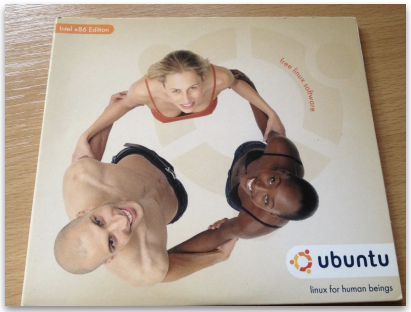
Packaging Swift for Ubuntu and more

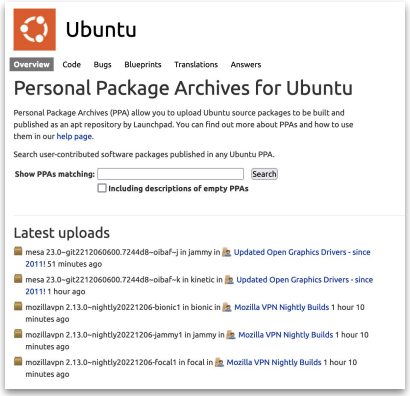
Matias Piipari
https://fosstodon.org/@mz2
Director of Engineering

CANONICAL ubuntu

Brief history of Canonical's Linux packaging story

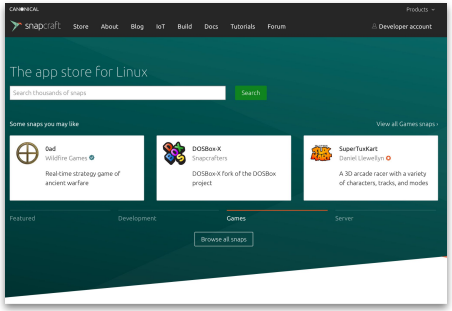


main
universe



main
universe

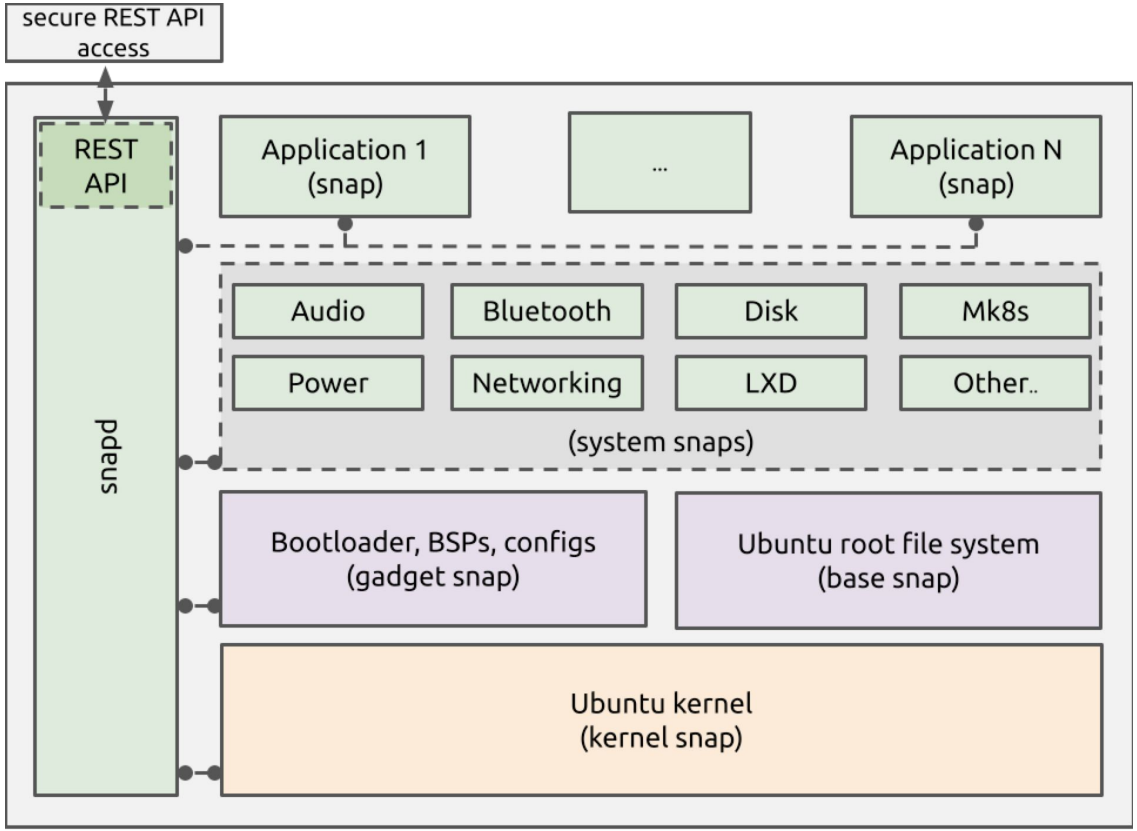
Personal Package Archives (PPAs)



main
universe
PPAs

Snap Store (snapcraft.io)

Ubuntu Core


secure REST API access
REST API
snapd
Application 1 (snap)
...
Application N (snap)
Audio
Bluetooth
Disk
Mk8s
Power
Networking
LXD
Other..
(system snaps)
Bootloader, BSPs, configs (gadget snap)
Ubuntu root file system (base snap)
Ubuntu kernel (kernel snap)

We recently helped package .NET for Jammy

https://devblogs.microsoft.com/dotnet/dotnet-6-is-now-in-ubuntu-2204/

.NET 6 is now in Ubuntu 22.04

Richard Lander

August 16th, 2022 | 45 | 20

.NET 6 is now included in Ubuntu 22.04 (Jammy) and can be installed with just `apt install dotnet6`. This change is a major improvement and simplification for Ubuntu users. We're also releasing .NET with Chiseled Ubuntu Containers, a new small and secure container offering from Canonical. These improvements are the result of a new partnership between Canonical and Microsoft.

