The Road to KDE Neon Core

Gosh! We're surrounded by Snaps everywhere!

Kevin Ottens

HAUTE COUTURE

enioka

whoami

- Started to use KDE with 1.0-beta1 in 1997
- Procrastinated until 2003 to finally contribute code
- Fell in love with the community back then
- Kept doing things here and there... most notably helped with:
  - kdelibs
  - KDE Frameworks architecture
  - the KDE Manifesto
  - Community Data Analytics
- Part of the enioka Haute Couture family
- Living in Toulouse

Introduction

Ubuntu Core

Ubuntu Core is a minimal, secure and strictly confined operating system

- Designed for embedded systems
- Immutable version of Ubuntu
- Secure by design, containerised
- All the components are in snaps, even the kernel and the snap daemon (snapd)
- OTA updates with automated rollback if needed

Ubuntu Core Desktop

A fully containerised desktop, where each component is immutable and isolated

- Announced last year
- Full desktop session on top of Ubuntu Core
- Same benefits, but for user facing GUIs
  - security: harder for malicious software to change the system or spread themselves
  - stability: updates can't leave the system in an unstable state
  - reproducibility: easier to audit and verify the system
  - manageability: no inconsistencies from system to system
- Comes with extra challenges
  - Harder to draw the boundaries between desktop components
  - Quite some storage used

Ubuntu Core Desktop

A fully containerised desktop, where each component is immutable and isolated

- Announced last year
- Full desktop session on top of Ubuntu Core
- Same benefits, but for user facing GUIs
  - security: harder for malicious software to change the system or spread themselves
  - stability: updates can't leave the system in an unstable state
  - reproducibility: easier to audit and verify the system
  - manageability: no inconsistencies from system to system
- Comes with extra challenges
  - Harder to draw the boundaries between desktop components
  - Quite some storage used

Ubuntu Core Desktop

A fully containerised desktop, where each component is immutable and isolated

- Announced last year
- Full desktop session on top of Ubuntu Core
- Same benefits, but for user facing GUIs
  - security: harder for malicious software to change the system or spread themselves
  - stability: updates can't leave the system in an unstable state
  - reproducibility: easier to audit and verify the system
  - manageability: no inconsistencies from system to system
- Comes with extra challenges
  - Harder to draw the boundaries between desktop components
  - Quite some storage used

KDE Neon Core

All the KDE Neon benefits on top of Ubuntu Core

- Follows a similar architecture to Ubuntu Core Desktop
- Plasma based user experience
- Greatest and latest KDE software
- Also provides building blocks to snap package KDE applications for use out of Ubuntu Core

Snap Confinement Basics

Making a Snap

- Requires a build recipe (`snapcraft.yaml`)
- `snapcraft` will build the package

- Recipe structure
  - Metadata
  - Targeted base system
  - Apps provided in the package
  - Interfaces (slots and plugs)
  - Packages needed for building
  - How to build each part

Making a Snap

- Requires a build recipe (`snapcraft.yaml`)
- `snapcraft` will build the package
- Recipe structure
  - Metadata
  - Targeted base system
  - Apps provided in the package
  - Interfaces (slots and plugs)
  - Packages needed for building
  - How to build each part

Making a Snap cont'd

Recipe extract

```
name: ark
confinement: strict
grade: stable
base: core22
adopt-info: ark
apps:
    ark:
        extensions:
            - kde-neon-6 # <= forces settings useful for all KDE applications
        common-id: org.kde.ark.desktop
        desktop: usr/share/applications/org.kde.ark.desktop
        command: usr/bin/ark
        plugs:
            - home
            - system-backup
slots:
    session-dbus-interface:
        interface: dbus
        name: org.kde.ark
        bus: session
[...]
```

Making a Snap cont'd

What's in the kde-neon-6 extension?

- Ensures the right environment at application start (`$PATH`, `$XDG_*`, etc.)
- Declares build time and runtime dependencies on KDE Frameworks and Qt
- Declares common plugs, in particular
  - `desktop`
  - `opengl`
  - `wayland`
  - `x11`
  - `audio-playback`

How Does It Work?

- When an application is launched the following happens
  - snap-confine sets up the execution environment
    - `$HOME`, `$SNAP` and `$SNAP_*` environment variables are set
    - a private mount namespace is set
    - a private `/tmp` directory is set
    - command specific seccomp filter is put in place
    - command specific apparmor profile is put in place
    - hand over to snap-exec started in this new execution environment
  - snap-exec reads `meta.yaml` and launches the correct command
- Applications can also be declared as daemons
  - This leads to a systemd service which simply does a `snap run`
- Where are the seccomp filters and apparmor profiles coming from?
  - snapd creates them when packages are installed/removed
  - snapd updates them when interfaces are connected/disconnected
- Corollary: snapd has code mapping interface states to seccomp and apparmor templates

How Does It Work?

- When an application is launched the following happens
  - snap-confine sets up the execution environment
    - `$HOME`, `$SNAP` and `$SNAP_*` environment variables are set
    - a private mount namespace is set
    - a private `/tmp` directory is set
    - command specific seccomp filter is put in place
    - command specific apparmor profile is put in place
    - hand over to snap-exec started in this new execution environment
  - snap-exec reads `meta.yaml` and launches the correct command
- Applications can also be declared as daemons
  - This leads to a systemd service which simply does a `snap run`
- Where are the seccomp filters and apparmor profiles coming from?
  - snapd creates them when packages are installed/removed
  - snapd updates them when interfaces are connected/disconnected
- Corollary: snapd has code mapping interface states to seccomp and apparmor templates

How Does It Work?

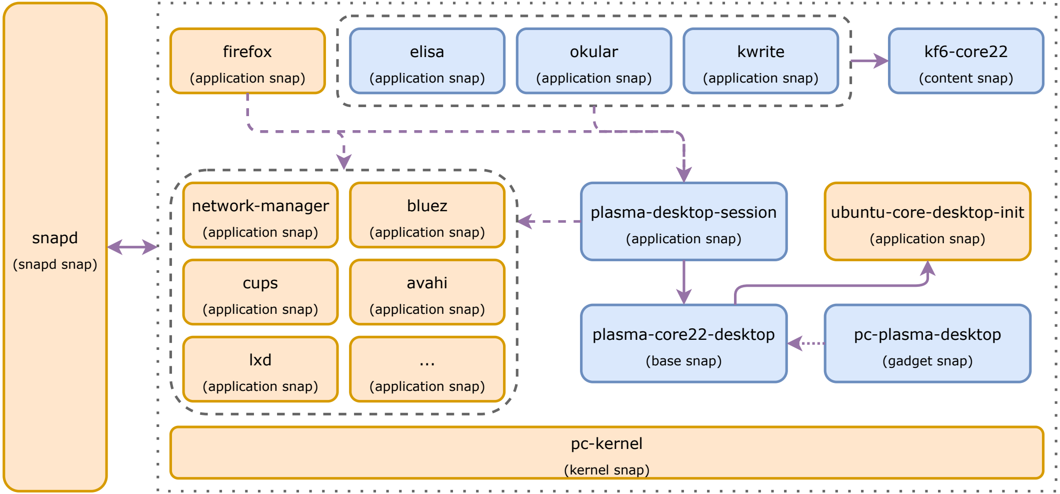
- When an application is launched the following happens
  - snap-confine sets up the execution environment
    - `$HOME`, `$SNAP` and `$SNAP_*` environment variables are set
    - a private mount namespace is set
    - a private `/tmp` directory is set
    - command specific seccomp filter is put in place
    - command specific apparmor profile is put in place
    - hand over to snap-exec started in this new execution environment
  - snap-exec reads `meta.yaml` and launches the correct command
- Applications can also be declared as daemons
  - This leads to a systemd service which simply does a `snap run`
- Where are the seccomp filters and apparmor profiles coming from?
  - snapd creates them when packages are installed/removed
  - snapd updates them when interfaces are connected/disconnected
- Corollary: snapd has code mapping interface states to seccomp and apparmor templates

How Does It Work?

