Creating a Debian Package

Outcomes

1.1 Access a shell prompt and issue commands with correct syntax
1.4 Create and edit text files
1.5 Create, delete, copy, and move files and directories
1.9 Utilize a package management system
1.10 Create a package
2.1 Configure container engines, create, and manage containers
2.2 Create a container image
2.3 Build a container image

Background

In this exercise we will be:

1. Installing GAM on an Ubuntu container via their installation script
2. Creating a Debian package for GAM
3. Creating a Dockerfile for a container that installs GAM from your package

Running the Ubuntu Container

`git pull` the class repo to get the most up-to-date version. You may need to `git stash` your changes if you have made any.

`cd` into `exercises/create-deb` and build the container, tagging it as `create-deb`. Then interactively run `bash` on the container, bind mounting `output` on the host to `/output` on the container:

```
$ docker build -t create-deb . ❶
STEP 1/3: FROM docker.io/library/ubuntu
Trying to pull docker.io/library/ubuntu:latest...
Getting image source signatures
Copying blob a1a21c96bc16 done   |
Copying config ce8f79aecc done   |
Writing manifest to image destination
STEP 2/3: RUN apt-get update
Get:1 http://archive.ubuntu.com/ubuntu noble InRelease [256 kB]
... ❷
--> 0bb629520f75
STEP 3/3: RUN apt-get install -y curl python3 xz-utils
Reading package lists...
Building dependency tree...
Reading state information...
...
COMMIT create-deb
--> 276150cb90ad
Successfully tagged localhost/create-deb:latest
276150cb90ad62993ad19a61d59bf2accd331b41bf2deee0029acbbcbc1d4f59
$ docker run -it -v "$(pwd)/output:/output" create-deb bash ❸
root@5c57be59d30b:/# ❹
```

❶ This first command builds the image from the Dockerfile and tags it as `create-deb`. Your output may look slightly different, this was built on a Linux machine using podman with `alias docker=podman`.

❷ When you see `...`, it means I've taken out some of the output to save space.

❸ This second command runs bash on our `create-deb` image *with* the output directory on our host linked to `/output` in the container.

❹ Notice how the prompt changed? We are now executing commands *inside* our container.

The second command make look a little bit strange at first. `$(pwd)/output:/output` should give us an absolute path to the host's output directory in Linux, Windows, and MacOS. If the paths specified in a bind mount are *not* absolute the mount may fail silently!

Installing GAM

Read the GAM README.md in their GitHub repository and follow the directions to install GAM from the BASH prompt on the running container. The script will ask you if you have a full browser. You can type in 'N', but it really shouldn't matter. The script will also ask if you want to set up a Google API project for GAM. You can type in 'N' because we will not actually be linking this to a real Google admin account.

Read the output from the installation script *carefully*. You will need to know where you can find the `gam` binary.

Creating a Debian Package

A Debian package is built from a directory single with a very specific structure. We'll call this the *deb package directory*. The *deb package directory* should be named whatever you want to call your package. Within that

directory there is a `DEBIAN` directory with a control file. The files your package will install are the *deb package directory* in the locations where they will be installed on a system.

This is probably easier seen than described:

```
package-name/ ❶
package-name/DEBIAN ❷
package-name/DEBIAN/control ❸
package-name/usr/bin/package-binary ❹
package-name/etc/package-config.toml ❺
```

❶ The whole thing is in a single *deb package directory*
❷ A special directory named `DEBIAN` contains files used by the Debian package tools to build the package
❸ Within the `DEBIAN` there is a `control` file that gives details about the package
❹ This package installs a binary file called `package-binary` in `/usr/bin/`
❺ This package also installs a configuration file called `package-config.toml` in `/etc/`

Go ahead and make the directory structure for your package, don't forget to give it a name, in the `/output` directory on your container. You'll need to use the `mkdir` and `cd` commands.

Copy (`cp`) the `gam` binary (you paid attention to where it was installed, right?) into your package as well. You'll probably want to install it in `/usr/bin/` as it's a tool that is used by any user on the system.

Finally, you'll need to create a `control` file for your package in the `DEBIAN` directory. You can do this within the container using a console text editor (`apt install nano` or maybe `apt install neovim` if you prefer) or you can do it on the host system using the bind mounted `output` directory.

Here is an example control file that you can adjust to meet the needs of your package:

```
Package: package-name ❶
Version: 1.0
Section: custom
Priority: optional
Architecture: all ❷
Essential: no
Installed-Size: 1024 ❸
Maintainer: your-name-here ❶
Description: In a line, what does this do? ❶
```

❶ you will need to update these
❷ believe it or not `gam` is actually a Python script, so this architecture is fine
❸ technically this is in 1 KiB units, for this exercise you don't have to calculate it

Once you have your *deb package directory* all set, with a control file and the `gam` binary in it you can build it with `dpkg-deb --build package-name` executed from the directory above your *deb package directory* and with `package-name` set to the name of your *deb package directory*.

Once you've run the `dpkg-deb` command you'll end up with the deb file. This is your package! Assuming the package was created in the `/output` directory you will be able to access it from the host machine. Double-check that you can access the file *before* you exit the container.

Creating a Dockerfile for an Image That Uses Your Package

Take a look at the Dockerfile in the main exercise directory: `it610/exercises/create-deb`. Make your own Dockerfile in the `output` directory that copies your `.deb` package into the image and installs it with `dpkg -i`.

Relevant documentation can be found here:

- Dockerfile reference: FROM
- Dockerfile reference: COPY
- Dockerfile reference: RUN
- dpkg man page

Questions

Now that you know how to make a package file and use it when creating custom images, please answer the following questions in the text submission section of this assignment:

1. Are there any security concerns with installing an application like GAM via a shell script? Give examples.
2. What is the GAM installation script written in? What does it do?
3. What are the advantages of having a deb package for GAM as opposed to just using the installation script? What are the disadvantages?
4. What was the hardest part of this exercise?