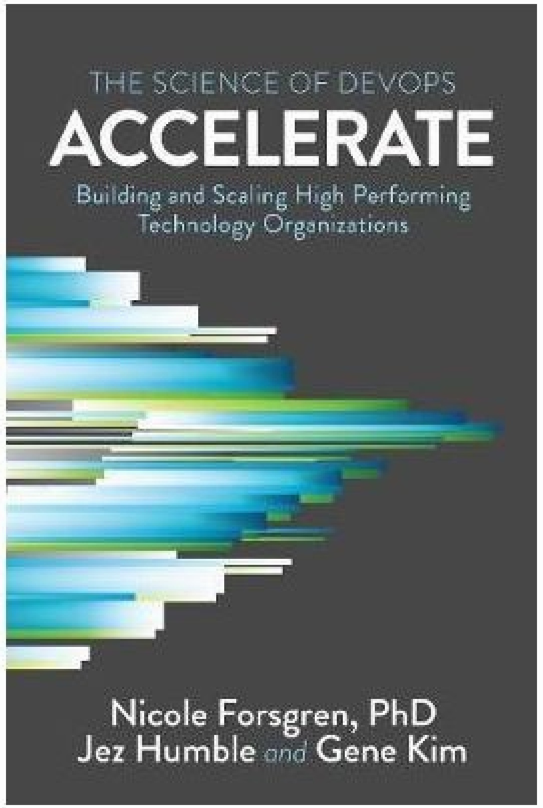
Build pipelines that don't suck

It is about Continuous Delivery

Four key metrics

- Lead time
- Deployment frequency
- Mean time to restore
- Change fail percentage


THE SCIENCE OF DEVOPS

ACCELERATE

Building and Scaling High Performing Technology Organizations

Nicole Forsgren, PhD
Jez Humble and Gene Kim

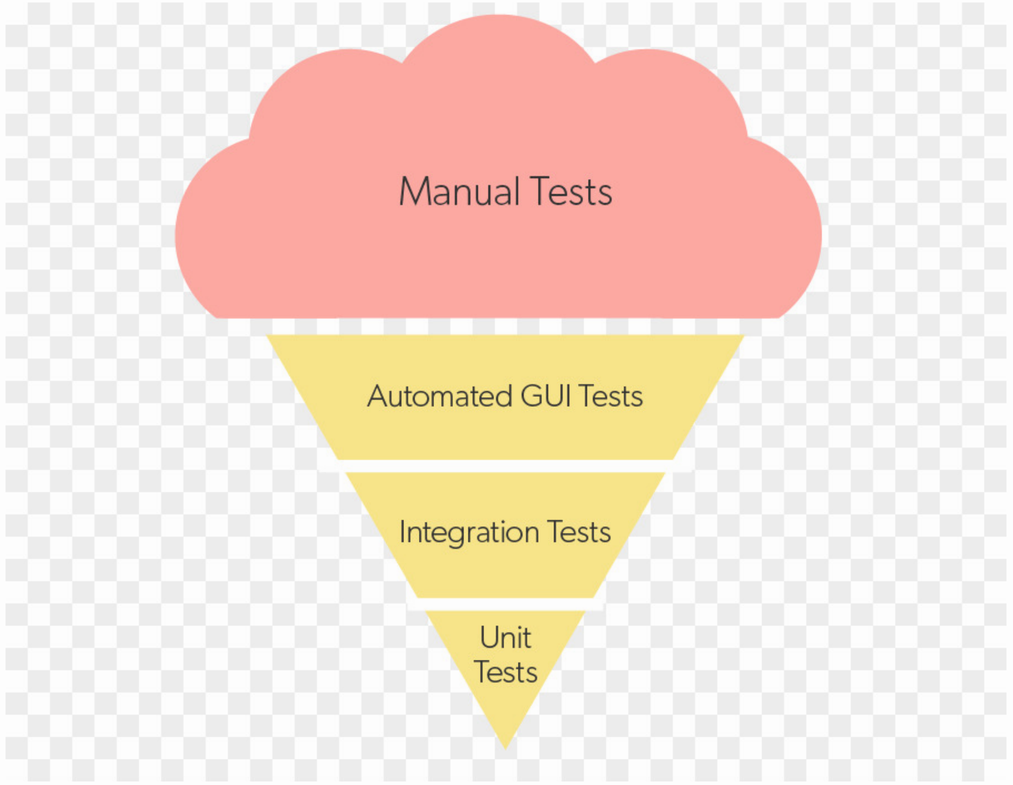
https://www.thoughtworks.com/radar/techniques/four-key-metrics