- When an application is launched the following happens
  - snap-confine sets up the execution environment
    - `$HOME`, `$SNAP` and `$SNAP_*` environment variables are set
    - a private mount namespace is set
    - a private `/tmp` directory is set
    - command specific seccomp filter is put in place
    - command specific apparmor profile is put in place
    - hand over to snap-exec started in this new execution environment
  - snap-exec reads `meta.yaml` and launches the correct command
- Applications can also be declared as daemons
  - This leads to a systemd service which simply does a `snap run`
- Where are the seccomp filters and apparmor profiles coming from?
  - snapd creates them when packages are installed/removed
  - snapd updates them when interfaces are connected/disconnected
- Corollary: snapd has code mapping interface states to seccomp and apparmor templates

KDE Neon Core Architecture

The Important Parts



- `plasma-core22-desktop` is populated with KDE Neon debian packages
- application and content snaps are populated by building from the code

First Boot Provisioning

- SDDM is provided by `plasma-core22-desktop`
- It is ran outside of confinement

- On first boot
  - No regular user is available
  - SDDM config is overloaded to auto-login as root in a special session
  - It only starts the `ubuntu-core-desktop-init` wizard

- Once a user is provisioned (thanks to the wizard)
  - SDDM config is reset to defaults

- Interestingly GDM requires no extra work for this
- The first run and provisioning feature is built in

First Boot Provisioning

- SDDM is provided by `plasma-core22-desktop`
- It is ran outside of confinement

- On first boot
  - No regular user is available
  - SDDM config is overloaded to auto-login as root in a special session
  - It only starts the `ubuntu-core-desktop-init` wizard

- Once a user is provisioned (thanks to the wizard)
  - SDDM config is reset to defaults

- Interestingly GDM requires no extra work for this
- The first run and provisioning feature is built in

First Boot Provisioning

- SDDM is provided by `plasma-core22-desktop`
- It is ran outside of confinement

- On first boot
  - No regular user is available
  - SDDM config is overloaded to auto-login as root in a special session
  - It only starts the `ubuntu-core-desktop-init` wizard

- Once a user is provisioned (thanks to the wizard)
  - SDDM config is reset to defaults

- Interestingly GDM requires no extra work for this
- The first run and provisioning feature is built in

First Boot Provisioning

- SDDM is provided by `plasma-core22-desktop`
- It is ran outside of confinement

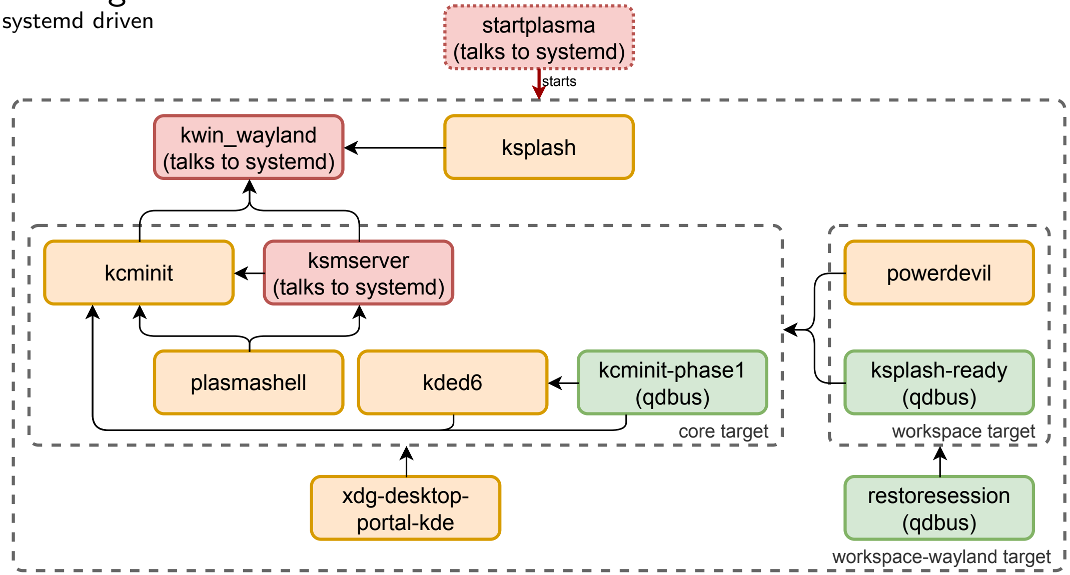
- On first boot
  - No regular user is available
  - SDDM config is overloaded to auto-login as root in a special session
  - It only starts the `ubuntu-core-desktop-init` wizard

- Once a user is provisioned (thanks to the wizard)
  - SDDM config is reset to defaults

- Interestingly GDM requires no extra work for this
- The first run and provisioning feature is built in

Starting Plasma

systemd driven



Warning: This graph is overly simplified and lying on purpose

- `startplasma` basically requests a single target and kickstarts the process
- systemd environment is manipulated by `startplasma`, `kwin` and `ksmserver`

Unconfined Startup

- Should be easy right?
- We declared `startplasma` and `xdg-desktop-portal-kde` as applications
- And yet. . . we would get a black screen!

- A deadlock between `kwin_wayland` and `xdg-desktop-portal-kde`
- With `kwin_wayland` stuck nothing could proceed
- But why the deadlock in the first place?

- Remember the `$SNAP` environment variable?
- Turns out that if set, `QGuiApplication` loads the `xdgdesktopportal` platform theme (among other things)

- This creates interesting runtime dependencies:
  - `kwin_wayland` → `xdgdesktopportal` → `xdg-desktop-portal-kde` → `kwin_wayland`
  - `xdg-desktop-portal-kde` → `xdgdesktopportal` → `xdg-desktop-portal-kde`

Unconfined Startup

- Should be easy right?
- We declared `startplasma` and `xdg-desktop-portal-kde` as applications
- And yet... we would get a black screen!

- A deadlock between `kwin_wayland` and `xdg-desktop-portal-kde`
- With `kwin_wayland` stuck nothing could proceed
- But why the deadlock in the first place?

- Remember the $SNAP environment variable?
- Turns out that if set, `QGuiApplication` loads the `xdgdesktopportal` platform theme (among other things)

- This creates interesting runtime dependencies:
  - `kwin_wayland` → `xdgdesktopportal` → `xdg-desktop-portal-kde` → `kwin_wayland`
  - `xdg-desktop-portal-kde` → `xdgdesktopportal` → `xdg-desktop-portal-kde`

Unconfined Startup

- Should be easy right?
- We declared `startplasma` and `xdg-desktop-portal-kde` as applications
- And yet... we would get a black screen!

- A deadlock between `kwin_wayland` and `xdg-desktop-portal-kde`
- With `kwin_wayland` stuck nothing could proceed
- But why the deadlock in the first place?

