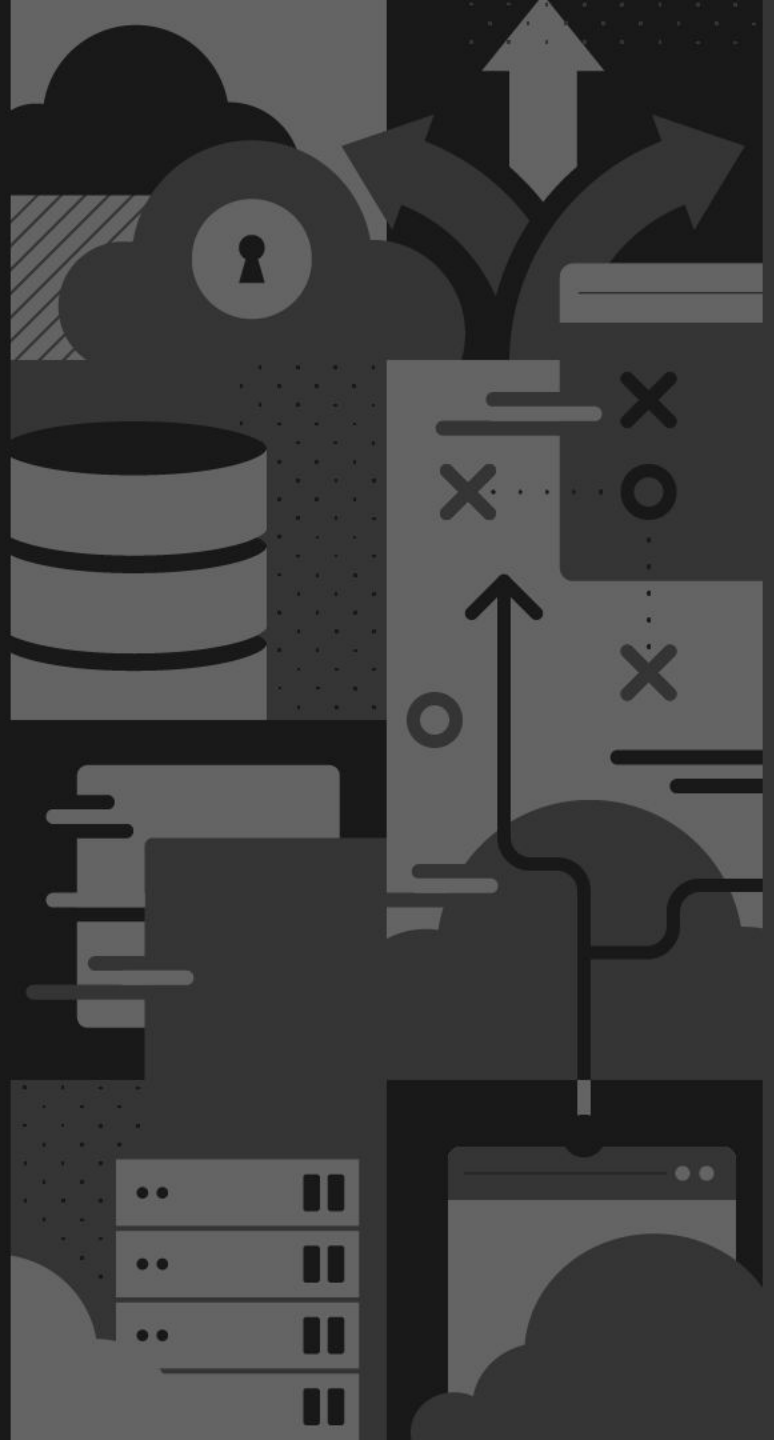


Containers Monorepo

Jan Kaluza, Red Hat Containers Team

podman
the daemonless container manager

Motivation

- To improve the release process, development workflow and reduce the vendoring issues, investigate the option to merge all the container repos (**polyrepos**) into one repo (**monorepo**).
- Goal: Investigate the monorepo approach for our three repositories:

  https://github.com/containers/common/

  https://github.com/containers/image

  https://github.com/containers/storage

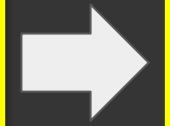
What is polyrepo?

- Single code repository with single project.

```
common.git/
├── pkg/              # The common source-code.
├── tests/            # Tests for common source-code.
├── types/            # Shared common types
└── go.mod            # go.mod for "common" package.
```

```
image.git/
├── pkg/              # The "image" source-code.
├── tests/            # Tests for "image" source-code.
└── go.mod            # go.mod for "image" package.
```

```
storage.git/
├── pkg/              # The "storage" source-code.
├── tests/            # Tests for "storage" source.
└── go.mod            # go.mod for "storage" package.
```



What is Monorepo?