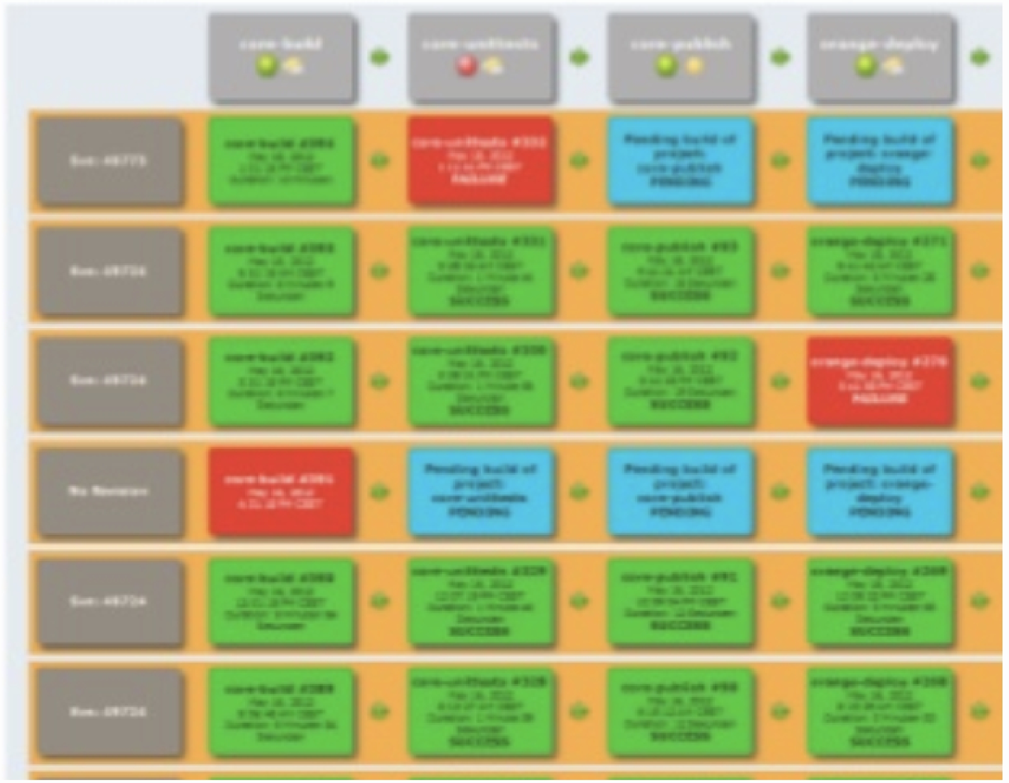
- Lead time
- **Deployment frequency**
- Mean time to restore
- **Change fail percentage**

Before we talk about pipelines...

