

Snap Quickstart Workshop

Ubuntu Summit 2024



Past iterations



Contents

- GTK snap walkthrough
- Interactive part
 - Create example step-by-step
 - Or create your own snap

<https://github.com/snapcrafters/snap-quickstart-workshop>



Setup

1. Install snapd (not needed on Ubuntu)

- Install the package: <https://snapcraft.io/docs/installing-snapd>
- And add classic snap support:

```
sudo ln -s /var/lib/snapd/snap /snap
```

2. Install Snapcraft

- ```
sudo snap install snapcraft --classic
```

## 3. Install LXD

- ```
sudo snap install lxd
```
- ```
sudo adduser `whoami` lxd
```





# GTK Hello World

Example 1



# Building without snap

- Create the file `~/exercises/hello-world-gtk/src/hello-world-gtk.c`
- Add the source code from <https://www.gtk.org/docs/getting-started/hello-world>
- Build the app

```
$ cd ~/exercises/hello-world-gtk/src
```

```
$ sudo apt install libgtk-4-dev
```

```
$ gcc $(pkg-config --cflags gtk4) -o hello-world-gtk \
 hello-world-gtk.c $(pkg-config --libs gtk4)
```

- Run the app

```
$./hello-world-gtk
```

- Remove the app

```
$ rm ./hello-world-gtk
```



# Creating the snap - snapcraft.yaml

File explaining how to build the app itself and the snap

- **metadata:** Name, version, summary, description, ...
- **apps:**
  - How to execute your app
  - What permissions to give your app
- **parts:**
  - How to build the binaries
  - What libraries your app needs

```
1 name: kcalc
2 version: 19.08.0
3 grade: stable
4 adopt-info: kcalc
5
6 confinement: strict
7 base: core18
8
9 apps:
10 kcalc:
11 common-id: org.kde.kcalc.desktop
12 command: kcalc
13 extensions:
14 - kde-neon
15 plugs:
16 - home
17 - opengl
18 - network
19 - network-bind
```



# First: how to make a snap

kcalc/snapcraft.yaml

*\$ snapcraft*

kcalc.snap

*\$ sudo snap install kcalc.snap --dangerous*

/snap/kcalc

```
1 name: kcalc
2 version: 19.08.0
3 grade: stable
4 adopt-info: kcalc
5
6 confinement: strict
7 base: core18
8
9 apps:
10 kcalc:
11 common-id: org.kde.kcalc.desktop
12 command: kcalc
13 extensions:
14 - kde-neon
15 plugs:
16 - home
17 - opengl
18 - network
19 - network-bind
```





# gtk-hello-world - snapcraft.yaml

Complete example: <https://github.com/snapcrafters/snap-quickstart-workshop>

## ✓ EXERCISES

### ✓ hello-world-gtk

#### ✓ src

**C** hello-world-gtk.c

**!** snapcraft.yaml



# Branding metadata

More info: <https://snapcraft.io/docs/snapcraft-yaml-schema>

Visible in the Snap Store

- **Name** must be lowercase letters, numbers and hyphen
- **Version** is a string
- **Summary** must be one line
- **Description** can be multiline

```
1 name: hello-world-gtk
2 version: '0.1'
3 summary: Gtk Hello World example
4 description: A simple Gtk example
```



# Technical metadata

More info: <https://snapcraft.io/docs/snapcraft-yaml-schema>

Describes the snap's technicalities

- **base** defines the Ubuntu version to use inside the snap
  - "core24": Ubuntu 24.04
  - "core22": Ubuntu 22.04
- **confinement** defines sandbox
  - "devmode" gives all access but logs
  - "strict" for regular sandbox
  - "classic" for no sandbox
- **compression** of the snap package
  - Always use "lzo" for best startup time!

5

```
base: core24
```

6

```
confinement: strict
```



# Apps

More info: <https://snapcraft.io/docs/snapcraft-yaml-schema#app-directives>

How to execute your apps & what permissions to give

- **extensions** add common functionality
  - ◆ *gnome* adds GUI & GTK support
  - ◆ *kde* adds GUI & Qt support
- **command** is the path to the binary, relative from snap root
- **plugs** describes the permissions to give your app
  - ◆ Supported permissions: <https://snapcraft.io/docs/supported-interfaces>

```
9 apps:
10 | hello-world-gtk:
11 | extensions: [gnome]
12 | command: src/hello-world-gtk
13 | plugs:
14 | | - removable-media
```



# Slots

More info: <https://snapcraft.io/docs/dbus-interface>

Desktop apps need access to session dbus for common functionality. This needs a declaration in slots.

- **interface:** dbus
- **name:** unique id of the app
- **bus:** session

```
16 slots:
17 session-dbus-interface:
18 interface: dbus
19 name: org.gtk.example
20 bus: session
```



# Parts

## Describes

- How to compile your app
- What dependencies it needs.

