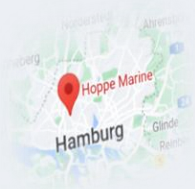
Sailing smoothly

Navigating a migration from **multi-repo** to **monorepo**

Hamburg Python Pizza

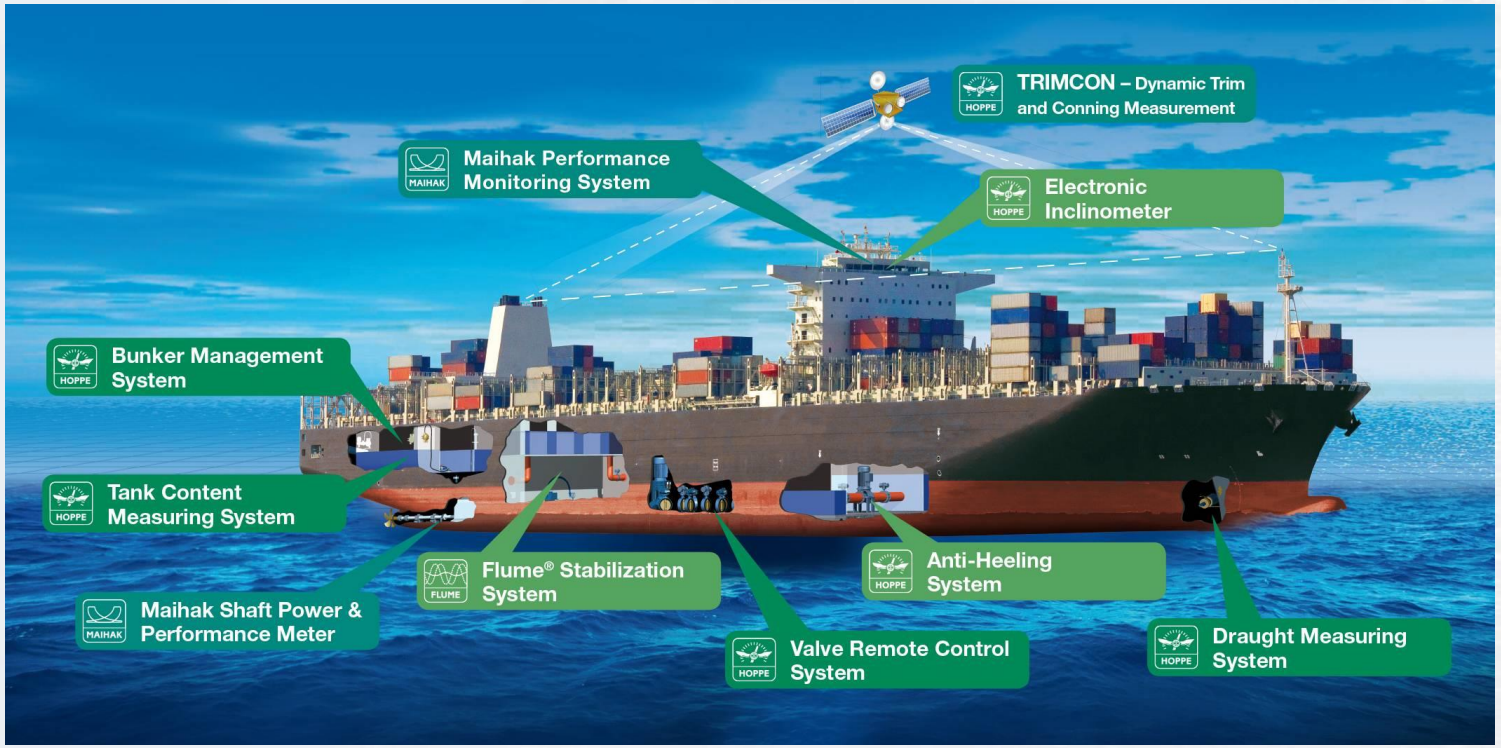
17.11.2023

Julio Batista Silva



Hoppe Marine

We develop fluid control and measurement systems for the maritime industry



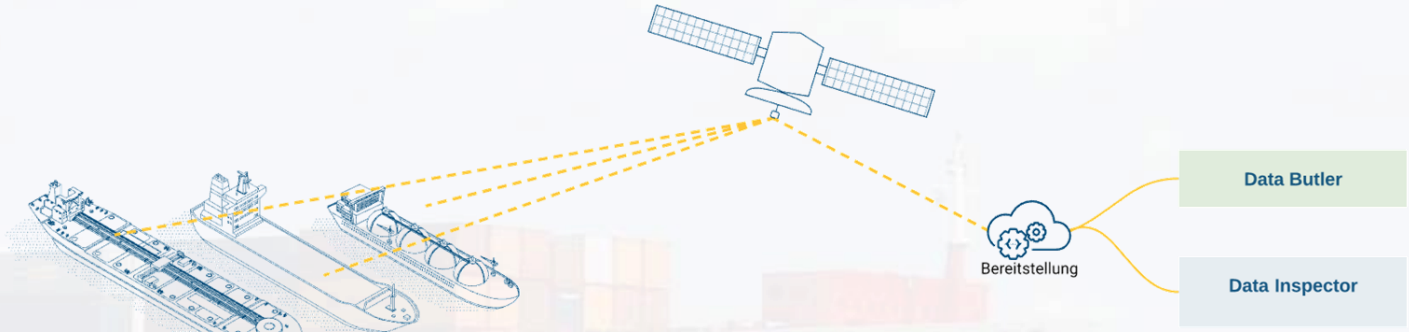
Approximately one of every eight new ships is equipped with at least one of our systems [2]

8000+ vessels

They generate a lot of data!

APIs and Pipelines

- This data must be
  - Cleaned
  - Validated
  - Aggregated