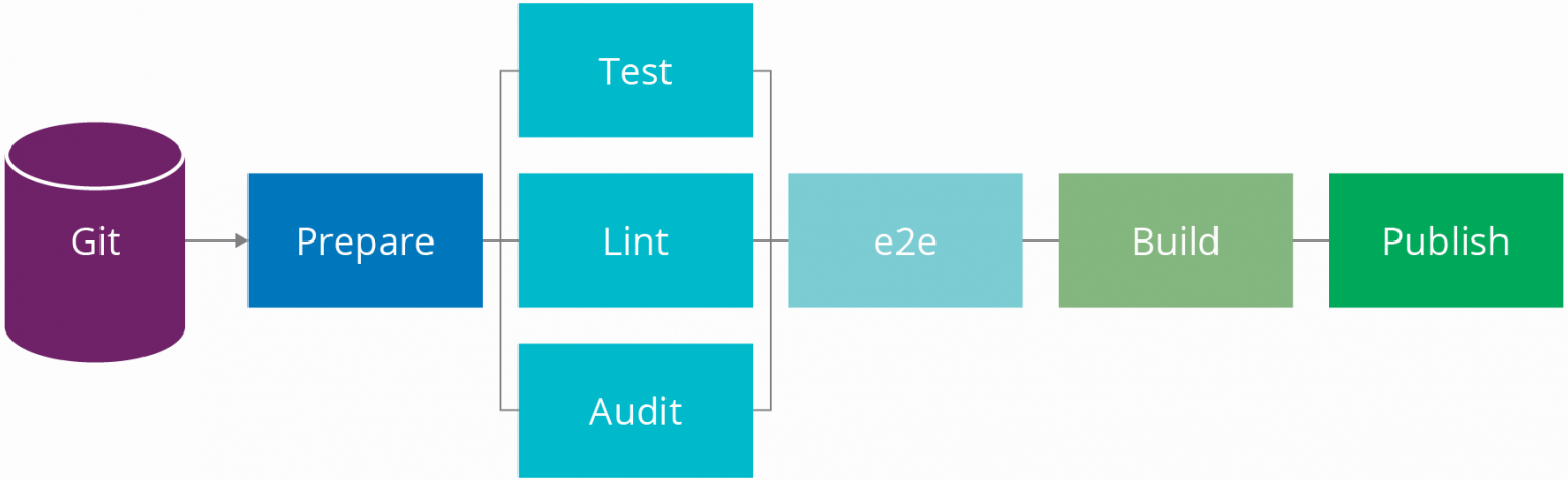
CI/CD is not an antidote for your dysfunction

A link to the past





Our target


Git
Prepare
Test
Lint
Audit
e2e
Build
Publish

A good pipeline is ...

A good pipeline *is* code

```yaml
- name: test
  serial: true
  plan:
  - aggregate:
    - get: git
      passed: [prepare]
      trigger: true
    - get: dev-container
      passed: [prepare]
  - task: test-js
    image: dev-container
    params:
      <<: *common-params
      TARGET: js
    file: git/pipeline/tasks/tests/task.yml
```

```yaml
jobs:
  build:

    working_directory: ~/app

    docker:
      - image: circleci/node:11.10.1

    steps:

      - checkout
      - run: yarn
      - run: yarn run linter:js
      - run: yarn run linter:css
      - run: yarn run linter:text
      - run: yarn test --coverage --runInBand
```