Chiselled Ubuntu container = almost Alpine tiny

https://devblogs.microsoft.com/dotnet/dotnet-6-is-now-in-ubuntu-2204/

First, the `runtime-deps` layer.

- Ubuntu 22.04 (Jammy): `112MB`
- Chiseled Ubuntu 22.04 (Jammy): `12.9MB`

And on the other end of the spectrum, the `aspnet` layer.

- Ubuntu 22.04 (Jammy): `213MB`
- Chiseled Ubuntu 22.04 (Jammy): `104MB`

It's reasonable to ask what Alpine looks like. It's a newer distro designed to be super small and componentized from the start. Alpine is `9.84MB` for `runtime-deps:6.0-alpine` and `100MB` for `aspnet:6.0-alpine`.

Swift.org distributes a tarball (😢) for Ubuntu.

Current status of Swift on Ubuntu

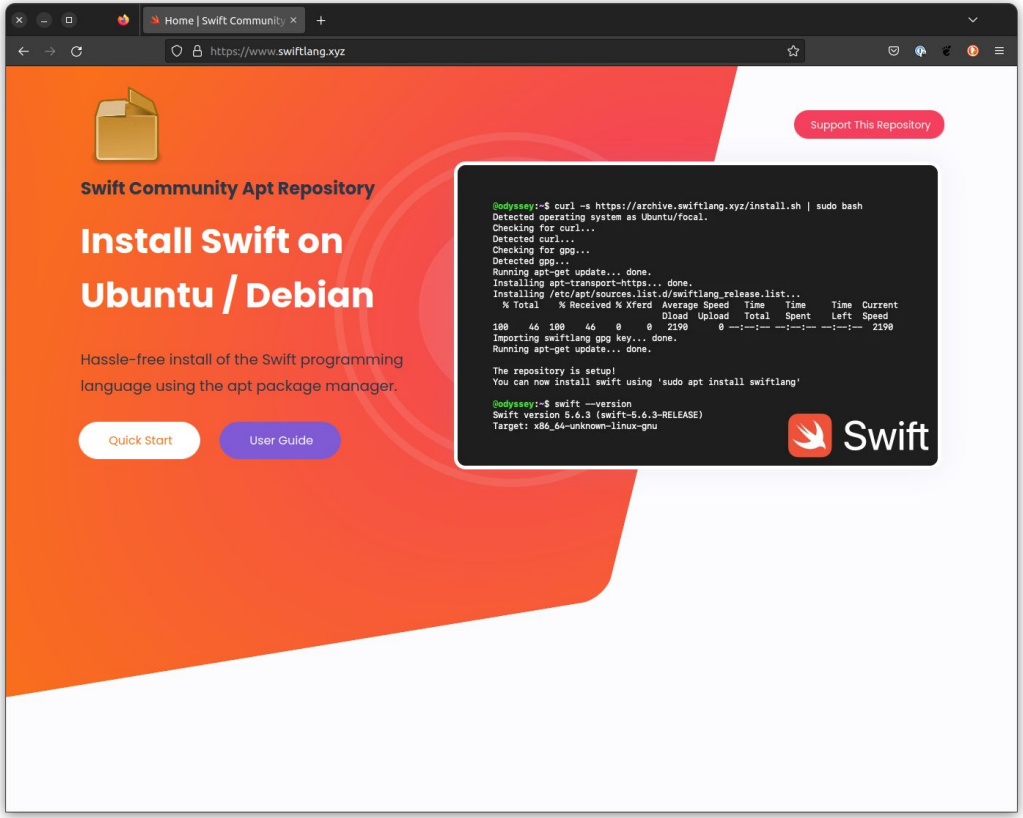
Swift.org distributed Swift on Ubuntu is great… as long as you are after a Docker image.
If you rely on the tarball though:

- Friction: fetch dependencies & untarball, set up `$PATH`.
- No upgrade path, bug / security fixes, livepatch from Canonical.
- Setting up path dependent tools can get awkward.