  - Displayed on our web interface
  - Made available to customers



- Transformations and analysis are done with **Pandas**, **scikit-learn**, etc.
- APIs are built with **FastAPI** and deployed as **AWS Lambda functions**

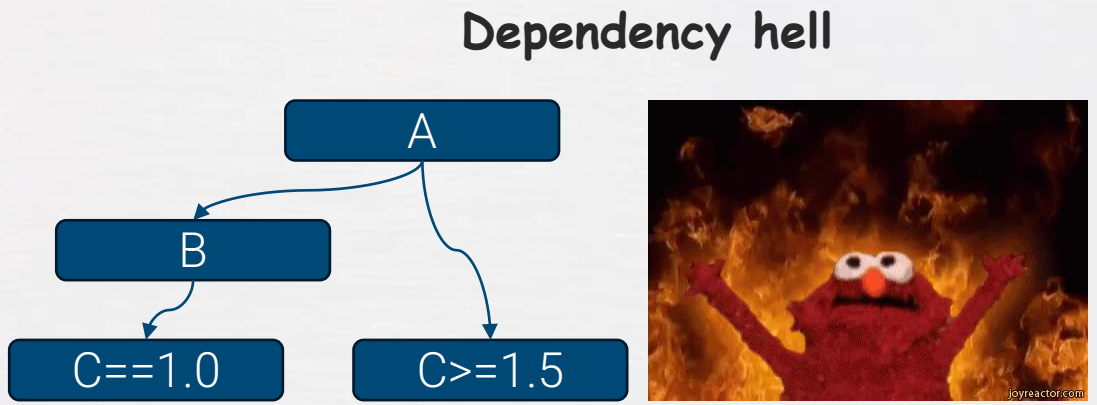
Each in its own repo…

Where things got hard

- Having separated repos made sense in the beginning but the codebase grew
- As of 3 months ago, our 4-people team alone was responsible for **56 repos**

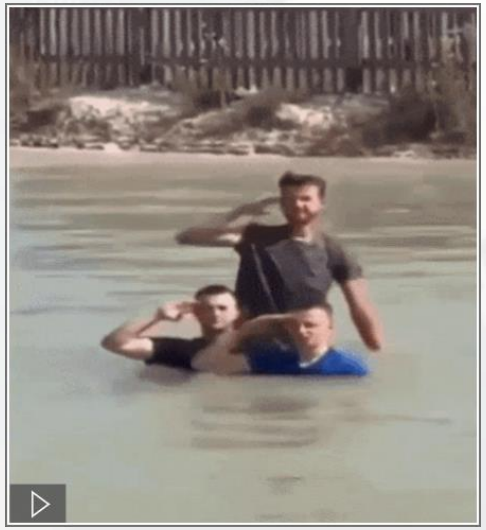
- APIs and pipelines are independent, but have common (internal and external) dependencies