- Remember the `$SNAP` environment variable?
- Turns out that if set, `QGuiApplication` loads the `xdgdesktopportal` platform theme (among other things)

- This creates interesting runtime dependencies:
  - `kwin_wayland` → `xdgdesktopportal` → `xdg-desktop-portal-kde` → `kwin_wayland`
  - `xdg-desktop-portal-kde` → `xdgdesktopportal` → `xdg-desktop-portal-kde`

Unconfined Startup

- Should be easy right?
- We declared `startplasma` and `xdg-desktop-portal-kde` as applications
- And yet. . . we would get a black screen!

- A deadlock between `kwin_wayland` and `xdg-desktop-portal-kde`
- With `kwin_wayland` stuck nothing could proceed
- But why the deadlock in the first place?

- Remember the `$SNAP` environment variable?
- Turns out that if set, `QGuiApplication` loads the `xdgdesktopportal` platform theme (among other things)

- This creates interesting runtime dependencies:
  - `kwin_wayland` → `xdgdesktopportal` → `xdg-desktop-portal-kde` → `kwin_wayland`
  - `xdg-desktop-portal-kde` → `xdgdesktopportal` → `xdg-desktop-portal-kde`

Unconfined Startup cont'd

- This variable was inherited from `startplasma` due to the way it pushes its environment to systemd via `UpdateActivationEnvironment`
- Several ways to reduce or avoid the issue
  - Remove the `xdgdesktopportal` platform theme from the base snap (not perfect due to other side-effects)
  - Add a oneshot systemd service executed before kwin to cleanup the systemd environment
  - Modify `startplasma` to not push confinement related environment variables to systemd
- This got us a working desktop in all its glory!
- It was still unconfined though. . .

Unconfined Startup cont'd

- This variable was inherited from `startplasma` due to the way it pushes its environment to systemd via `UpdateActivationEnvironment`
- Several ways to reduce or avoid the issue
  - Remove the `xdgdesktopportal` platform theme from the base snap (not perfect due to other side-effects)
  - Add a oneshot systemd service executed before kwin to cleanup the systemd environment
  - Modify `startplasma` to not push confinement related environment variables to systemd
- This got us a working desktop in all its glory!
- It was still unconfined though. . .

Unconfined Startup cont'd

- This variable was inherited from `startplasma` due to the way it pushes its environment to systemd via `UpdateActivationEnvironment`
- Several ways to reduce or avoid the issue
  - Remove the `xdgdesktopportal` platform theme from the base snap (not perfect due to other side-effects)
  - Add a oneshot systemd service executed before kwin to cleanup the systemd environment
  - Modify `startplasma` to not push confinement related environment variables to systemd
- This got us a working desktop in all its glory!
- It was still unconfined though. . .

Opening The Snapd Interfaces Pandora Box

- Now that we want to confine the session things will break badly again
- Confined processes aren't allowed to call `StartUnit` or `UpdateActivationEnvironment` on the user systemd...
- Time to get the power drill out!
- We submitted a new `systemd-user-control` interface
- The AppArmor profile of an application is changed when having it as plug
- `StartUnit` and `UpdateActivationEnvironment` become allowed

- This is risky as it allows the application to talk to systemd directly
- A good way to start applications unconfined
- Snap packages using it would need very fine review, not many are to be trusted with it

- As a matter of fact this is still in discussion due to this...

Opening The Snapd Interfaces Pandora Box

- Now that we want to confine the session things will break badly again
- Confined processes aren't allowed to call `StartUnit` or `UpdateActivationEnvironment` on the user systemd...

- Time to get the power drill out!
- We submitted a new `systemd-user-control` interface
- The AppArmor profile of an application is changed when having it as plug
- `StartUnit` and `UpdateActivationEnvironment` become allowed

- This is risky as it allows the application to talk to systemd directly
- A good way to start applications unconfined
- Snap packages using it would need very fine review, not many are to be trusted with it

- As a matter of fact this is still in discussion due to this...

Opening The Snapd Interfaces Pandora Box

- Now that we want to confine the session things will break badly again
- Confined processes aren't allowed to call `StartUnit` or `UpdateActivationEnvironment` on the user systemd…

- Time to get the power drill out!
- We submitted a new `systemd-user-control` interface
- The AppArmor profile of an application is changed when having it as plug
- `StartUnit` and `UpdateActivationEnvironment` become allowed

- This is risky as it allows the application to talk to systemd directly
- A good way to start applications unconfined
- Snap packages using it would need very fine review, not many are to be trusted with it

- As a matter of fact this is still in discussion due to this…

Opening The Snapd Interfaces Pandora Box

- Now that we want to confine the session things will break badly again
- Confined processes aren't allowed to call `StartUnit` or `UpdateActivationEnvironment` on the user systemd...

- Time to get the power drill out!
- We submitted a new `systemd-user-control` interface
- The AppArmor profile of an application is changed when having it as plug
- `StartUnit` and `UpdateActivationEnvironment` become allowed

- This is risky as it allows the application to talk to systemd directly
- A good way to start applications unconfined
- Snap packages using it would need very fine review, not many are to be trusted with it

- As a matter of fact this is still in discussion due to this...

A Note About KIO

- During startup we also hit `KIO::KProcessRunner` for various tasks
- This one would call `StartTransientUnit`

- `StartTransientUnit` is even more frowned upon than `StartUnit` security wise
- Can get you to start truly anything unconfined rather easily
- It doesn't require a preexisting unit declaration

- Luckily positioning `$_KDE_APPLICATIONS_AS_FORKING` does the trick
- It enforces the use of the old `fork()` based code path

A Note About KIO

- During startup we also hit `KIO::KProcessRunner` for various tasks
- This one would call `StartTransientUnit`

- `StartTransientUnit` is even more frowned upon than `StartUnit` security wise
- Can get you to start truly anything unconfined rather easily
- It doesn't require a preexisting unit declaration

- Luckily positioning `$_KDE_APPLICATIONS_AS_FORKING` does the trick
- It enforces the use of the old `fork()` based code path

A Note About KIO

- During startup we also hit `KIO::KProcessRunner` for various tasks
- This one would call `StartTransientUnit`

- `StartTransientUnit` is even more frowned upon than `StartUnit` security wise
- Can get you to start truly anything unconfined rather easily
- It doesn't require a preexisting unit declaration

- Luckily positioning `$_KDE_APPLICATIONS_AS_FORKING` does the trick
- It enforces the use of the old `fork()` based code path

Confining The Session

- Thanks to the `systemd-user-control` plug we could start the confined session!
- Not everything was properly working though (ksplash for instance)

- Long story short, further adjustments to snapd were needed
- This was all tested only with a GNOME Shell desktop previously

- Plasma sessions use the same D-Bus interfaces but a bit differently
- They also tend to introspect more agressively

- We thus improved the following interfaces for Plasma sessions
  - `desktop`
  - `upower-observe`
  - `system-observe`
  - `shutdown`

- We also declared all the D-Bus services the session would bind to
- And then the session was working properly from startup to shutdown

Confining The Session

- Thanks to the `systemd-user-control` plug we could start the confined session!
- Not everything was properly working though (ksplash for instance)

- Long story short, further adjustments to snapd were needed
- This was all tested only with a GNOME Shell desktop previously

- Plasma sessions use the same D-Bus interfaces but a bit differently
- They also tend to introspect more agressively

- We thus improved the following interfaces for Plasma sessions
  - `desktop`
  - `upower-observe`
  - `system-observe`
  - `shutdown`