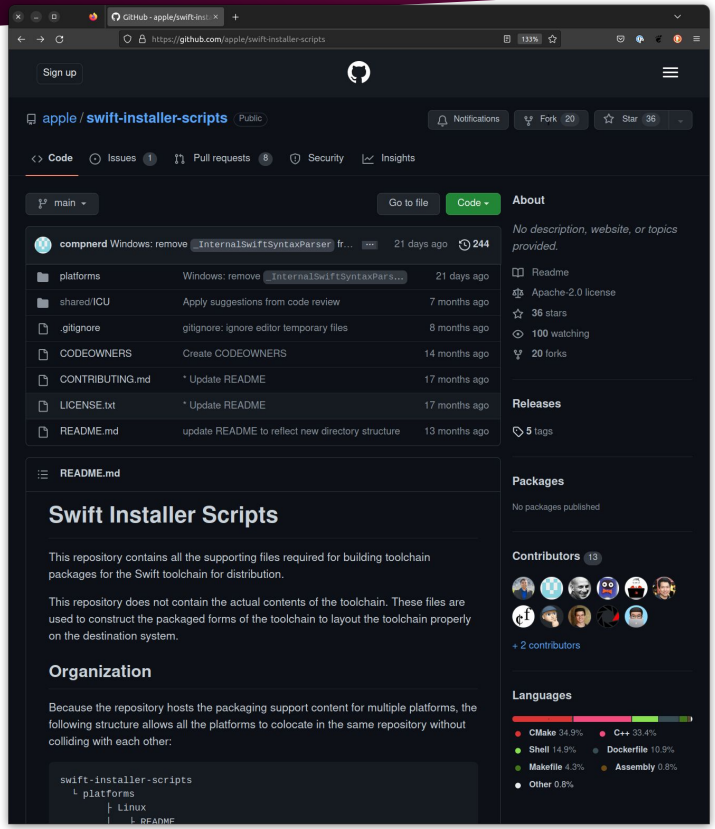
Practical implication of the friction: developer experience is not attractive.
(compare to, say, Rust, Golang).

- Docker great for deploying cloud native, but not ideal for all kinds of local tools.
- Packaging with SPM / mint or homebrew similarly a high friction route.

Swiftlang.xyz by @futurejones



github.com/apple/swift-installer-scripts



github.com/apple/swift-installer-scripts

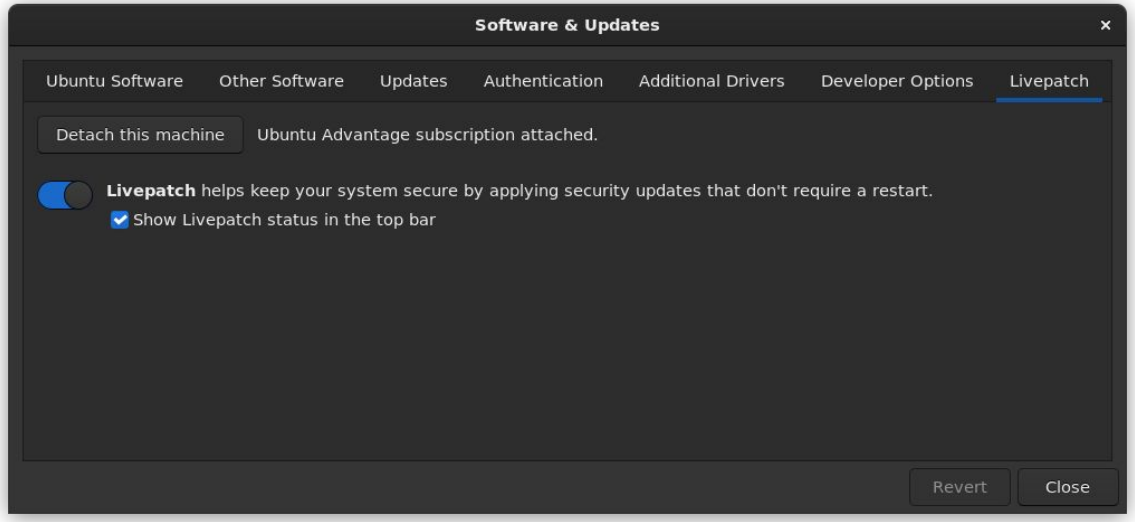
Linux Packages (RPM/Deb)

Currently Swift on Linux is distributed via tarball and Docker, and we would like to start supporting RPM and Debs officially on swift.org. The goal is to provide a seamless install process for Swift on Linux by utilizing the platform's native package manager (RPM/Deb).

- Step 1. Develop native packages / installers for the distributions
- Step 2. Offer the native packages / installers through swift.org
  - Support all officially supported Linux platforms
  - Code signed by swift.org certificate
  - Repository hosted on swift.org
- Step 3. Offer the native packages / installer through official repositories for the various platforms
  - Work with official repositories to accept package specs
  - Deprecate swift.org packages / installer repository
- Step 4. Deprecate swift.org Linux tarballs

Count me in!

Swift in `universe` would be Ubuntu Pro supported (no need to get it to `main`)



23,000 more packages secured

Expanded coverage for over 10x more open-source packages, now including the Ubuntu Universe repository (in beta)

Packaging Swift based software in `debs` still not the way to go for many even if `swiftlang` were in universe.

Packaging your app to N distro specific formats is no-one's idea of fun
(which is why dev and distro packager traditionally have been different people):

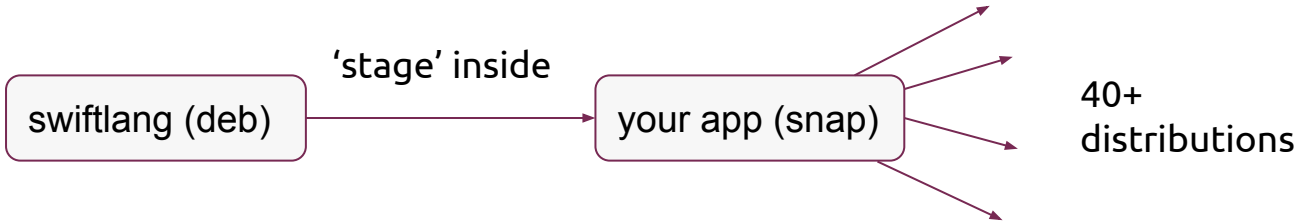
- kernel, glibc, other key shared library dependency versions and fs paths.
- bad packages (or failed installs of good packages) can break a user system.

