Measuring notebook reproducibility with repo2docker

Motivation

- A large-scale study of notebook reproducibility was published in 2019 [1]
- repo2docker is a tool for automatically generating environments from repos, to aid in reproducibility [2]
- we want to evaluate repo2docker's automatic env generation, and compare with the study's conclusions
- Study did not evaluate execution of languages other than Python; repo2docker can produce environments with Julia, R, etc.

We built `https://github.com/minrk/repo2docker-checker` to automate fetching and building repos with repo2docker, then running a test script to classify whether it should be considered reproducible

Caveats

What qualifies as reproducible?
We chose: - repo2docker successfully builds an image, and - it has notebooks - notebooks execute without error

Background

- reproduce previous study
- compare with repo2docker
- evaluate "automate existing best practices" claim
- failure to reproduce != repo2docker failure!

Process

- collect repositories to test
  - source 1: github search for language:jupyter
  - source 2: queries from [1]
- build image with repo2docker
- find notebooks in repo (up to 5)
- execute notebooks with nbconvert

Categories

Based on queries from previous study database, we collected and tested samples of repos, grouped by classification in the study:

- Julia notebooks (not tested previously)
- R notebooks (not tested previously)
- Installation failed (environment specified, could not install)
- Import error / ModuleNotFound (likely missing dependency specification)
- Success without dependencies (environment not found by study, execution succeeds)
- Success with default dependencies (study ran with anaconda, success)

Outcomes

Several possible classes of failure, some of which are interesting for repo2docker, others are not:

- build failure
  - due to partially pinned environment (e.g. pinned numpy not compatible with Python 3.8)
  - due to pre-build dependencies (e.g. apt-get install C libraries)
  - actually invalid requirements
- execution failure
  - environment not specified at all
  - environment insufficiently specified
  - user input required (e.g. credentials for using an API, or "fill out X before executing")

Results

Right	Left	Default	Center
12	12	12	12
123	123	123	123
1	1	1	1

Table: Demonstration of pipe table syntax.

Conclusions

- A large fraction of build failures were due to pinning packages (most often numpy or pandas) but not Python, because there is no community standard for pinning Python (conda, Pipfile insufficiently common, often lack Python itself)
- repo2docker success rate could be greatly improved if we looked at more of the repo to determine runtime language and/or version:
  - last repository commit date could pick older Python by default
  - notebook `language info` metadata includes runtime and version
- repo2docker does not implement best practices adopted in R and Julia communities
  - big caveat for Julia: most repos tested were older than the advent of the new `project.toml`

Future work

- Test repo2docker with different strategies for picking runtimes / versions
- Evaluate more repos, our sample size was small
- Look at more recent R / Julia repos for community standards for specifying environments

Bibliography

[1] João Felipe, Leonardo, Vanessa, and Juliana. Dataset of A Large-scale Study about Quality and Reproducibility of Jupyter Notebooks, March 2019. URL `https://doi.org/10.5281/zenodo.2592524`.

[2] Project Jupyter, Matthias Bussonnier, Jessica Forde, Jeremy Freeman, Brian Granger, Tim Head, Chris Holdgraf, Kyle Kelley, Gladys Nalvarte, Andrew Osheroff, M Pacer, Yuvi Panda, Fernando Perez, Benjamin Ragan Kelley, and Carol Willing. Binder 2.0 - Reproducible, interactive, sharable environments for science at scale. In Fatih Akici, David Lippa, Dillon Niederhut, and M Pacer, editors, *Proceedings of the 17th Python in Science Conference*, pages 113 – 120, 2018. doi: 10.25080/Majora-4af1f417-011.

Vilde Dille Øvreeide and Min Ragan-Kelley

Simula Research Laboratory

simula