- Single code repository with multiple projects.

```
monorepo.git/
├── pkg/              # The common source-code.
├── tests/            # Tests for common source-code.
├── types/            # Shared common types
├── image/            # Independent "image" Go package
│   ├── ...
│   └── go.mod        # go.mod for "image" package
├── storage/          # Independent Go package
│   ├── ...
│   └── go.mod        # go.mod for "storage" package
└── go.work           # Ties everything together
```


podman
the daemonless container manager

Is monorepo better?

- No simple answer. **Let's imagine we decide to switch to monorepo.**
- There are following topics to cover:
  - ? Migration
  - ? Importing go modules
  - ? Contributions
  - ? Releases
  - ? Builds and CI
  - ? Code Ownership
  - ? Issue tracking

podman
the daemonless container manager

Monorepo: Migration

✅ There are existing tools and processes to migrate multiple polyrepos to single monorepo, preserving the git-history.

- https://github.com/newren/git-filter-repo

❌ No real blockers.

Monorepo: Importing go modules

✅ Go modules can be imported from monorepo the same way as from polyrepo.

❓ Need to change all the imports to new monorepo URL.

❌ No real blockers.

podman

the daemonless container manager

Monorepo: Contributions

- ✅ Easy to discover the right repository to edit – there's just one.
- ✅ Single, atomic, PR for any change.
  - Easier to review and understand the "big picture".
- ✅ No need to constantly update the referenced versions of other modules.
- ✅ All the modules use the same versions of dependencies and are tested together.
  - Easier vendoring.
- ❓ Harder to list PRs just for a single module.
  - Needs CODEOWNERS and/or labeler to assign PRs based on the paths changed.
  - "History" in module's directory for "merged".
- ❌ No real blockers.

podman
the daemonless container manager

Monorepo: Releases

✅ Each module can still be (and probably should be) released independently.

✅ Or releases can be atomic, releasing all the modules together.

✅ Much easier vendoring.

❓ Releasing each module in monorepo separately needs per-project tags.

- v5.35.0 -> storage-v5.35.0.

❓ With PRs touching multiple modules, forward/backward compatible changes are less intuitive than with polyrepo.

❓ We have to change the release process, but hopefully make it easier.

❌ No real blockers.

podman

the daemonless container manager

Monorepo: Builds and CI

✅ *With the right tooling*, all modules can be tested together, preventing incompatible changes to land into repository.

❓ Needs smart builds and CI system, otherwise we build and test modules unrelated to PR. This "**could**" increase the test run-time dramatically without any benefit.

- Bazel, Pants, Earthly?

❓ Or set of custom pipelines tailored to podman.

- But who will maintain these?

❌ Fedora does not include any popular build system which would support monorepos.

- We could use plain "go build" locally and something else for PRs, but should we?

podman
the daemonless container manager

Monorepo: Code Ownership

✅ No benefit?

❓ Github permissions are built around polyrepos.

❓ We can switch to per-directory CODEOWNERS if we need acks from different team members per module.

❌ No real blocker as long as all the polyrepos have the same maintainers.

Monorepo: Issue tracking

✅ Easier for user to report problems - just single tracker.

❓ All issues are in the single issue tracker.

- Need to distinguish between module using labels and triage them - extra work which was not needed previously.

❌ No real blocker.

Monorepo: Summary

- ✅ Migration
- ✅ Importing go modules (one time change of URLs needed)
- ✅ Contributions (Easier to contribute, harder to browse PRs, CODEOWNERS).
- ✅ Releases (Easier vendoring, but needs release process change and thinking about “tags”)
- ❓ Builds and CI (Will the tests scale? ❌ Fedora vs. smart build systems)
- ✅ Code Ownership (fine if the whole team can edit all the modules)
- ❓ Issue tracking (Issues mixed together, labels to distinguish them)

What to do next?

- Consider hybrid model.
  - Use monorepo just for some modules like storage, image and common.
- Merge them into newly created production-ready monorepo (one week).
  - Use go.work as described in "Investigate monorepo" document.
  - Setup CODEOWNERS and labeler if needed.
  - This is task should be doable in a few days.
- Implement the Github CI workflow in a naive way where all the tests are executed for any change to see the real impact on contributors (one week).
  - Hard to predict to me, because I have no admin experience with cirrus-ci. What permissions are needed? Where to get them?
  - The Github workflow part is not hard.
- Explore how the "Release process" would change.
- Evaluate.

podman
the daemonless container manager