FACTORY PRODUCED CI/CD PIPELINES

NIALL BYRNE

ABOUT ME:

- TORONTO BASED DEVELOPER

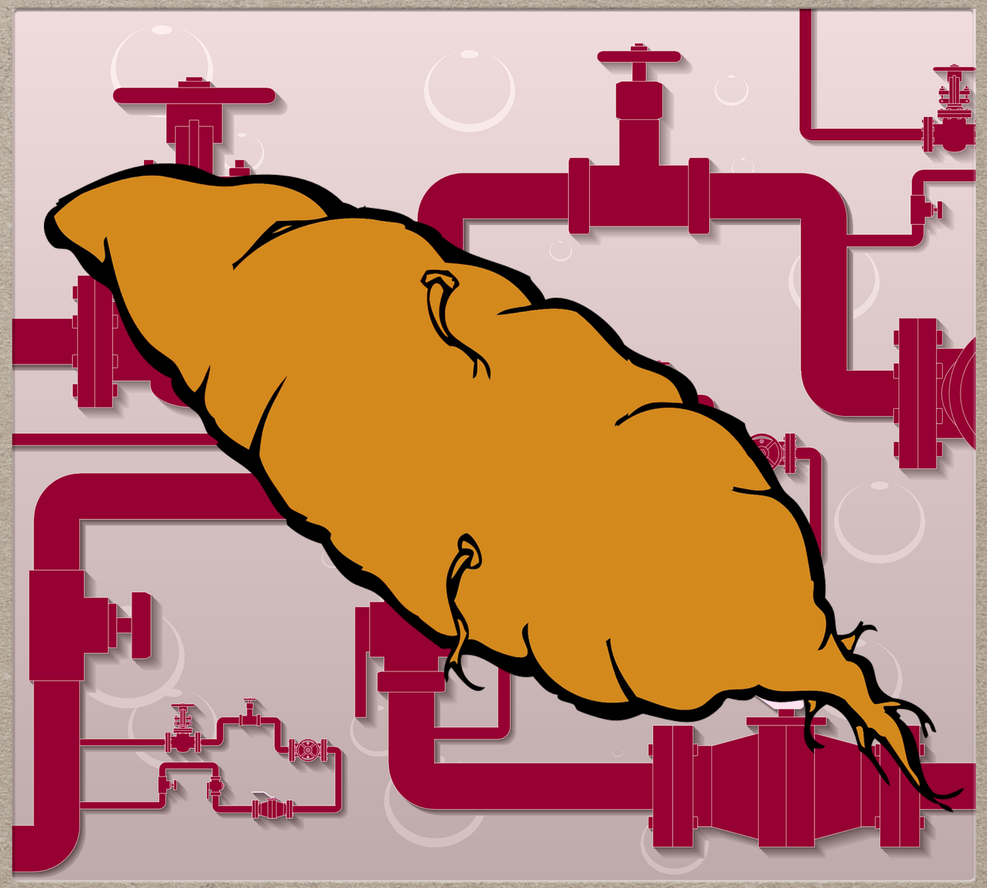
- SPECIALIZE PYTHON AND CLOUD SYSTEMS

- OCCASIONAL SPEAKER, FREQUENT PIPELINE TESTER

- RECENT JAVASCRIPT CONVERT (WHICH IS PART OF TODAY'S STORY...)

PIPELINES AS CODE

- Let’s skip the mandatory CI/CD definition slide, as you guys all know what it is…
- PAC is not new: The general approach is that pipelines themselves should be placed under version control, and use modularized refactored (reusable) components that are well tested
- The quest to manage yaml rather than complex scripting
- Github actions, Gitlab Containers, Jenkins Plugins, Concourse CI Containers



DEVELOPMENT ENVIRONMENT SETUP FRUSTRATION!