Your attention is naturally divided based on addressable audience.

Snap and Flatpak formats both tackle these problems (different tech, scope).

- Flatpak: desktop applications.
- Snap:
  - desktop, CLI utilities, servers (whole distributed systems), embedded applications.
  - systemd units (e.g. network-manager, snapd itself).
  - whole system configuration ("gadget snaps"), kernel updates ("kernel snaps").

Ubuntu Debs can be staged inside universal Snaps



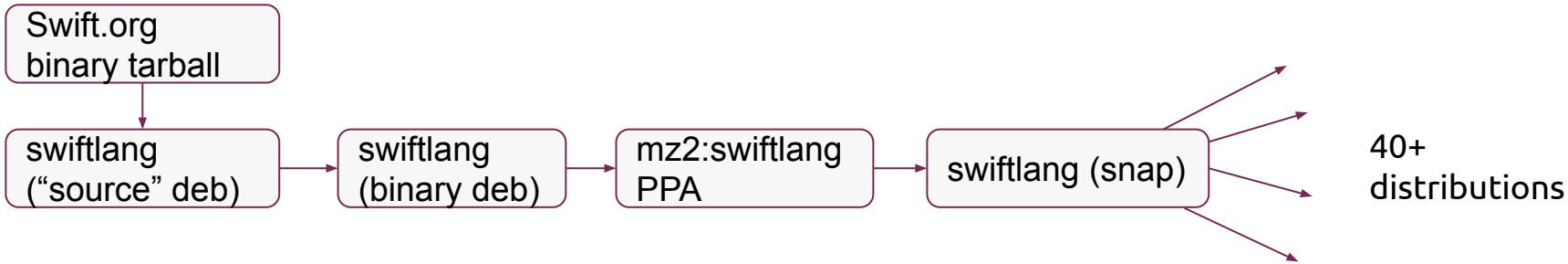
```
mz2 in athena in swiftlang-snap/plugin/vapor on ?main [!?]
❯ vapor.toolbox new foo                                                    (base)
Cloning template...
name: foo
Would you like to use Fluent? (--fluent/--no-fluent)
[y/n> y
fluent: Yes
db: Postgres (Recommended)
Would you like to use Leaf? (--leaf/--no-leaf)
[y/n> y
leaf: Yes
Generating project files
[+ Package.swift
+ main.swift
+ configure.swift
+ routes.swift
+ Todo.swift
+ CreateTodo.swift
+ .gitkeep
+ TodoController.swift
+ AppTests.swift
+ index.leaf
+ .gitkeep
+ Dockerfile
+ docker-compose.yml
+ .gitignore
+ .dockerignore
Creating git repository
Adding first commit
```

```
 _    __   ___  ___  ___
\ \  / /\  | |_) / / \ | |_)
 \_\/ /--\ |_|   \_\_/ |_| \
a web framework for Swift

Project foo has been created!

Use cd 'foo' to enter the project directory
Then open your project, for example if using Xcode type open Package.swift or code . if using VSCode

mz2 in athena in swiftlang-snap/plugin/vapor on ?main [!?] took 8s
❯                                                                          (base)
```

Status of my Swift packaging experiments

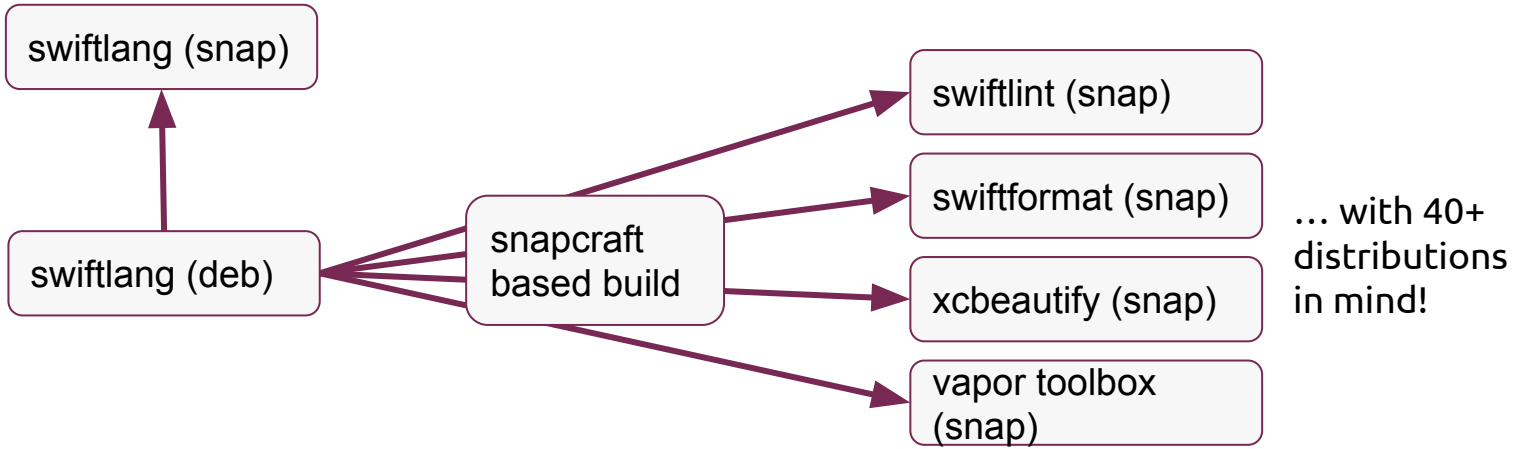
```
apt add-repository ppa:mz2/swiftlang
apt update
apt install swiftlang
```