- We also declared all the D-Bus services the session would bind to
- And then the session was working properly from startup to shutdown

Confining The Session

- Thanks to the `systemd-user-control` plug we could start the confined session!
- Not everything was properly working though (ksplash for instance)

- Long story short, further adjustments to snapd were needed
- This was all tested only with a GNOME Shell desktop previously

- Plasma sessions use the same D-Bus interfaces but a bit differently
- They also tend to introspect more agressively

- We thus improved the following interfaces for Plasma sessions
  - `desktop`
  - `upower-observe`
  - `system-observe`
  - `shutdown`

- We also declared all the D-Bus services the session would bind to
- And then the session was working properly from startup to shutdown

Confining The Session

- Thanks to the `systemd-user-control` plug we could start the confined session!
- Not everything was properly working though (ksplash for instance)

- Long story short, further adjustments to snapd were needed
- This was all tested only with a GNOME Shell desktop previously

- Plasma sessions use the same D-Bus interfaces but a bit differently
- They also tend to introspect more agressively

- We thus improved the following interfaces for Plasma sessions
  - `desktop`
  - `upower-observe`
  - `system-observe`
  - `shutdown`

- We also declared all the D-Bus services the session would bind to
- And then the session was working properly from startup to shutdown

Confining The Session

- Thanks to the `systemd-user-control` plug we could start the confined session!
- Not everything was properly working though (ksplash for instance)

- Long story short, further adjustments to snapd were needed
- This was all tested only with a GNOME Shell desktop previously

- Plasma sessions use the same D-Bus interfaces but a bit differently
- They also tend to introspect more agressively

- We thus improved the following interfaces for Plasma sessions
  - `desktop`
  - `upower-observe`
  - `system-observe`
  - `shutdown`

- We also declared all the D-Bus services the session would bind to
- And then the session was working properly from startup to shutdown

Confining The Session (Take 2)

Oopsie!

- Happy to have a proper session, we stayed like this and worked on other tasks
- But something was not feeling quite right…
- Until a comment on one of our snapd adjustments was our wake up call

- `systemd-cgls` confirmed that some very important processes were not confined
- Any service not declared as application in the snap would use their regular systemd service file… bypassing snapd!

- Back to the drawing board…
- Aliases to the rescue to overload the `plasma-*.service` files using snapd ones

- This required improving further snapd interfaces:
  - `screen_inhibit_control` and `login_session_observe`
  - We also adjusted `desktop` some more

- Finally back on track with most processes properly confined
- Yes, even `kwin`, `plasmashell` and `kded6` having specific confinement rules

Confining The Session (Take 2)

Oopsie!

- Happy to have a proper session, we stayed like this and worked on other tasks
- But something was not feeling quite right. . .
- Until a comment on one of our snapd adjustments was our wake up call

- `systemd-cgls` confirmed that some very important processes were not confined
- Any service not declared as application in the snap would use their regular systemd service file. . . bypassing snapd!

- Back to the drawing board. . .
- Aliases to the rescue to overload the `plasma-*.service` files using snapd ones

- This required improving further snapd interfaces:
  - `screen_inhibit_control` and `login_session_observe`
  - We also adjusted `desktop` some more

- Finally back on track with most processes properly confined
- Yes, even `kwin`, `plasmashell` and `kded6` having specific confinement rules

Confining The Session (Take 2)

Oopsie!

- Happy to have a proper session, we stayed like this and worked on other tasks
- But something was not feeling quite right...
- Until a comment on one of our snapd adjustments was our wake up call

- `systemd-cgls` confirmed that some very important processes were not confined
- Any service not declared as application in the snap would use their regular systemd service file... bypassing snapd!

- Back to the drawing board...
- Aliases to the rescue to overload the `plasma-*.service` files using snapd ones

- This required improving further snapd interfaces:
  - `screen_inhibit_control` and `login_session_observe`
  - We also adjusted `desktop` some more

- Finally back on track with most processes properly confined
- Yes, even `kwin`, `plasmashell` and `kded6` having specific confinement rules

Confining The Session (Take 2)

Oopsie!

- Happy to have a proper session, we stayed like this and worked on other tasks
- But something was not feeling quite right. . .
- Until a comment on one of our snapd adjustments was our wake up call

- `systemd-cgls` confirmed that some very important processes were not confined
- Any service not declared as application in the snap would use their regular systemd service file. . . bypassing snapd!

- Back to the drawing board. . .
- Aliases to the rescue to overload the `plasma-*.service` files using snapd ones

- This required improving further snapd interfaces:
  - `screen_inhibit_control` and `login_session_observe`
  - We also adjusted `desktop` some more

- Finally back on track with most processes properly confined
- Yes, even `kwin`, `plasmashell` and `kded6` having specific confinement rules

Confining The Session (Take 2)

Oopsie!

- Happy to have a proper session, we stayed like this and worked on other tasks
- But something was not feeling quite right. . .
- Until a comment on one of our snapd adjustments was our wake up call

- `systemd-cgls` confirmed that some very important processes were not confined
- Any service not declared as application in the snap would use their regular systemd service file. . . bypassing snapd!

- Back to the drawing board. . .
- Aliases to the rescue to overload the `plasma-*.service` files using snapd ones

- This required improving further snapd interfaces:
  - `screen_inhibit_control` and `login_session_observe`
  - We also adjusted `desktop` some more

- Finally back on track with most processes properly confined
- Yes, even `kwin`, `plasmashell` and `kded6` having specific confinement rules

Launching Apps

- But how can `plasmashell` or `krunner` start other applications?
- One point of the confined processes is their inability to call `snap run`...
- How are other applications started?

- There is a pending snapd feature for this
- snapd generates desktop files for the declared applications

Launching Apps

- But how can `plasmashell` or `krunner` start other applications?
- One point of the confined processes is their inability to call `snap run`. . .
- How are other applications started?

- There is a pending snapd feature for this
- snapd generates desktop files for the declared applications

Launching Apps cont'd

- The Exec line is:
  `snap routine desktop-launch --desktop <desktop file>`
- This would in turn talk to `io.snapcraft.PrivilegedDesktopLauncher`
- Only allowed if the requesting application has the `desktop-launch` plug

- "Fun fact", this would also horribly break if the `$SNAP` variable is leaked to the systemd environment
  - Applications wouldn't start
  - Any application connected to the `desktop` slot of `plasma-desktop-session` would prevent the startup
    - Don't you like black screens by now?

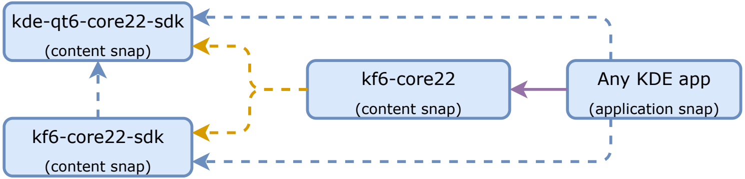
Launching Apps cont'd

- The Exec line is:
  `snap routine desktop-launch --desktop <desktop file>`
- This would in turn talk to `io.snapcraft.PrivilegedDesktopLauncher`
- Only allowed if the requesting application has the `desktop-launch` plug

- "Fun fact", this would also horribly break if the `$SNAP` variable is leaked to the systemd environment
  - Applications wouldn't start
  - Any application connected to the `desktop` slot of `plasma-desktop-session` would prevent the startup
    - Don't you like black screens by now?

What About the Apps?