- The usual suspects…
  - Managing OS level and Language level dependencies
  - Configuring a linter, customizing the linting rules- adapting the linter to any frameworks you are using (Django comes to mind…)
  - Configuring formatters for sorting imports and automating the bulk of the day to day linting stuff (black, yapf, prettier)
  - Configuring a convention for tests, and a test discovery tool, (pytest, jest)
  - Configuring a coverage tool, and ensuring it works with your test suites (coverage, istanbul)
  - Configuring a dependency audit tool (safety, npm audit)
  - Any other security testing tools you want to run to catch hard coded passwords, etc. (gitleaks, bandit…)

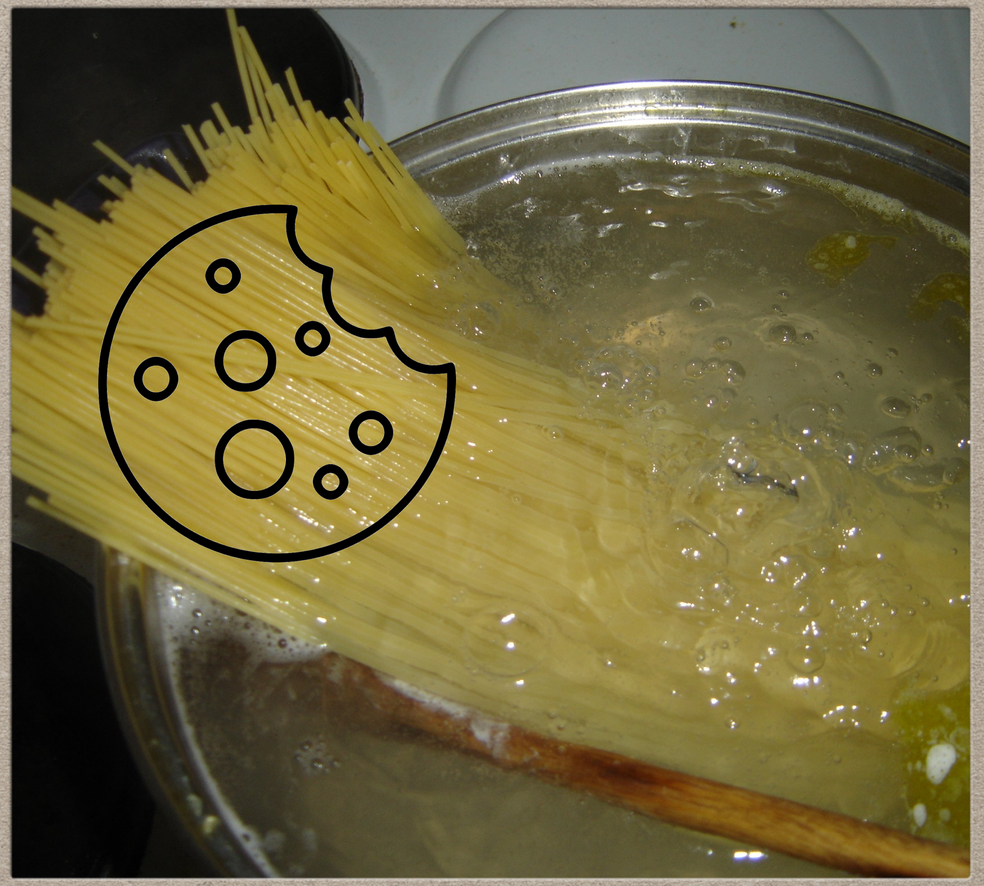
PIPELINE SURGERY

THE SIMPLE SOLUTION: BASH EVERYTHING

- Pull out all the logic from the pipeline and embed it in the development container
- Use a multistage build to produce a production container, without the dev niceties
- Don’t re-invent the wheel, use the development environment in the CI/CD pipeline as you have already implemented all the steps you need
- Complement the container with third party plugins/containers for CD or whatever else is needed in CI
- Glue everything together with BASH

PYTHON HAS ALL THE TOOLS, BUT JAVASCRIPT LOVES BOILERPLATE

- My own journey from jQuery to React, introduced me to templating as a basic language feature…
  - npm init
  - create-react-app / create-next-app
- I started looking for a way to replicate my tweaked bash scripts and python development environment:
  - cookiecutter