Status of my Swift packaging experiments

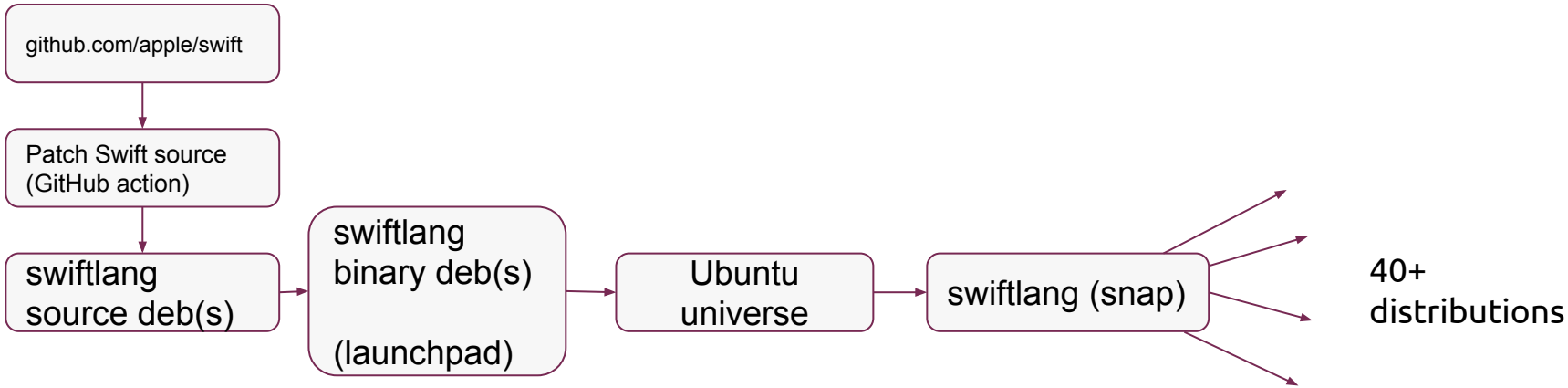



```
snap install ./swiftlang.snap
```