Building Blocks



kde-qt6-core22-sdk
(content snap)

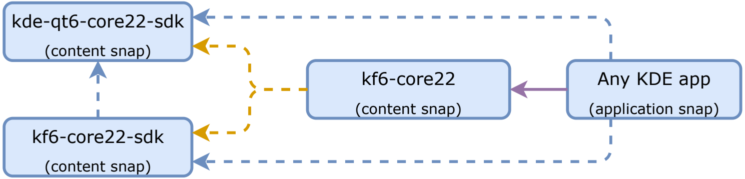
kf6-core22-sdk
(content snap)

kf6-core22
(content snap)

Any KDE app
(application snap)

- Unlike the session snaps, application and content snaps are populated by building from the code
- Theming comes with its own challenges though
- This is hardly extensible through the content snap
- Also `QStyle` means binaries, this is harder to maintain. . .

Building Blocks



- Unlike the session snaps, application and content snaps are populated by building from the code

- Theming comes with its own challenges though
- This is hardly extensible through the content snap
- Also `QStyle` means binaries, this is harder to maintain...

Covered Applications

- As part of this project we covered a few applications
  - Discover
  - Gwenview
  - Okular
  - KWrite
  - Elisa
- This allows us to have basic use cases covered
  - viewing documents, images, or videos, listening to music, editing files
- They are also good blueprints for further app packaging
- Good news: most of them worked out of the box without patching
- Only Discover required patches due to its stronger proximity with snapd
  - it was assuming auto-update of packages, but this can be disabled so we fixed it
  - it was going through `snap run` to launch installed apps, unusable when confined

Covered Applications

- As part of this project we covered a few applications
  - Discover
  - Gwenview
  - Okular
  - KWrite
  - Elisa
- This allows us to have basic use cases covered
  - viewing documents, images, or videos, listening to music, editing files
- They are also good blueprints for further app packaging
- Good news: most of them worked out of the box without patching
- Only Discover required patches due to its stronger proximity with snapd
  - it was assuming auto-update of packages, but this can be disabled so we fixed it
  - it was going through `snap run` to launch installed apps, unusable when confined

Covered Applications

- As part of this project we covered a few applications
  - Discover
  - Gwenview
  - Okular
  - KWrite
  - Elisa
- This allows us to have basic use cases covered
  - viewing documents, images, or videos, listening to music, editing files
- They are also good blueprints for further app packaging
- Good news: most of them worked out of the box without patching
- Only Discover required patches due to its stronger proximity with snapd
  - it was assuming auto-update of packages, but this can be disabled so we fixed it
  - it was going through `snap run` to launch installed apps, unusable when confined

Covered Applications

- As part of this project we covered a few applications
  - Discover
  - Gwenview
  - Okular
  - KWrite
  - Elisa
- This allows us to have basic use cases covered
  - viewing documents, images, or videos, listening to music, editing files
- They are also good blueprints for further app packaging
- Good news: most of them worked out of the box without patching
- Only Discover required patches due to its stronger proximity with snapd
  - it was assuming auto-update of packages, but this can be disabled so we fixed it
  - it was going through `snap run` to launch installed apps, unusable when confined

Automated Tests

Under construction...

- We'd like to see the snap packages validated and tested before publication
- This means having appium tests!

- So far only KWrite, Okular and Gwenview have a test suite

- This uncovered issues with the AT-SPI WebDriver
  - One function of the API wasn't conform to the specification, making some tests harder to write
  - It was not possible to override the way the application is started (and we need it to go through `snap run`)

- We're still seeing flakiness with some tests on the CI, this still needs to be investigated

Automated Tests

Under construction...

- We'd like to see the snap packages validated and tested before publication
- This means having appium tests!

- So far only KWrite, Okular and Gwenview have a test suite

- This uncovered issues with the AT-SPI WebDriver
  - One function of the API wasn't conform to the specification, making some tests harder to write
  - It was not possible to override the way the application is started (and we need it to go through `snap run`)

- We're still seeing flakiness with some tests on the CI, this still needs to be investigated

Automated Tests

Under construction...

- We'd like to see the snap packages validated and tested before publication
- This means having appium tests!
- So far only KWrite, Okular and Gwenview have a test suite
- This uncovered issues with the AT-SPI WebDriver
  - One function of the API wasn't conform to the specification, making some tests harder to write
  - It was not possible to override the way the application is started (and we need it to go through `snap run`)
- We're still seeing flakiness with some tests on the CI, this still needs to be investigated

Automated Tests

Under construction...

- We'd like to see the snap packages validated and tested before publication
- This means having appium tests!
- So far only KWrite, Okular and Gwenview have a test suite
- This uncovered issues with the AT-SPI WebDriver
  - One function of the API wasn't conform to the specification, making some tests harder to write
  - It was not possible to override the way the application is started (and we need it to go through `snap run`)
- We're still seeing flakiness with some tests on the CI, this still needs to be investigated

CI/CD

Build And Publish Snaps

- We naively assumed snap building on the KDE CI was already working
- Turns out the plug was pulled a few weeks before we started our project
- So we had to build something new. . .
- `snapcraft` proved to be difficult to run inside rootless containers. . .
- So we moved everything to a specific VM provided by sysadmin
- We have SSH access there, it will serve as a blueprint for ephemeral VMs later on
- It properly builds snaps and push them to the store
- It only pushes to the `latest/edge` channel for now
- There are plans to push on different channels depending on the branch
- Once the flakiness issues with the AT-SPI WebDriver are solved, appium tests will be used to check the snap is working before pushing

Build And Publish Snaps

- We naively assumed snap building on the KDE CI was already working

- Turns out the plug was pulled a few weeks before we started our project
- So we had to build something new. . .

- `snapcraft` proved to be difficult to run inside rootless containers. . .

- So we moved everything to a specific VM provided by sysadmin
- We have SSH access there, it will serve as a blueprint for ephemeral VMs later on

- It properly builds snaps and push them to the store
- It only pushes to the `latest/edge` channel for now
- There are plans to push on different channels depending on the branch

- Once the flakiness issues with the AT-SPI WebDriver are solved, appium tests will be used to check the snap is working before pushing

Build And Publish Snaps

- We naively assumed snap building on the KDE CI was already working

- Turns out the plug was pulled a few weeks before we started our project
- So we had to build something new. . .

- `snapcraft` proved to be difficult to run inside rootless containers. . .

- So we moved everything to a specific VM provided by sysadmin
- We have SSH access there, it will serve as a blueprint for ephemeral VMs later on

- It properly builds snaps and push them to the store
- It only pushes to the `latest/edge` channel for now
- There are plans to push on different channels depending on the branch

- Once the flakiness issues with the AT-SPI WebDriver are solved, appium tests will be used to check the snap is working before pushing

Build And Publish Snaps

- We naively assumed snap building on the KDE CI was already working

- Turns out the plug was pulled a few weeks before we started our project
- So we had to build something new. . .

- `snapcraft` proved to be difficult to run inside rootless containers. . .