- Things got hard
  - Hard to test
  - Hard to keep everything updated
  - Hard to keep consistency
  - Hard to synchronize dependencies



Example:
Change a function in one shared library

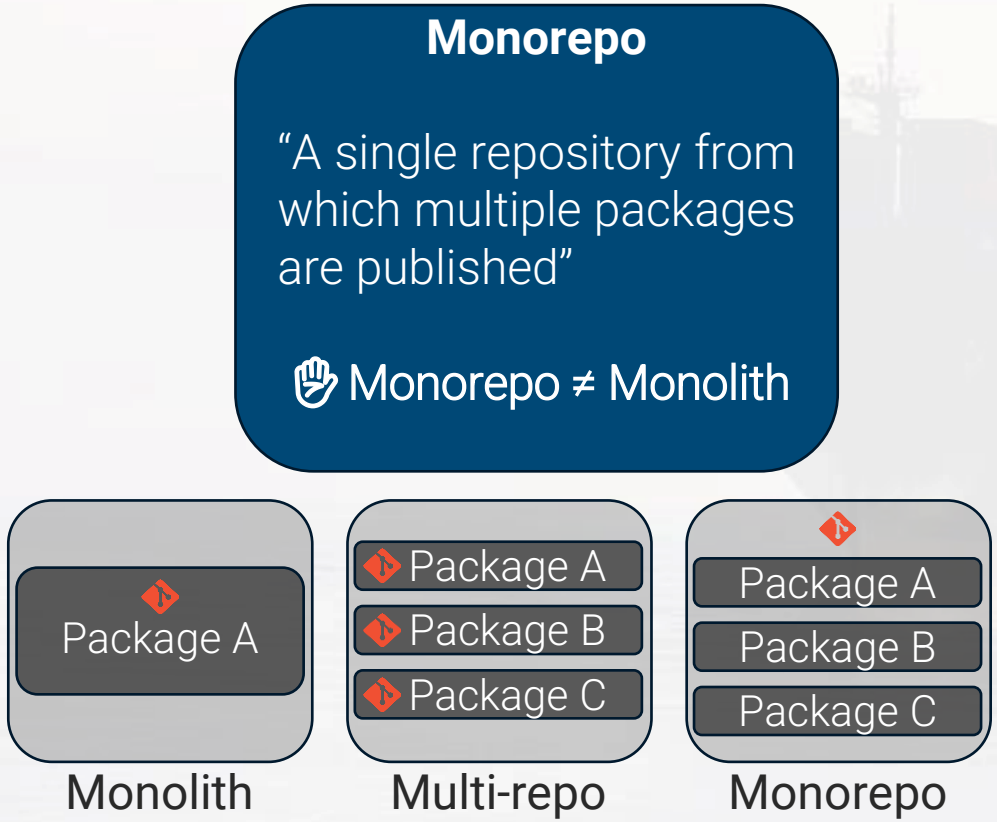
1. Clone the repository where the function is defined
2. Create a virtual environment
3. Change the function
4. Run tests
5. Open a PR
6. Publish a new version of the library to our package repository
7. Find, in all repositories, where this function is used
8. Clone those repositories
9. Create a virtual environment for each
10. Update the dependency
11. Change the function call there (if required)
12. Run tests
13. Open a PR
14. Repeat recursively



Bad not just because it’s a lot of work

Solution

Feature freeze. Time for refactoring



Before migrating to the monorepo:

- We wrote more tests
- Defined a well-organized repo structure
- Defined **code standards** and tooling
  - Poetry for dependency management
  - Ruff for sorting imports, formatting and linting
  - Pytest for unit tests
  - Docker for reproducibility
  - Pre-commit hooks
- Opted for **trunk-based development**
  (With short-lived branches)

What about deployment?