Status of my Swift packaging experiments

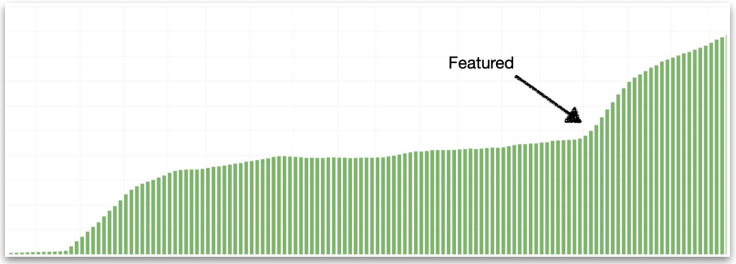


What should be happening instead



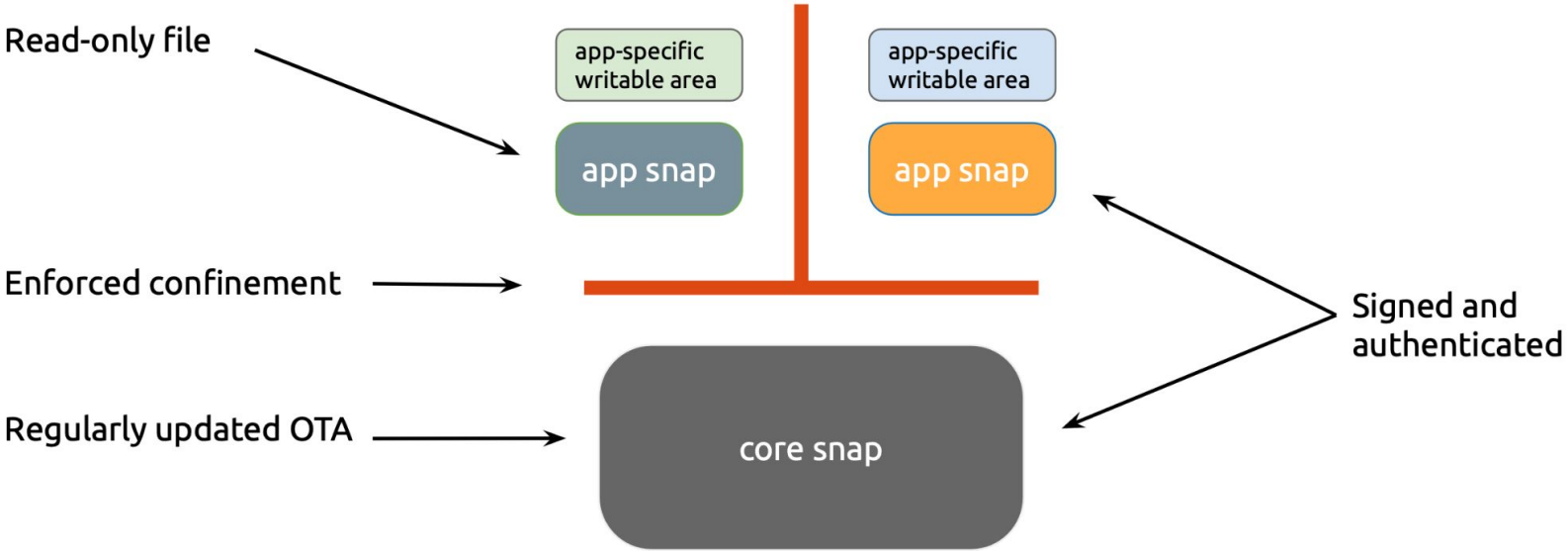

```
snap install ./swiftlang.snap
```

Linux is easy to target via Snap Store

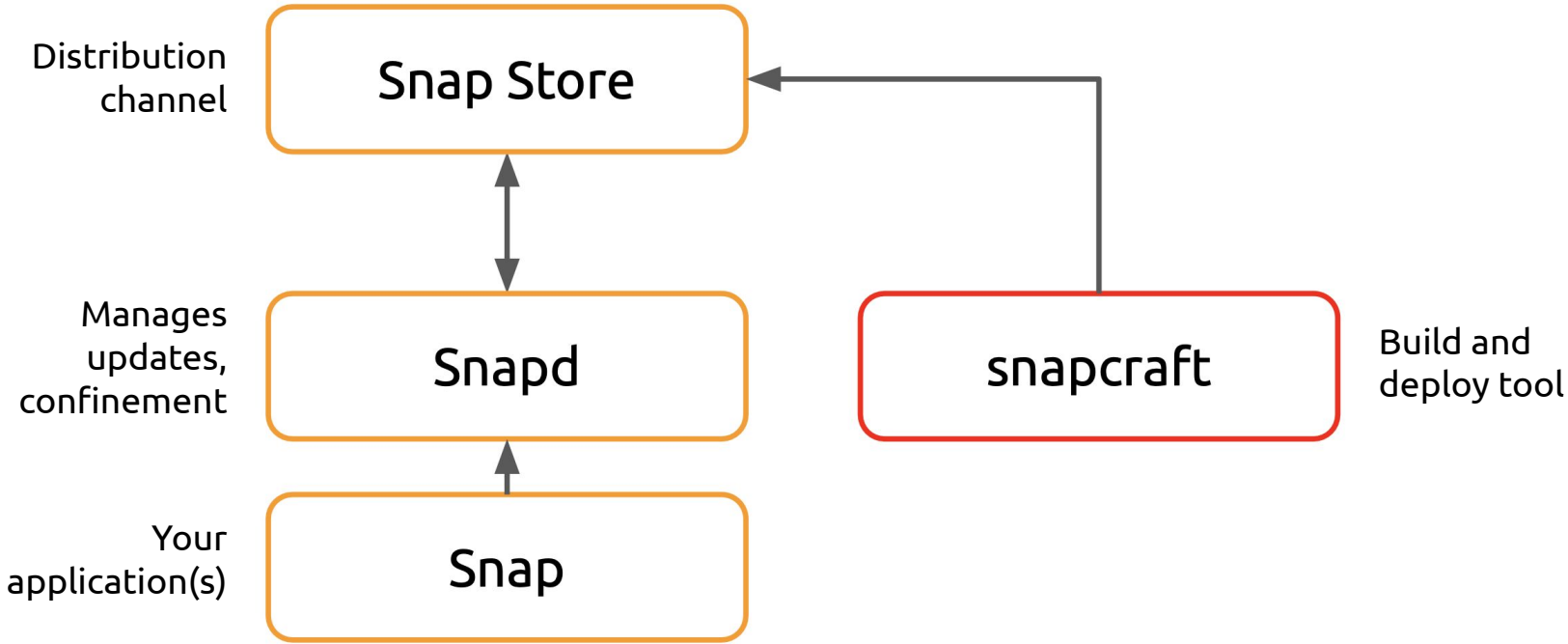


- Target anything systemd + snapd enabled (40+ distributions) across amd64, arm64, arm8f, risc-v, …
- Can be a good way to get discovered (getting featured).
- Can’t brick someone’s system with your software: immutable, transactional, rollbacks, sandboxed.
- Install metrics per version & geography available.
- Controlled risk grades for updates with “channels” (stable / beta / …) – automated updates ***can*** be stopped.
- Easy to integrate to pre-existing CI/CD pipelines.
- Free remote build environment for all the micro-architectures (no cross-compiling flags to invent).

Architecture of a Snap



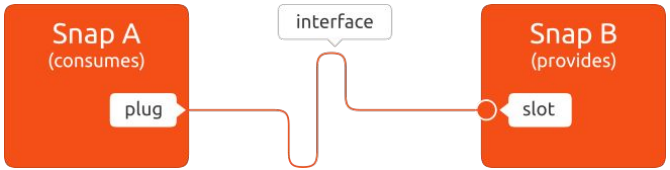
Architecture of a Snap



Interfaces ~= “entitlements” in macOS sandbox terms

chromium
discord
firefox
…

snapd




```
❯ snap interface audio-record
name:audio-record
summary: allows audio recording via supporting services
plugs:
  - chromium
  - discord
  - firefox
  - mattermost-desktop
  - rocketchat-desktop
  - signal-desktop
  - slack
  - steam
  - telegram-desktop
  - vlc
slots:
  - snapd
```

Interfaces ~= “entitlements” in macOS sandbox terms

- Example interfaces:
  - **content** interface
    - access to mount locations from providers inside the consumer mount namespace
    - = way to express a dependency between packages that snapd will resolve
  - **serial-port** interface
    - Allow access to specific device nodes (/dev/ttyS*, /dev/ttyUSB*, /dev/ttyMX*..)
    - Allow access to related char devices (c 4:*, …)
  - **process control** interface
    - Allow access to /usr/bin/{p}kill
    - Allow nice, setpriority, sched_set* syscalls

Interfaces are an abstraction to snapd managed, Linux kernel provided security systems.