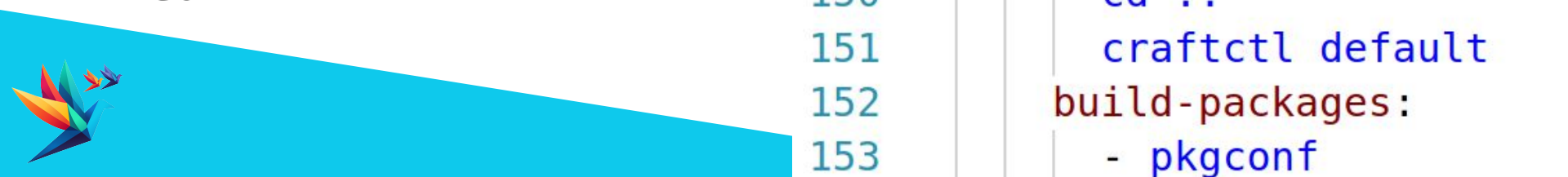
One part for each component that needs to be built separately

- C++
- Python
- Go

More info: <https://snapcraft.io/docs/adding-parts>

Deep info: <https://snapcraft.io/docs/parts-lifecycle>

```
142 parts:
143 hello-world-gtk:
144 plugin: dump
145 source: .
146 override-build: |
147 set -eux
148 cd src
149 gcc $(pkg-config --cflags --libs gtk+-3.0) *.c
150 cd ..
151 craftctl default
152 build-packages:
153 - pkgconf
```



# Plugin

More info: <https://snapcraft.io/docs/supported-plugins>

Which build system to use

- Binary packages or installers
  - dump: copy files (or DIY)
- source code
  - python
  - cmake
  - meson
- DIY with a scripts
  - dump or nil

```
142 parts:
143 hello-world-gtk:
144 plugin: dump
145 source: .
146 override-build: |
147 set -eux
148 cd src
149 gcc $(pkg-config --cflags
150 (pkg-config --libs
151 cd ..
152 craftctl default
153 build-packages:
154 - pkgconf
```



# Source

More info: <https://snapcraft.io/docs/supported-plugins>

Where to get source code or binaries.

- Folder in repo with snapcraft.yaml
- External repo
- Remote file

```
142 parts:
143 hello-world-gtk:
144 plugin: dump
145 source: .
146 override-build: |
147 set -eux
148 cd src
149 gcc $(pkg-config --cflags
150 (pkg-config --libs
151 cd ..
152 craftctl default
153 build-packages:
154 - pkgconf
```





# Override-build

More info: <https://snapcraft.io/docs/overrides>

Change default build logic of plugin with bash script

- call compilers directly
- prepare environment
- run default logic
  - craftctl default

```
142 parts:
143 hello-world-gtk:
144 plugin: dump
145 source: .
146 override-build: |
147 set -eux
148 cd src
149 gcc $(pkg-config --cflags gtk+-3.0)
150 (pkg-config --libs gtk+-3.0)
151 cd ..
152 craftctl default
153 build-packages:
154 - pkgconf
```



# Build-packages

More info: <https://snapcraft.io/docs/overrides>

Tools needed to build the snap

- Compilers
- SDK libraries
- Build tools

Will not be present in the final snap

```
142 parts:
143 hello-world-gtk:
144 plugin: dump
145 source: .
146 override-build: |
147 set -eux
148 cd src
149 gcc $(pkg-config --cflags
150 (pkg-config --libs
151 cd ..
152 craftctl default
153 build-packages:
154 - pkgconf
```



# Stage-packages

More info: <https://snapcraft.io/docs/overrides>

Dependencies needed to run the app.

(and that are not part of an extension)

Will be in the final snap

```
34
```

```
35
```

```
stage-packages:
- curl
```



# Thanks! Questions?

Now it's your turn!

- Either follow the tutorial:  
<https://github.com/snapcrafters/snap-quickstart-workshop>
- Or try snapping your own app and we'll help you out!
- Or try snapping these snaps may be?  
[Marktext](#) or [Marknote](#)



# Next steps

- AppStream via adopt-info
- Specify architectures -> see CI workshop
- Channels and tracks -> see CI workshop
- Building dependencies from source



# AppStream as Metadata via adopt-info

Appstream Metadata => Snap  
Metadata

Parse the metadata from a part

Let snapcraft know that “this”  
part will be used as the source  
of metadata

```
1 name: newsflash
2 base: core24
3 version: '3.3.5'
4 adopt-info: newsflash
5 compression: lzo
6 issues: https://github.com/soumyaDghosh/newsflash-snap/issues
7 grade: stable
8 confinement: strict
9 platforms:
10 amd64:
11 arm64:
12 armhf:
```

```
108 organize:
109 snap/newsflash/current: .
110 parse-info: [usr/share/metainfo/io.gitlab.news_flash.NewsFlash.appdata.xml]
111
```



# Architectures...

Snaps can be built on arches that is supported by Ubuntu and the list is huge. Check [here](#) to know the list.

In the snap manifest, you should explicitly mention the architecture... But, why? To know that, join the [workshop tomorrow](#) on the CI that we use to maintain and publish our snaps by us at 2PM....



# Build your dependencies from source

Build your deps in different parts!

- Benefits
  - Latest updates and releases
  - Support for custom patches
  - Everything on your control
- Disadvantages
  - Complexity
  - Miss the CVE checks done by Ubuntu for its archive packages





# Things to help you in the process

- Use Gnome/KDE extensions if the libraries are related to this
- Use [ffmpeg](#) or [webkitgtk](#) shared library snaps, if your app needs them
- Try keeping the files only that your app needs
- Try to keep your snap populated with all the possible metadata you can add