- So we moved everything to a specific VM provided by sysadmin
- We have SSH access there, it will serve as a blueprint for ephemeral VMs later on

- It properly builds snaps and push them to the store
- It only pushes to the `latest/edge` channel for now
- There are plans to push on different channels depending on the branch

- Once the flakiness issues with the AT-SPI WebDriver are solved, appium tests will be used to check the snap is working before pushing

Build And Publish Snaps

- We naively assumed snap building on the KDE CI was already working
- Turns out the plug was pulled a few weeks before we started our project
- So we had to build something new...
- `snapcraft` proved to be difficult to run inside rootless containers...
- So we moved everything to a specific VM provided by sysadmin
- We have SSH access there, it will serve as a blueprint for ephemeral VMs later on
- It properly builds snaps and push them to the store
- It only pushes to the `latest/edge` channel for now
- There are plans to push on different channels depending on the branch
- Once the flakiness issues with the AT-SPI WebDriver are solved, appium tests will be used to check the snap is working before pushing

Build And Publish Snaps

- We naively assumed snap building on the KDE CI was already working

- Turns out the plug was pulled a few weeks before we started our project
- So we had to build something new. . .

- `snapcraft` proved to be difficult to run inside rootless containers. . .

- So we moved everything to a specific VM provided by sysadmin
- We have SSH access there, it will serve as a blueprint for ephemeral VMs later on

- It properly builds snaps and push them to the store
- It only pushes to the `latest/edge` channel for now
- There are plans to push on different channels depending on the branch

- Once the flakiness issues with the AT-SPI WebDriver are solved, appium tests will be used to check the snap is working before pushing

Building Images

- `ubuntu-image` is used to assemble a system image based on our snaps
- It was not easy to setup on our CI

- Input files (called models) need to be signed
- We tried to have developers signing locally with their own keys to make test images
- While the CI was signing with the official KDE key to make published images

- Turned out to be a problem
  - Admittedly cumbersome for each developers to have keys to manage
  - Also tooling would later prevent building the image in some circumstances (some key snap packages need to be signed with the same key as the model)

- We couldn't rely on using "Ben as a Service" to sign the models when they change!
- Instead we have a manual CI job meant to create updated models
- Those are always signed with the KDE key
- This is much easier
- Developers still need to download the resulting artifacts and commit them

Building Images

- `ubuntu-image` is used to assemble a system image based on our snaps
- It was not easy to setup on our CI

- Input files (called models) need to be signed
- We tried to have developers signing locally with their own keys to make test images
- While the CI was signing with the official KDE key to make published images

- Turned out to be a problem
  - Admittedly cumbersome for each developers to have keys to manage
  - Also tooling would later prevent building the image in some circumstances (some key snap packages need to be signed with the same key as the model)

- We couldn't rely on using "Ben as a Service" to sign the models when they change!
- Instead we have a manual CI job meant to create updated models
- Those are always signed with the KDE key
- This is much easier
- Developers still need to download the resulting artifacts and commit them

Building Images

- `ubuntu-image` is used to assemble a system image based on our snaps
- It was not easy to setup on our CI

- Input files (called models) need to be signed
- We tried to have developers signing locally with their own keys to make test images
- While the CI was signing with the official KDE key to make published images

- Turned out to be a problem
  - Admittedly cumbersome for each developers to have keys to manage
  - Also tooling would later prevent building the image in some circumstances (some key snap packages need to be signed with the same key as the model)

- We couldn't rely on using "Ben as a Service" to sign the models when they change!
- Instead we have a manual CI job meant to create updated models
- Those are always signed with the KDE key
- This is much easier
- Developers still need to download the resulting artifacts and commit them

Building Images

- `ubuntu-image` is used to assemble a system image based on our snaps
- It was not easy to setup on our CI

- Input files (called models) need to be signed
- We tried to have developers signing locally with their own keys to make test images
- While the CI was signing with the official KDE key to make published images

- Turned out to be a problem
  - Admittedly cumbersome for each developers to have keys to manage
  - Also tooling would later prevent building the image in some circumstances (some key snap packages need to be signed with the same key as the model)

- We couldn't rely on using "Ben as a Service" to sign the models when they change!
- Instead we have a manual CI job meant to create updated models
- Those are always signed with the KDE key
- This is much easier
- Developers still need to download the resulting artifacts and commit them

Building ISOs

- System images are nice for quickly testing, you can just spawn qemu
  - We even provide a script in the repository to do this
- But really, people will need ISOs to install on a computer or in a VM
- Again this was incompatible with rootless containers
- Lots of the steps require root
  - e.g. a few bind mounts are needed
- We can't wait to see the KDE Sysadmins deliver their project to switch from containers to VMs!

Building ISOs

- System images are nice for quickly testing, you can just spawn qemu
  - We even provide a script in the repository to do this
- But really, people will need ISOs to install on a computer or in a VM
- Again this was incompatible with rootless containers
- Lots of the steps require root
  - e.g. a few bind mounts are needed
- We can't wait to see the KDE Sysadmins deliver their project to switch from containers to VMs!

Building ISOs

- System images are nice for quickly testing, you can just spawn qemu
  - We even provide a script in the repository to do this
- But really, people will need ISOs to install on a computer or in a VM
- Again this was incompatible with rootless containers
- Lots of the steps require root
  - e.g. a few bind mounts are needed
- We can't wait to see the KDE Sysadmins deliver their project to switch from containers to VMs!

Building ISOs

- System images are nice for quickly testing, you can just spawn qemu
  - We even provide a script in the repository to do this
- But really, people will need ISOs to install on a computer or in a VM
- Again this was incompatible with rootless containers
- Lots of the steps require root
  - e.g. a few bind mounts are needed
- We can't wait to see the KDE Sysadmins deliver their project to switch from containers to VMs!

Development Challenges

Encountered Problems Recap

- In the assembled system we covered several issues
  - Black screens of various provenance (with and without confinement)
  - Applications not being started
  - Applications not doing what they should

- How did we approach those?

Problems With a Snap Application

- Can be anything ranging from not starting to weird GUI glitches
- Probably worth checking if it is reproduceable outside of Ubuntu Core
- In this case you can use regular snap troubleshooting recipes

- `snappy-debug`
  - To process logs and point Seccomp or AppArmor violations
  - It even suggests fixes
- `snap run --shell`
  - To introspect the process environment
  - It greatly helps to understand how an application “sees the system”
- `snap run --strace`
  - To ease syscall debugging
  - It helps to find system call failures
- `snap run --gdbserver`
  - To run the application with gdb
  - debuginfod support will be required

Problems With a Snap Application

- Can be anything ranging from not starting to weird GUI glitches
- Probably worth checking if it is reproduceable outside of Ubuntu Core
- In this case you can use regular snap troubleshooting recipes

- `snappy-debug`
  - To process logs and point Seccomp or AppArmor violations
  - It even suggests fixes

- `snap run --shell`
  - To introspect the process environment
  - It greatly helps to understand how an application “sees the system”

- `snap run --strace`
  - To ease syscall debugging
  - It helps to find system call failures

- `snap run --gdbserver`
  - To run the application with gdb
  - debuginfod support will be required

Problems With a Snap Application

- Can be anything ranging from not starting to weird GUI glitches
- Probably worth checking if it is reproduceable outside of Ubuntu Core
- In this case you can use regular snap troubleshooting recipes

- `snappy-debug`
  - To process logs and point Seccomp or AppArmor violations
  - It even suggests fixes

- `snap run --shell`
  - To introspect the process environment
  - It greatly helps to understand how an application “sees the system”

- `snap run --strace`
  - To ease syscall debugging
  - It helps to find system call failures

- `snap run --gdbserver`
  - To run the application with gdb
  - debuginfod support will be required

Problems With a Snap Application

- Can be anything ranging from not starting to weird GUI glitches
- Probably worth checking if it is reproduceable outside of Ubuntu Core
- In this case you can use regular snap troubleshooting recipes