- Interfaces are very granular, regularly added to and adjusted.
- Debugging mechanisms are provided for debugging denials.
- Implementation of individual interfaces are generally compact and readable.

Confinement (“sandboxing”)

Logical separation: a Snap is an immutable, self-contained, read-only mounted filesystem.

Further isolation mechanisms rely on Linux kernel capabilities:

- **Namespaces**: processes in a snap see their own view of the system (user, file, processes, network interface, hostname, …).
- **seccomp**: system call filters (BPF) for the package’s processes.
- **cgroups**: limit access to system resources (like devices) that can be consumed (quotas): CPU, memory, networking, access to various device categories.
- **AppArmor**: snapd managed Mandatory Access Control profiles (at `/var/lib/snapd/apparmor/profiles`) which can be further extended by sys admin.

From CLI tools to a Kubernetes cluster

- Snaps can house systemd services, IMO *the* simplest way I have found to create one.

- Confinement model is very flexible for many classes of applications. Examples:
  - microk8s: a lightweight Kubernetes runnable on public clouds or edge.
  - juju: cloud agnostic deploy and service lifecycle manager for distributed systems.
  - MAAS: data centre management, provisioning system.
  - steam: Steam together with (different degrees of bleeding) edge version of mesa.

- Hooks provided to respond to configuration changes (and install, refresh time).
  - E.g. configuration changes made with `snap config get / set` (akin to `defaults get / set` on macOS).
- Health checks can be created to confirm package is in workable state -> rolled back to last working snapshot if not.

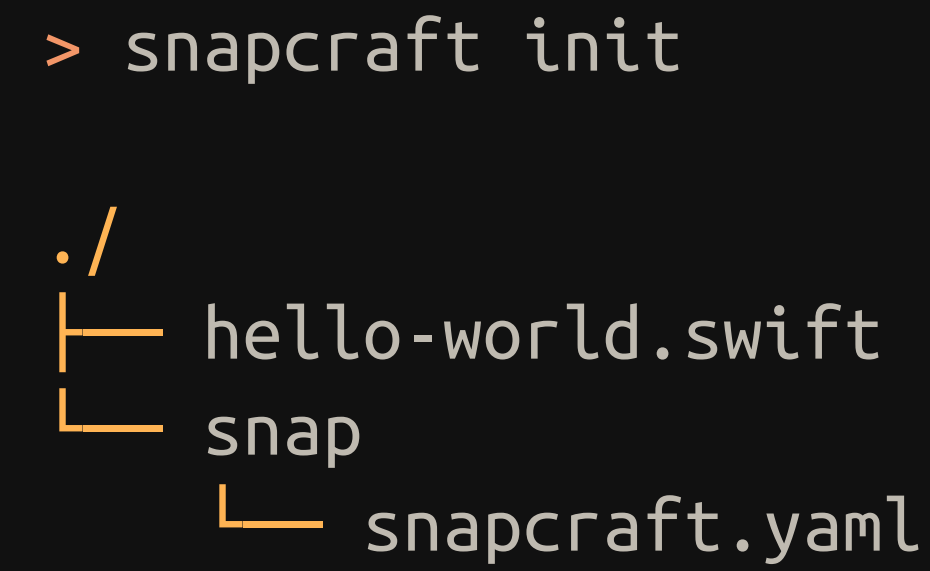
So let’s make one!

snapcraft: build tool for Snap packages

- Reads package metadata, build steps from a `snap/snapcraft.yaml`
- Build done in an isolated environment, either using an LXD system container or multipass (qemu).
- Remote building (on Launchpad.net) also available for amd64, arm64, RISC-V, … : `snapcraft remote-build`
- To distribute app, snapcraft authenticates you and uploads the package to Snap Store (Store handles codesigning – no local dev code signing hell to be expected!)

```
> cat hello-world.swift

print("Hello world")
```

```
> swiftc hello-world

> hello-world
Hello world
```


