Monorepos

Working with a single, big, scary version control repository

Benjamin Eberlei

August 23, 2015

Me

passion for software quality

We promote high quality code with trainings and consulting

`http://qafoo.com`

- Doctrine Project Team-Lead
- Blog `www.whitewashing.de`
- Tweeting `@beberlei`

Tideways

Profiling, Performance Monitoring, Error Tracking for PHP

`https://tideways.io`

What is a Monolithic Repository (monorepo)?

A **single** version control repository containing multiple

- projects
- applications
- libraries,

often using a common build system.

History of Version Control

Before Git/Mercurial we all used Subversion and monorepos where widespread.

Example

Apache Project has code from **all** projects in one SVN repository.

Example

Linux Kernel contains support for tons of different drivers in one central repository.

Monorepos in the Wild

- Google use monorepo for their whole codebase
- Facebook merge their three monorepo into one scary monorepo
- Twitter migrated from many repositories to monorepo
- Etsy
- many more...

passion for software quality

Monolithic Applications?

Code in monolithic repositories is **not** automatically monolithic itself!

Components

- A component is a library, application, ...
- Organize components in useful directory hierachy
- A single component **can** be monolithic
- ..or a ”micro-service/library”
- Components have dependencies to each other
  - by directly sharing code
  - by calling external APIs (REST, RPC)

A Common Build System

High level build-system that standardizes continouos integration stages

- Bazel/Blaze by Google
- Buck by Facebook
- Pants by Twitter
- We use Ant

Reasons Why Monorepos Are Awesome

- Discoverability
- Refactorings
- Pull-Requests
- Code-Reuse
- Dependencies
- Testing and CI/CD
- Productivity

High Discoverability For Developers

- Developers can read and explore the whole codebase
- `grep`, IDEs and other tools can search the whole codebase
- IDEs can offer auto-completion for the whole codebase
- Code Browsers can links between all artifacts in the codebase

Code-Reuse is cheap

Almost zero cost in introducing a new library

- Extract library code into a new directory/component
- Use library in other components
- Profit!

Refactorings in one commit

Allow large scale refactorings with one single, atomic, history-preserving commit

- Extract Library/Component
- Rename Functions/Methods/Components
- Housekeeping (phpcs-fixer, Namespacing, ...)

Pull Requests in one commit

- Simplify changes that affects multiple layers and components
- Allows to revert them

No Dependency/Version Mess

- No need for versioning in components
- No need to manage dependencies and versions
- Avoids multi-pull-request merging in required order
- Avoids forgetting to update composer/git submodule
- Forces dependees to update to newest version

No Repository-Access Micromanagement

- Every developer can always read the whole code
- Avoids cost of micromanaging repository access.
- Restricting commit access possible, but bad idea

Efficient Testing and Continuous Integration/Delivery

- Single repository makes dependency resolving easy
- Only run the tests for components that changed
- But also run the tests for all dependencies, recursively
- Avoid manual test micromanagement

Increased Developer Productivity

- New developers can get started more easily
- All developers can move between different projects
- Avoids information silos

Gregory Szorc On Monolithic Repositories

Monolithic repositories are ... compatible with the ebb and flow of ... large software projects. Components, features, products, and teams come and go, merge and split. ***The only constant is change****.*

”Downsides”?

Require Collective Responsibility for Team and Developers

Require Trunk-Based Development

Force you to have only one version of everything

Scalability Requirements for the Repository

And Then came Git...

- We unlearned monorepos
- NPM and Heroku force deployable-unit=one-repository pattern
- Many package managers followed suite (Composer..)
- Lead to the ”Microlibrary” movement

How to fix tooling for monorepos?

- Project Structure
- Composer
- Git
- Builds
- CI Systems

Example Project Structure

```
golang/src/tideways/cli
golang/src/tideways/collector
golang/src/tideways/daemon
golang/src/tideways/sql
golang/src/tideways/xhprof
components/automation
components/chrome-extension
components/landingpage
components/php-extension
components/php-library
components/profiler
components/profiler-ui
components/replicator
playground/*
```

Qafoo
passion for software quality

Multiple composer.json and monorepos

- Cause vendor folder duplications (slow)
- Autoloading must be tricked into loading other monorepo dependencies.
- Different dependencies for each component cause crashes

Relative Autoloading

```
# components/Foo/composer.json
{
        "autoload": {
                "psr-0": {"Foo": "src/"}
        }
}
```

```
# components/Bar/composer.json
{
        "autoload": {
                "psr-0": {
                        "Foo": "../Foo/src/"
                        "Bar": "src/"
                }
        }
}
```

passion for software quality

Dependency Clash

```
# components/Foo/composer.json
{
    "require": "symfony/http-foundation": "~3.0"
}
```

```
# components/Bar/composer.json
{
    "require": "symfony/http-foundation": "2.4.*"
}
```

Dependency Clash: Use both Foo and Bar?

```
# components/Baz/composer.json
{
    "autoload": {
        "psr-0": {
            "Foo": "../Foo/src",
            "Bar": "../Bar/src"
        }
    },
    "require": {
        "symfony/http-foundation": "~3.0"
    }
}
```

One composer.json and monorepo

- Use the same `vendor/autoload.php` in all components
- Or use Fiddler to build custom autoloaders for each component

Fiddler

- monorepo support on top of Composer
- `https://github.com/beberlei/fiddler`

Fiddler Concepts

- Fiddler Package Names are directory names
- One global composer.json
- One version of every dependency onl
- Each component with a fiddler.json

Fiddler: Global composer.json

```
# ./composer.json
{
        "require": {
                "symfony/http-foundation": "~3.0",
                "doctrine/dbal": "~2.5"
        }
}
```

Fiddler: Component with Vendor Dependency

```
# components/Foo/fiddler.json
{
    "autoload": {
        "psr-0": {"Foo": "src"}
    },
    "require": [
        "vendor/symfony/http-foundation"
    ]
}
```

Fiddler: Component

```
# components/Bar/fiddler.json
{
    "autoload": {
        "psr-0": {"Bar": "src"}
    },
    "require": [
        "components/Foo",
        "vendor/doctrine/dbal"
    ]
}
```

passion for software quality

Git as a Deployment Tool

- Git is used for Deployment
  - Heroku-Style PaaS
  - Pull on Production
- (force) pushing subtrees/build results to deplyo repository
- Plans for fiddler to repackage component as tarball

Build and CI Tools

passion for software quality

THANK YOU

Rent a quality expert
qafoo.com