- `snappy-debug`
  - To process logs and point Seccomp or AppArmor violations
  - It even suggests fixes

- `snap run --shell`
  - To introspect the process environment
  - It greatly helps to understand how an application “sees the system”

- `snap run --strace`
  - To ease syscall debugging
  - It helps to find system call failures

- `snap run --gdbserver`
  - To run the application with gdb
  - debuginfod support will be required

Problems With a Snap Application

- Can be anything ranging from not starting to weird GUI glitches
- Probably worth checking if it is reproduceable outside of Ubuntu Core
- In this case you can use regular snap troubleshooting recipes

- `snappy-debug`
  - To process logs and point Seccomp or AppArmor violations
  - It even suggests fixes

- `snap run --shell`
  - To introspect the process environment
  - It greatly helps to understand how an application “sees the system”

- `snap run --strace`
  - To ease syscall debugging
  - It helps to find system call failures

- `snap run --gdbserver`
  - To run the application with gdb
  - debuginfod support will be required

Problems With The Session

- `snappy-debug` can come in handy still
  - Be careful about the proposed fixes, they can be misleading in this context!
- Otherwise… a bit on your own regarding snap specific tooling
- This requires going straight to lower levels
- Rolling your own `plasma-desktop-session` snap
  - Not that hard or time consuming to iterate
  - Allows to easily modify session startup scripts
    - `export QT_LOGGING_RULES="*.debug=true"`
    - `export QDBUS_DEBUG=1`
    - `systemd-analyze --user set-log-level debug`
  - All the logs you can dream of!
- Rolling you own `plasma-core22-desktop` snap
  - We provide a `enable-developer-access.sh` to tune it
  - Opens root access on the first serial port
  - Installs extra developer tools (AppArmor, gdb and D-Bus related)
  - From there you can attach and debug anything
    - Configuring debuginfod is strongly recommended of course

Problems With The Session

- `snappy-debug` can come in handy still
  - Be careful about the proposed fixes, they can be misleading in this context!

- Otherwise... a bit on your own regarding snap specific tooling
- This requires going straight to lower levels

- Rolling your own `plasma-desktop-session` snap
  - Not that hard or time consuming to iterate
  - Allows to easily modify session startup scripts
    - `export QT_LOGGING_RULES="*.debug=true"`
    - `export QDBUS_DEBUG=1`
    - `systemd-analyze --user set-log-level debug`
  - All the logs you can dream of!

- Rolling you own `plasma-core22-desktop` snap
  - We provide a `enable-developer-access.sh` to tune it
  - Opens root access on the first serial port
  - Installs extra developer tools (AppArmor, gdb and D-Bus related)
  - From there you can attach and debug anything
    - Configuring debuginfod is strongly recommended of course

Problems With The Session

- `snappy-debug` can come in handy still
  - Be careful about the proposed fixes, they can be misleading in this context!
- Otherwise. . . a bit on your own regarding snap specific tooling
- This requires going straight to lower levels
- Rolling your own `plasma-desktop-session` snap
  - Not that hard or time consuming to iterate
  - Allows to easily modify session startup scripts
    - `export QT_LOGGING_RULES="*.debug=true"`
    - `export QDBUS_DEBUG=1`
    - `systemd-analyze --user set-log-level debug`
  - All the logs you can dream of!
- Rolling you own `plasma-core22-desktop` snap
  - We provide a `enable-developer-access.sh` to tune it
  - Opens root access on the first serial port
  - Installs extra developer tools (AppArmor, gdb and D-Bus related)
  - From there you can attach and debug anything
    - Configuring debuginfod is strongly recommended of course

Problems With The Session

- `snappy-debug` can come in handy still
  - Be careful about the proposed fixes, they can be misleading in this context!

- Otherwise... a bit on your own regarding snap specific tooling
- This requires going straight to lower levels

- Rolling your own `plasma-desktop-session` snap
  - Not that hard or time consuming to iterate
  - Allows to easily modify session startup scripts
    - `export QT_LOGGING_RULES="*.debug=true"`
    - `export QDBUS_DEBUG=1`
    - `systemd-analyze --user set-log-level debug`
  - All the logs you can dream of!

- Rolling you own `plasma-core22-desktop` snap
  - We provide a `enable-developer-access.sh` to tune it
  - Opens root access on the first serial port
  - Installs extra developer tools (AppArmor, gdb and D-Bus related)
  - From there you can attach and debug anything
    - Configuring debuginfod is strongly recommended of course

It Is All Immutable!

- This needs to be kept in mind
- Be strategic in what you can prioritise
- You can iterate quickly on
  - application snaps
  - `plasma-desktop-session` snap (requires logging out though)
  - even snapd (might require a reboot)
- But iterating on `plasma-core22-desktop` changes...
- They often require regenerating the system image
- This is a much slower feedback loop

It Is All Immutable!

- This needs to be kept in mind
- Be strategic in what you can prioritise

- You can iterate quickly on
  - application snaps
  - `plasma-desktop-session` snap (requires logging out though)
  - even snapd (might require a reboot)

- But iterating on `plasma-core22-desktop` changes…
- They often require regenerating the system image
- This is a much slower feedback loop

It Is All Immutable!

- This needs to be kept in mind
- Be strategic in what you can prioritise

- You can iterate quickly on
  - application snaps
  - `plasma-desktop-session` snap (requires logging out though)
  - even snapd (might require a reboot)

- But iterating on `plasma-core22-desktop` changes. . .
- They often require regenerating the system image
- This is a much slower feedback loop

Current Limitations

The Woes of Unmerged Snapd Changes

- As mentioned `systemd-user-control` is still in discussion
- This has unfortunate consequences. . .

- The official snapd doesn't have the interface
- This means we can only start the session with a temporary snapd fork

- The snap store assertions for `plasma-desktop-session` doesn't allow the `systemd-user-control` interface
- So we can't publish `plasma-desktop-session` on the store
- This means injecting a local build when making images

- If `plasma-desktop-session` doesn't come from the store, snapd won't auto-connect its interfaces
- This means manual connections are necessary for anything to start

- This isn't great for the user experience for now
- We hope this will get solved soon, making everything nicer to use

The Woes of Unmerged Snapd Changes

- As mentioned `systemd-user-control` is still in discussion
- This has unfortunate consequences...

- The official snapd doesn't have the interface
- This means we can only start the session with a temporary snapd fork

- The snap store assertions for `plasma-desktop-session` doesn't allow the `systemd-user-control` interface
- So we can't publish `plasma-desktop-session` on the store
- This means injecting a local build when making images

- If `plasma-desktop-session` doesn't come from the store, snapd won't auto-connect its interfaces
- This means manual connections are necessary for anything to start

- This isn't great for the user experience for now
- We hope this will get solved soon, making everything nicer to use

The Woes of Unmerged Snapd Changes

- As mentioned `systemd-user-control` is still in discussion
- This has unfortunate consequences. . .

- The official snapd doesn't have the interface
- This means we can only start the session with a temporary snapd fork

- The snap store assertions for `plasma-desktop-session` doesn't allow the `systemd-user-control` interface
- So we can't publish `plasma-desktop-session` on the store
- This means injecting a local build when making images

- If `plasma-desktop-session` doesn't come from the store, snapd won't auto-connect its interfaces
- This means manual connections are necessary for anything to start

- This isn't great for the user experience for now
- We hope this will get solved soon, making everything nicer to use

The Woes of Unmerged Snapd Changes

- As mentioned `systemd-user-control` is still in discussion
- This has unfortunate consequences. . .