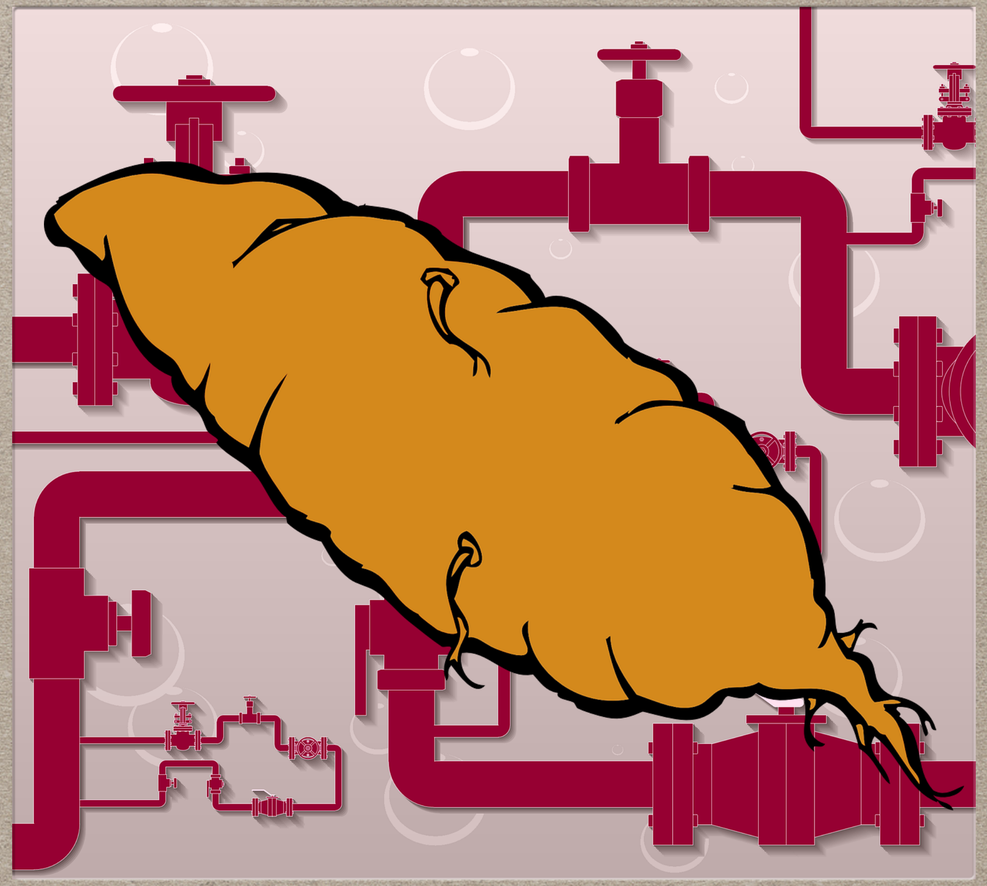


TEMPLATED SOLUTION #1: BASH COOKIES

- Cookiecutter uses Jinja2 under the hood, I know Ansible quite well, so I figured this qualified me to bake. So I made a codebase factory first…
- I extracted a generic template from the repository, that contained:
  - Linting configuration, Testing setup, Dependency Management, the Dockerfile, the Documentation, and the Bash scripting that glued it all together
- I like Github workflows. I'm just calling the bash scripts in my Dev environment for the most part, so it was really easy to template my GitHub workflow yaml as well:
  - Pipeline factory implementation #1 was born

TEMPLATING PIPELINES AS CODE

- **Because yaml is syntactically simple compared to a bash or python script, it's very easy to template**
- **Because my pipeline code invokes the dev environment for a good chunk of the CI steps, it's already pretty refactored and battle tested**
- **Because of the extensibility of PAC, it's very easy to add more functionality to the pipeline**



SOLUTION #1 DOWNSIDES: IT'S BASH... SO YOUR COOKIES MAY CRUMBLE.

- This worked quite well. I had CI/CD from day one on new repositories- whether they were packages, or Flask or Django applications
- The pipeline guts were all in the codebase, so the pipeline was fairly portable... I was able to take it over to Gitlab without too much trouble
- But it's BASH...
  - not easily testable
  - hard to maintain...
  - easy to break...

