KDE’s CI and CD Infrastructure

Ben Cooksley
Hannah von Reth
Julius Künzel
Volker Krause

Continuous Integration

- Every repository should have this!
- Compiles and tests your code on every change made
- Done in a reproducible, clean environment
- Not limited to C++ code:
  - JSON/XML/Yaml validators
  - Python linter
  - REUSE license validator

Setting up CI

- Configured using a .gitlab-ci.yml file at the top level of your repository
  - Where possible include existing templates rather than rolling your own
  - If you do need to do something special though you can create a custom job

- Currently supports building:
  - Linux/glibc and Linux/musl
  - FreeBSD
  - Windows
  - Android

.gitlab-ci.yml 1.47 KiB

```yaml
include:
  - project: sysadmin/ci-utilities
    file:
      - /gitlab-templates/json-validation.yml
      - /gitlab-templates/yaml-lint.yml
```

Project Dependencies

.kde-ci.yml 325 B

```yaml
Dependencies:
- 'on': ['@all']
  'require':
    'frameworks/extra-cmake-modules': '@same'

- 'on': ['Linux', 'FreeBSD']
  'require':
    'libraries/plasma-wayland-protocols': '@latest'
```

- Managed in two different ways depending on whether it is another KDE project (internal) or not (external)
- List internal dependencies in .kde-ci.yml
- External dependencies are provided by CI images:
  - For Linux/FreeBSD: uses system packages
  - For Windows/Android: uses Craft
- Bulk rebuilds of dependencies are done using seed jobs

Configuring CI Jobs

- Handled using a .kde-ci.yml file in your repository if you are using the standard CI templates
- Provides lots of different options to allow customising workflows:
  - Mandatory passing tests
  - Custom build options
- If you need more, see sysadmin/ci-utilities for the default settings and the scripts driving this

global.yml 729 B

```yaml
Dependencies: {}

RuntimeDependencies: {}

Environment:
  KDECI_BUILD: "TRUE"

Options:
  in-source-build: False
  use-ccache: False
  ccache-large-cache: False
  cmake-options: ''
  release-build: False
  test-before-installing: False
  run-tests: True
  tests-load-sensitive: False
  tests-run-in-parallel: False
  per-test-timeout: 60
  setup-x-environment: True
  setup-dbus-session: True
  force-inject-asan: False
  ctest-arguments: ''
  require-passing-tests-on: []
  run-cppcheck: True
```

Continuous Delivery

- Every application should have this!
- Builds fully functional application packages

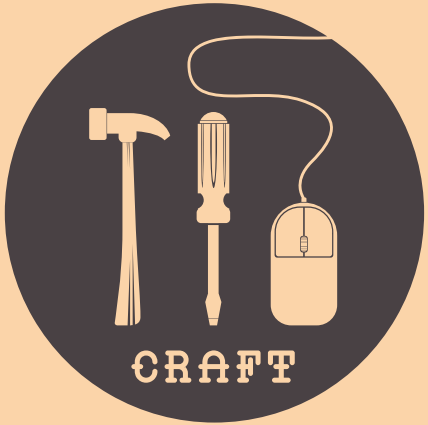
- Supports Linux, Windows, Android, macOS
- Signing, store submission

- Set up: include template in .gitlab-ci.yml
- Signing/publishing needs to be enabled in sysadmin/ci-utilities > signing

CD: Application Metadata

- Driven by Appstream files in each repository
- Automatically used for:
  - apps.kde.org
  - Flathub
  - Google (Play), F-Droid and Microsoft Store information

CD: Craft



- Drives builds for everything but Flatpak/Snap
- Basically a distro supporting Windows, Linux and macOS
  - Prebuild cache
  - Release with debug info
- Each app and each dependency needs a Craft blueprint: packaging/craft-blueprints-kde
  - Describes building and packaging
  - No changes to the CMakeLists.txt needed
  - Full deployment, not just the app
- Customize via .craft.ini and .craftignore
  - Never set the version for KDE frameworks
- Resulting CD artifacts **must** be released

CD: Linux Appimage

- Runs on all *recent- versions of Linux (libc)
- Simple to create and use
- Usually contains everything the app needs but system libs

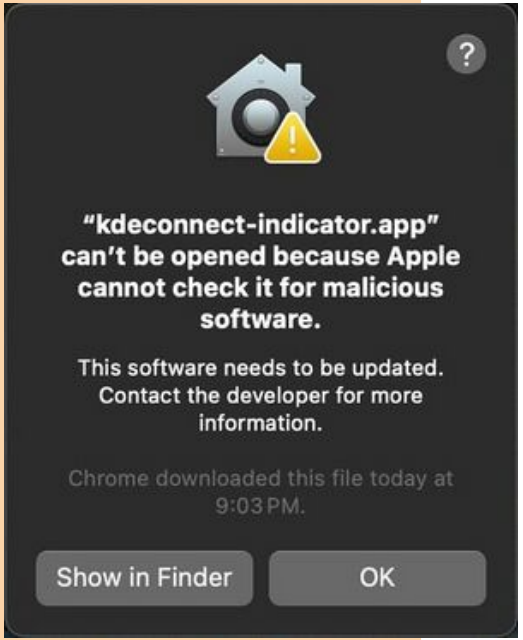
CD: Windows

- Windows Store (opt in)
  - Releases are automatically prepared
  - Release is done by a maintainer
  - Sideload version for testing *-sideload.appx
- NSIS
  - Classic Windows installer *.exe
- Portable archive
  - Plain archive, runable everywhere *.7z
- Only releases are signed

CD: Android

- Builds APK and AAB packages
- On master and release branches automatically signed and uploaded to KDE's F-Droid stable and "nightly" repositories
- Upload to Google Play is also available, publishing needs manual steps still though

CD: macOS



- Two architectures: ARM and x86_64
- Builds DMG bundles
- Needs a plist file (can be generated by CMake)
- Notarization for better user experience

CD: Flatpak

- Manifest file in each repo: .flatpak-manifest.json
- Only for nightlies! (stable manifest are on github.com/flathub)
- Describes building the app and all dependencies on top of KDE Frameworks
- Result installable by opening flatpakref from https://cdn.kde.org/flatpak/ with Discover etc.

CD: Snap

- See Kévin’s talk later today in Room 1

Other things

- The CD side is also capable of many other things including:
  - Publishing packages to PyPi
  - Deploying websites to KDE.org infrastructure

- While on the CI side we can take energy consumption measurements
  - Separate BoF for this - Monday 10:00 Room 2

Wishlist & Outlook

- GUI tests on Windows, unit tests on Android
- Automated testing of CD packages
- Status overview/dashboard
- Publishing to Flathub

Conclusion

- CI/CD got a lot more accessible for everyone with the move from Jenkins to Gitlab
- Most things can be done via an MR or even in your own repository directly!
- BoF: Tuesday 15:00 Room 3