Onebrain

Microprojects for data science

Dan Miller, Alex Deng, Narek Amirbekian, Navin Sivanandam, and Rodolfo Carboni
Airbnb

Introduction

Scaling and reproducing scientific work is difficult

Challenges

- Copy/pasted code
- Monorepo vs. microrepo?
- R and Python?
- Dependency hell

Ecosystem of solutions, need for single coordinating layer … enter Onebrain

Wrap {analysis, training, evaluation, …} ↦ microprojects

- CI/CD
- Configuration/dependency management,
- Production platforms and other DS' code can easily reference consistent API wrappers

Scaling usage

Growth within a year

180

Weekly active users

... from 5

20k traces / day

495

Total users

... from 5

214

Distinct projects

... from 3

Python/R libraries
web apps
command line tools

84

Internal web apps

2.5k users
collectively

Scientific work in enterprise

Different from both software for academic research and enterprise software engineering

Academic Research	Software Engineering
Generally individuals working independently	Teams working in parallel
Custom-built software	Reusable components
Work must be eventually reproducible	Work subject to extensive testing before deployment
Success uncertain	Generally timeline to success is known
Pushing the frontier of what is known	Leveraging tried and true technology

Scientific work in enterprise

Different from both software for academic research and enterprise software engineering

Science for Enterprise

Serially independent, but projects change ownership frequently

Heavy customization of reusable components

Reproducible at a "click of a button"

Success uncertain, but intermediate stages of work need to be shareable

Using state of art technology, many different languages and software

Scientific work in enterprise

Existing ways of working and tooling are not optimized for the problem of 'Science at Scale'

Scaling innovation	⇒	Scientists need to be able to leverage many different languages and software packages in order to rapidly innovate. **How can we do that with stable, highly quality code?**
Multi-use deployments	⇒	The output of scientific work can be anything from a prototype to a production algorithm. **Can our development environment support both?**
Reproducibility and reusability	⇒	We need code modularization and reuse from SWE, while also supporting scientific reproducibility and review. **How do we support science best practices?**

Onebrain

A coding standard

People write their code into an arbitrary directory structure, with a machine-readable (YAML) configuration file at its root

Metadata

Entry points

Environment

```yaml
name: my-cool-api
version: 0.0.2
namespace: prototypes
description: I'm making a cool API
tag: [llm]
authors:
  - Jane Doe <jane.doe@airbnb.com>

build_enabled: true

entry_points:
  main:
    type: shell
    command: uvicorn app.main:app --host 0.0.0.0 --port {{ port }}
    parameters:
      port:
        type: int
        default: 8880

env:
  python:
    pip:
      fastapi: ==0.100.0
      # https://github.com/tiangolo/fastapi/discussions/9709#discussioncomment-6396226
      pydantic: ==1.10.11
      uvicorn[standard]: ==0.22.0
```

Onebrain

A coding standard

Beyond project metadata, the configuration file specifies everything needed to execute the code - from the hardware requirements and library dependencies, to the commands for code execution

```
entry_points:
  query_historical_data:
      type: bash
      command: python src/input_data.py
      parameters:
        ds:
          type: str
          help: Starting date (Format 2022-01-01)
  fit_models:
          type: bash
          ...
```

Onebrain

A CLI

User interaction with Onebrain happens through a CLI

```
> brain generate
  ...(initializes a project skeleton)

> brain setup-deps
  ...(install required code libraries)

> brain run -e fit-models
  ...(executes project code)

> brain –help
  ...(help on 20+ commands available)
```

Onebrain

A template repository

Onebrain's back-end architecture enables seamless publication of code, and

```
# onebrain.yml
build_enabled: true
```

And anyone can use an existing project as a template for their own work

```
> brain generate --project existing-project-name
```

Or generate a one-click URL that anyone can follow to spin up a server and run your project

```
> brain generate-url
```

And because it's "just code", Onebrain projects can be checked into a version-controlled repo, and any repo can be a Onebrain repo

Onebrain

Code library publisher

Library producers can write their code as a Onebrain project, and simply pushing it to a Onebrain repo will automatically publish it into a hosted package repository.

From there, anyone else can install it with standard package managers for use in their own code, or seamlessly reference it within other Onebrain projects.

```
# Python
pip install air-forecaster --extra-index-url https://airbnb.ch/onebrain

# R
remotes::install_url('https://airbnb.ch/onebrain/air-forecaster')
```

Onebrain

Interactive web app publisher

Onebrain projects can accommodate application code (e.g. Streamlit, Shiny) to be hosted as a web page.

Because Onebrain is already a replicable-code environment, the code can be checked into an application repo that will automatically launch the app into a hosted server

Integrated with idiomatic auth, hosting, networking

Onebrain

And many more things

Tools for

- **Remote Server Configuration** - start/stop remote server instances, Jupyter kernels, Google Colab
- **Automatic IDE Configuration for Remote Work** - behind-the-scenes SSH/remote drive configuration
- **Remote File Synching** - local/remote directory mapping and rsync-type synchronization