TEMPLATED SOLUTION #2: PUT PANTS ON

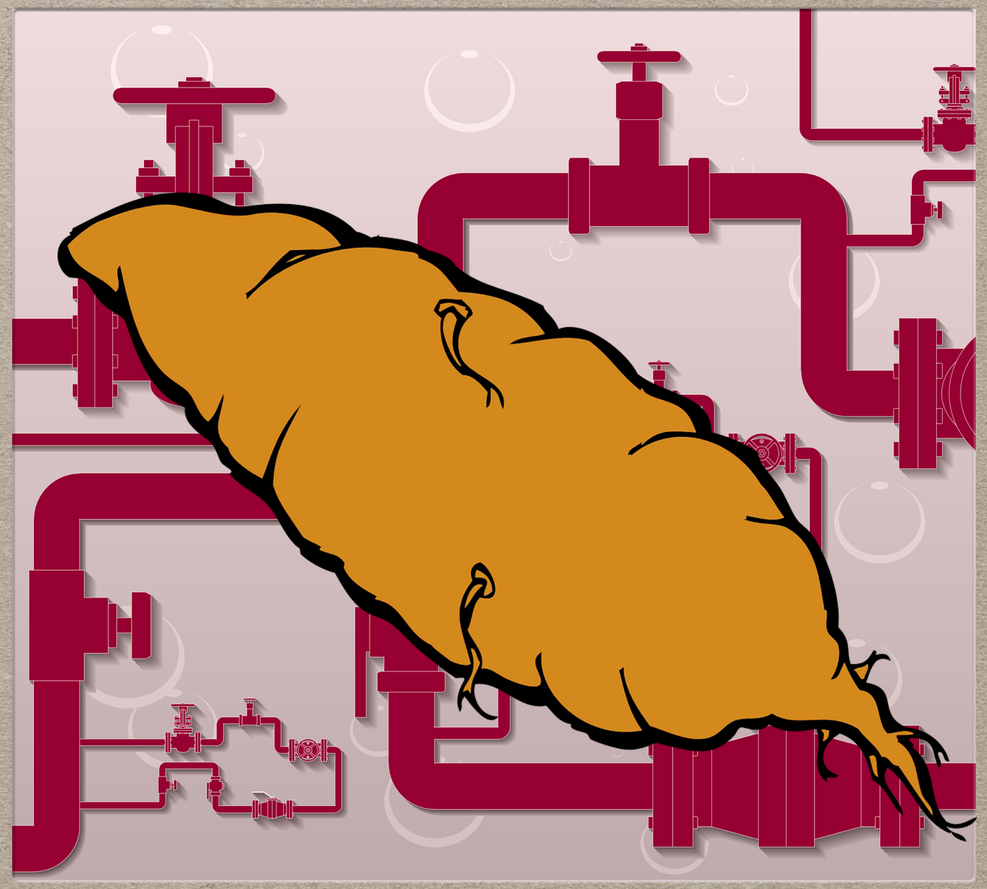
- Pants was developed by Twitter, Foursquare and Square as a build system
- It has some pretty neat parallelization features, that drastically speed up testing and linting of large codebases
- It’s tested code
- It’s trivial to keep the existing container api using some BASH aliasing

TEMPLATED SOLUTION #3: A PYTHON CLI

- We already have an easy to test language in the container: Python
- We have BASH scripts that outline all the functionality we need
- Can we convert the BASH into Python, and create a minimal, testable CLI that is easy to maintain?

RECAP

- PAC lets us manage simple yaml documents rather than complex scripts
- It may make sense to reuse the development environment tooling for code analysis, linters in any CI/CD to avoid introducing splintered execution environments
- End to end testing is hard, and time consuming. It may be more efficient to keep templates of working pipeline configurations, so they can quickly be reused
- New repositories are nicer/faster to get off the ground if you can provide some templating from day one



SOME RELEVANT LINKS

- https://github.com/cookiecutter/cookiecutter
- https://github.com/yeoman/yeoman
- https://www.recallstack.icu/en/2020/04/18/yeoman-and-cookiecutter-are-dead-long-live-copier/
- https://github.com/databrickslabs/cicd-templates
- https://github.com/jondot/hygen
- https://www.cprime.com/resources/blog/templatizing-delivery-the-future-of-ci-cd/
- https://hodovi.cc/blog/creating-templates-for-gitlab-ci-jobs/
- https://gitlab.freedesktop.org/wayland/ci-templates
- https://hceris.com/templating-concourse-pipelines-with-jsonnet/

PYTHON-IN-A-BOX

- https://github.com/shared-vision-solutions/python-in-a-box
- https://github.com/shared-vision-solutions/pib_cli
- https://github.com/niall-byrne/mmmm_cookies