> snapcraft init
./
├── hello-world.swift
└── snap
    └── snapcraft.yaml

```yaml
name: hello-world
base: core22
version: "0.1"
summary: …
description: …

# 'stable' to release to candidate/stable
channels.
grade: stable

# 'devmode' during dev,
# 'strict' when you have it confined right,
# 'classic' when not confined.
confinement: strict

# express a package repository dependency
# to mz2/swiftlang PPA.
package-repositories:
  - type: apt
    ppa: mz2/swiftlang
```

```
parts:
  hello-world:
    plugin: dump
    source: .
    stage-packages:
      - swiftlang
    override-build: |
      swiftc hello-world.swift
      mkdir -p $SNAPCRAFT_PART_INSTALL/bin
      mv hello-world $SNAPCRAFT_PART_INSTALL/bin/hello-world

apps:
  hello-world:
    command: bin/hello-world
```

```
> ldd /snap/hello-world/current/bin/hello-world
```

```
ldd /snap/hello-world/current/bin/hello-world
    linux-vdso.so.1 (0x00007ffc09a8c000)
    libswift_StringProcessing.so => not found
    libswift_Concurrency.so => not found
    libswiftCore.so => not found
    libswiftSwiftOnoneSupport.so => not found
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
(0x00007f4f68a00000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f4f68d48000)
```

```
parts:
  hello-world:
    plugin: dump
    source: .
    build-packages:
      - swiftlang
      - chrpath
    stage-packages:
      - swiftlang
    override-build: |
      swiftc hello-world.swift
      patchelf --set-rpath $ORIGIN/../lib/swiftlang/lib/swift/linux ./hello-world
      mkdir -p $SNAPCRAFT_PART_INSTALL/bin
      mv hello-world $SNAPCRAFT_PART_INSTALL/bin/hello-world

apps:
  hello-world:
    command: bin/hello-world
```

```
> snapcraft
> sudo snap install hello-world.snap

> hello-world
Hello world
```

So let’s make another one!

SwiftFormat

```
parts:
  swiftformat:
    plugin: dump
    source: https://github.com/nicklockwood/SwiftFormat.git
    source-tag: 0.50.5
    build-packages:
      - patchelf
    stage-packages:
      - swiftlang
    override-build: |
      swift build -c release

      BUILT_BIN=`swift build -c release --show-bin-path`/swiftformat
      patchelf --set-rpath /usr/lib/swiftlang/lib/swift/linux $BUILT_BIN

      mkdir -p $SNAPCRAFT_PART_INSTALL/bin
      install -v $BUILT_BIN $SNAPCRAFT_PART_INSTALL/bin

apps:
  swiftformat:
    command: bin/swiftformat
    plugs:
      - home
```

xcbeautify

```
parts:
  swiftlint:
    plugin: dump
    source: https://github.com/tuist/xcbeautify.git
    source-tag: 0.16.0
    build-packages:
      - swiftlang
      - patchelf
    stage-packages:
      - swiftlang
    override-build: |
      make install
      BUILT_BIN=`swift build $SWIFT_FLAGS --show-bin-path`/xcbeautify
      patchelf --set-rpath '$ORIGIN:$ORIGIN/../usr/lib' $BUILT_BIN
      mkdir -p $SNAPCRAFT_PART_INSTALL/bin
      install -v $BUILT_BIN $SNAPCRAFT_PART_INSTALL/bin

apps:
  xcbeautify:
    command: bin/xcbeautify
    plugs:
      - home
```

Vapor Toolbox

```
apps:
  vapor:
    command: bin/vapor
    command-chain:
      - bin/toolbox-launcher
    plugs:
      - home
      - network
      - network-bind

layout:
  /usr/lib/swiftlang:
    symlink: $SNAP/usr/lib/swiftlang
  /etc/gitconfig:
    bind-file: $SNAP_DATA/etc/gitconfig
  /usr/lib/git-core:
    symlink: $SNAP/usr/lib/git-core
  /usr/share/git-core/templates:
    symlink: $SNAP/usr/share/git-core/templates
```

Service (systemd unit)

Creating a service

https://snapcraft.io/docs/services-and-daemons

```
apps:
  your-vapor-app:
    command: bin/your-vapor-app
    daemon: simple # simple | oneshot | forking | notify

    restart-condition: always #on-failure|on-success|…|never
    stop-mode: sigterm # sigterm|sigterm-all|sighup|…|sigint-all
    # 20+ more options
```

`snapcraft` is pluggable.
How about a Swift plugin?

Snapcraft is pluggable. How about a Swift plugin?

```yaml
parts:
  hello-world:
    plugin: swift
    source: .
    swift-revision: 5.7.1-RELEASE
    swift-product: hello-world

apps:
  hello-world:
    command: bin/hello-world
```

Snapcraft is pluggable. How about a Swift plugin?

```
parts:
  hello-world:
    plugin: swift
    source: .
    swift-revision: 5.7.1-RELEASE
    swift-product: swiftlint
    swift-configuration: release
    swift-flags: -Xswiftc -static-stdlib
    Swift-include-path:
      -.
    swift-linker: lld
    swift-link:
      - CFURLSessionInterface
      - CFXMLInterface
      - curl
      - xml2
    stage-packages:
      - libxml2
      - libcurl4
apps:
  swiftlint:
    command: bin/swiftlint
```

Halp! Can we work together on this?

- Repackaging Swift.org tarballs = a hack to let me explore the value of doing the rest.
  - I got further, but got stuck being able to remove the hack.
  - … because Swift version, OS version, CPU architecture specific toolchain build errors plausibly need more maintainer attention.

- Building “real” debs with Ubuntu blessed compiler flags included would be useful:
  - Symbol stripping doesn’t work quite right for Debian build tool chain.
  - A debug symbol package would be lovely to include as well.
  - Microarchitecture version used is occasionally changed, has performance effect.
  - FS paths inside the package are pretty awkward.

- A single package with a 480M kitchen sink inside it is not ideal.
  - Compiler toolchain, headers, runtime libs all in one package.
  - (A snapcraft plugin can deal with this by staging only specific paths.)
- How about some minimal, rootless Ubuntu Docker images, too?

Thank you. Questions?

Slides over at https://matiaspiipari.dev
https://fosstodon.org/@mz2

(Also, we’re hiring, lots!)