$ wget "https://github.com/niess/python-appimage/releases/\
download/python3.10/python3.10.19-cp310-cp310-manylinux_2_28_x86_64.AppImage" \
-O python.AppImage
$ chmod +x python.AppImage
$ ./python.AppImage --appimage-extract
$ mv squashfs-root/* .
$ rm -rf python.AppImage python* squashfs-root/
```

5. **Extract your game source:**

```
$ tar --strip-components=1 -zxvf ../tictactoe-v1.0.tar.gz
```

6. **Install dependencies from `requirements.txt`:**

```
./usr/bin/python3.10 -m pip install -r \
./requirements.txt --target ./usr/lib/python3.10/site-packages/
```

7. **Copy your game files:**

```
cp main.py usr/bin/
cp minMaxAgent.py usr/bin/
cp -r assets usr/bin/
```

8. **Update the `AppRun` entrypoint:**

```
#! /bin/bash
# If running from an extracted image, then export ARGV0 and APPDIR
if [ -z "${APPIMAGE}" ]; then
    export ARGV0="$0"

    self=$(readlink -f -- "$0") # Protect spaces (issue 55)
    here="${self%/*}"
    tmp="${here%/*}"
    export APPDIR="${tmp%/*}"
fi

# Resolve the calling command (preserving symbolic links).
export APPIMAGE_COMMAND=$(command -v -- "$ARGV0")

# Export TCl/Tk
export TCL_LIBRARY="${APPDIR}/usr/share/tcltk/tcl8.6"
export TK_LIBRARY="${APPDIR}/usr/share/tcltk/tk8.6"
export TKPATH="${TK_LIBRARY}"

# Export SSL certificate
export SSL_CERT_FILE="${APPDIR}/opt/_internal/certs.pem"

# Export PyGame
export PYTHONPATH="$APPDIR/usr/lib/python3.10/site-packages:$APPDIR/usr/bin"

# Call Python
"$APPDIR/opt/python3.10/bin/python3.10" "$APPDIR/usr/bin/main.py" "$@"
```

Make it executable: `chmod +x AppRun`

9. **Add metadata:**

```
$ mv pt.ua.deti.iei.tictactoe.desktop ./tictactoe.desktop
$ cp usr/bin/assets/icon.png ./pt.ua.deti.iei.tictactoe.png
```

10. **Build, Run, and Cleanup:**

```
cd .. # Go back to ex3-appimage directory
appimagetool TTT.AppDir

chmod +x *.AppImage
./*.AppImage

rm -rf TTT.AppDir *.AppImage tictactoe-v1.0.tar.xz
```