- This is the biggest challenge, in my opinion
- The old CI/CD pipelines ran all tests and deployed everything in the repo
- How to test and deploy only what is affected by your changes?
  - Custom scripts that parse git diff-tree 👎
  - CI tool specific configuration 👎
- There are better tools! 😀 😀 😀
  (But they are not always trivial to setup 😭 😭 😭)

Tooling

Bazel

Gradle

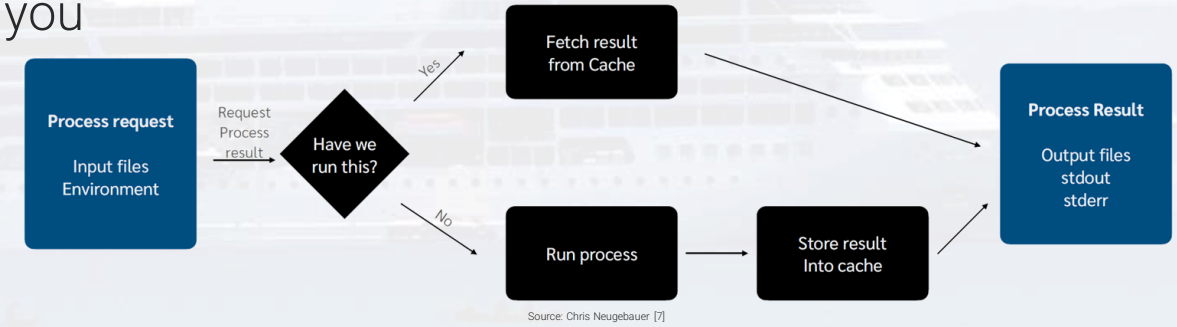
Lerna

TURBOREPO

Pants 2

- Feature rich build system
- Developed with Python use cases in mind (also supports Go, Java, Scala, Kotlin, Shell, Docker, and more)
- Infers a lot about your repo via static analysis
- Fast
  - Caching (also remote)
  - Concurrency
- Extendable
- Works with goals. You write what you want to achieve, and Pants orchestrates the execution for you

```
pants test apis/api_1
pants package pipelines/pipeline_2/**
pants lint fmt --changed-since=HEAD
```



Source: Chris Neugebauer [7]

Pros and Cons

Monorepo

Pros

- Single environment
- Visibility
- Uniformity
- Easier to test changes in shared libraries
- No need for a private Python package repository
- Atomic PRs ⇒ Easier code review
- Common version of dependencies

Cons

- Requires dedicated tooling to be practical
- Large repository
- PRs accumulate (and you get notified a lot* 🔔)
- Large commit history *
- Git history is lost during migration

Multi-repo / Polyrepo

Pros

- Simpler to develop if components are independent
- Simpler CD pipeline
- Easier to grant fine-grained access*
- No risk of triggering a large CI build for small changes

Cons

- Harder to keep track of changes across repositories
- Prone to inconsistencies between repos
- Harder to debug and test while refactoring

* There are tools to deal with that

Lessons Learned and Hints

- Monorepos are not a silver bullet
- They bring benefits, but also can also create new problems. Weight the pros and cons carefully
- Migrating many repositories can be a lot of work
- Dedicated tools exist, but they are not always trivial to setup and there will be a learning curve to use
- Pants can be adopted incrementally
- You can have a hybrid-repo. Not all the codebase has to be migrated
- Organization is key
- You should still avoid tight coupling between components
- For AWS Lambdas, shared dependencies can be deployed as **lambda layers**

References

1. https://www.hoppe-marine.com
2. https://www.marinelink.com/news/solutions-shipping414299
3. https://monorepo.tools
4. https://sdtimes.com/softwaredev/the-monorepo-approach-to-code-management
5. https://www.tweag.io/blog/2023-04-04-python-monorepo-1/
6. https://www.pantsbuild.org/
7. "Hermetic Environments in Pantsbuild" aka "Fast and Reproducible Tests, Packaging, and Deploys with Pantsbuild“ by Chris Neugebauer at Pycon 2022
8. "Python monorepos: what, why and how“ by Benjy Weinberger at EuroPython 2021

Thank You