- The official snapd doesn't have the interface
- This means we can only start the session with a temporary snapd fork

- The snap store assertions for `plasma-desktop-session` doesn't allow the `systemd-user-control` interface
- So we can't publish `plasma-desktop-session` on the store
- This means injecting a local build when making images

- If `plasma-desktop-session` doesn't come from the store, snapd won't auto-connect its interfaces
- This means manual connections are necessary for anything to start

- This isn't great for the user experience for now
- We hope this will get solved soon, making everything nicer to use

The Woes of Unmerged Snapd Changes

- As mentioned `systemd-user-control` is still in discussion
- This has unfortunate consequences. . .

- The official snapd doesn't have the interface
- This means we can only start the session with a temporary snapd fork

- The snap store assertions for `plasma-desktop-session` doesn't allow the `systemd-user-control` interface
- So we can't publish `plasma-desktop-session` on the store
- This means injecting a local build when making images

- If `plasma-desktop-session` doesn't come from the store, snapd won't auto-connect its interfaces
- This means manual connections are necessary for anything to start

- This isn't great for the user experience for now
- We hope this will get solved soon, making everything nicer to use

Upcoming Work

The Switch to Core 24

- Actually some preliminary work has been done
- This required a KDE Neon snapshot for Noble Numbat
- Shouldn't impact our architecture much
- That's a lot of components which will change
  - So a lot could go wrong...
- Will also allow a better approach for the provisioning
- We'll introduce the use of `provd` and a new wizard
- This should bring more configurability to the provisioning

The Switch to Core 24

- Actually some preliminary work has been done
- This required a KDE Neon snapshot for Noble Numbat

- Shouldn't impact our architecture much
- That's a lot of components which will change
  - So a lot could go wrong...

- Will also allow a better approach for the provisioning
- We'll introduce the use of `provd` and a new wizard
- This should bring more configurability to the provisioning

The Switch to Core 24

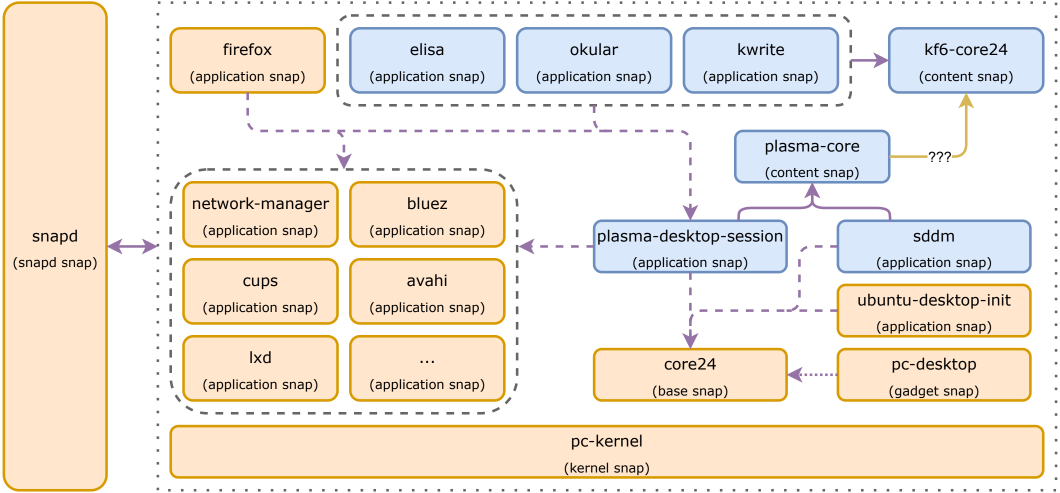
- Actually some preliminary work has been done
- This required a KDE Neon snapshot for Noble Numbat

- Shouldn't impact our architecture much
- That's a lot of components which will change
  - So a lot could go wrong...

- Will also allow a better approach for the provisioning
- We'll introduce the use of `provd` and a new wizard
- This should bring more configurability to the provisioning

More Modular Architecture

Warning: subject to changes!



- Current architecture was a good start but isn't ideal
  - Composability, components size and coupling to the base snap
- Time to attempt to decouple and to slice things further

Lessons Learned

Lessons Learned

- I never really liked packaging... that said, application snaps are easier to write
- The documentation is generally good and the recipes rather short
- The `kde-neon-6` snapcraft extension helps quite a bit

- The Ubuntu Core Desktop architecture being more in a state of flux, this is obviously less documented and tools have sharper edges

- The behavior enforced via the snap store can make things harder for development
- `systemd-user-control` being on hold prevents unlocking quite some of the potential for now

Lessons Learned

- I never really liked packaging... that said, application snaps are easier to write
- The documentation is generally good and the recipes rather short
- The `kde-neon-6` snapcraft extension helps quite a bit

- The Ubuntu Core Desktop architecture being more in a state of flux, this is obviously less documented and tools have sharper edges

- The behavior enforced via the snap store can make things harder for development
- `systemd-user-control` being on hold prevents unlocking quite some of the potential for now

Lessons Learned

- I never really liked packaging... that said, application snaps are easier to write
- The documentation is generally good and the recipes rather short
- The `kde-neon-6` snapcraft extension helps quite a bit

- The Ubuntu Core Desktop architecture being more in a state of flux, this is obviously less documented and tools have sharper edges

- The behavior enforced via the snap store can make things harder for development
- `systemd-user-control` being on hold prevents unlocking quite some of the potential for now

Lessons Learned cont'd

- We have good tooling to debug systemd related issues nowadays

- Double check and even triple check what is really confined
- Confining progressively makes things easier

- Avoid using `StartTransientUnit` in application code and dependencies
- Or provide a fork based alternative
- `KIO::KProcessRunner` fork based implementation has to be maintained
- We can't afford to deprecate it if we want to get serious at sandboxing

Lessons Learned cont'd

- We have good tooling to debug systemd related issues nowadays

- Double check and even triple check what is really confined
- Confining progressively makes things easier

- Avoid using `StartTransientUnit` in application code and dependencies
- Or provide a fork based alternative
- `KIO::KProcessRunner` fork based implementation has to be maintained
- We can't afford to deprecate it if we want to get serious at sandboxing

Lessons Learned cont'd

- We have good tooling to debug systemd related issues nowadays

- Double check and even triple check what is really confined
- Confining progressively makes things easier

- Avoid using `StartTransientUnit` in application code and dependencies
- Or provide a fork based alternative
- `KIO::KProcessRunner` fork based implementation has to be maintained
- We can't afford to deprecate it if we want to get serious at sandboxing

Where To Contribute?

- If you're interested here are the GitLab projects to monitor
  - https://invent.kde.org/neon/ubuntu-core
  - https://invent.kde.org/neon/snap-packaging
- We also have some documentation
  - https://community.kde.org/Guidelines_and_HOWTOs/Snap
  - https://community.kde.org/Neon/Core
- Come talk to us!

Acknowledgments

- Thanks to the enioka Haute Couture team
  - Benjamin Port
  - Antoine Gonzalez
  - Antoine Herlicq
- Thanks to the contractors we're working with
  - Scarlett Moore
  - Carlos De Maine
- Thanks to the gearheads who provided help on their spare time
  - Harald Sitter
  - David Edmundson
  - Ben Cooksley

Thank You!

Questions?

ervin@kde.org

kevin.ottens@enioka.com