Don't modify it through a UI

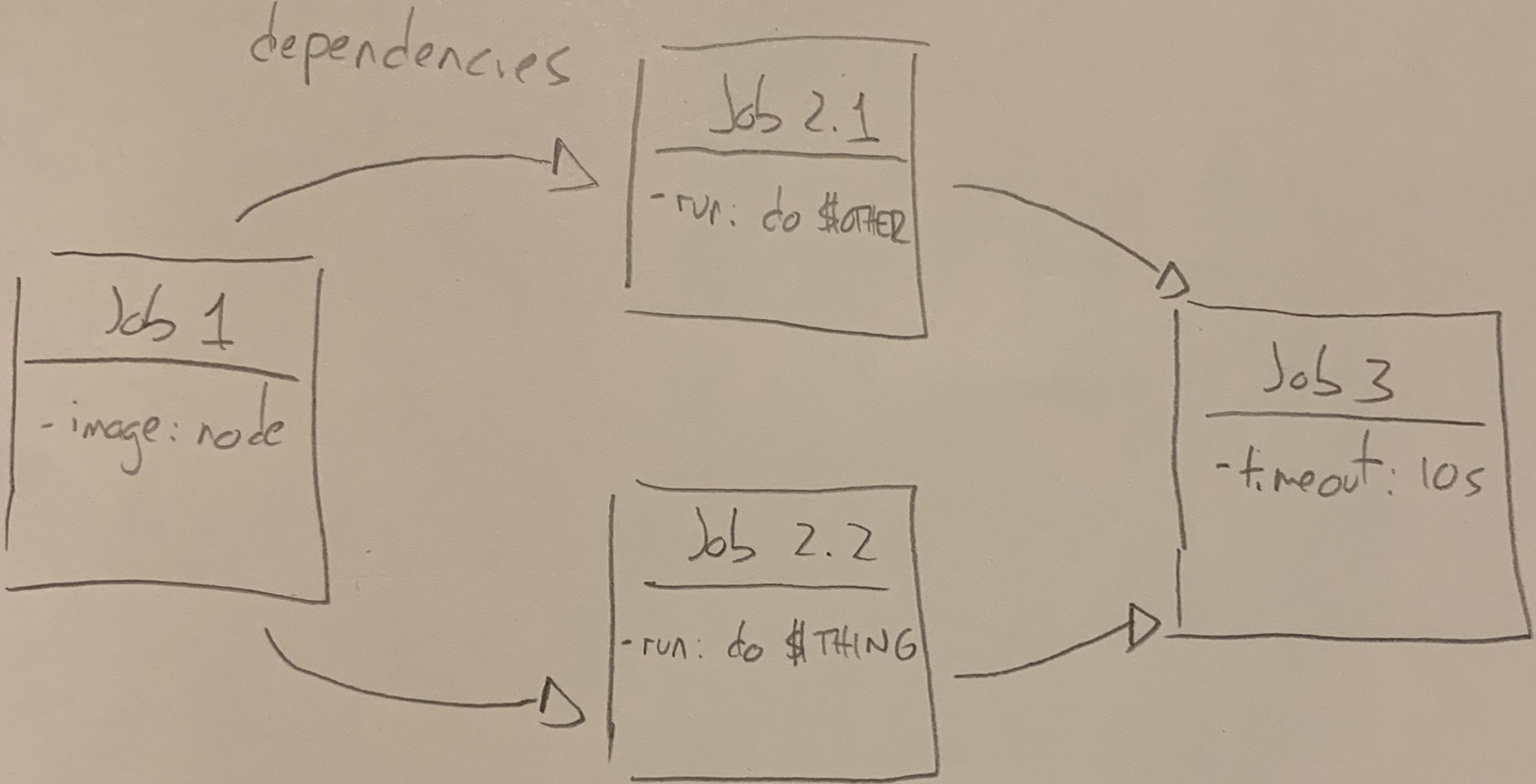
Keep it close to the app

Keep it versioned

https://www.gocd.org/2017/05/02/what-does-pipelines-as-code-really-mean/

A good pipeline is maintainable

Declarative


dependencies
Job 1
- image: node
Job 2.1
- run: do $OTHER
Job 2.2
- run: do $THING
Job 3
- timeout: 10s

Not a full blown programming language

Locally executable

https://www.thoughtworks.com/insights/blog/praise-go-script-part-i

```
./go
usage: ./go <goal>

  goal:

  linter-js                        -- Run the linter for js files
  linter-css                       -- Run the linter for css files
  linter-html                      -- Run the linter for html files
  linter                           -- Run all linters

  test-js                          -- Run unit tests

  audit                            -- Audit packages

  e2e                              -- Run end to end tests

  build                            -- Build the bundle
```

```
goal_test-js() {
  export MAPS_KEY=${MAPS_KEY:-$(gopass store/map-key)}
  npm t
}
```

```
goal_build() {
  gradle_with_credentials build -x test
}

gradle_with_credentials() {
  if [ -z "$USER" ] || [ -z "$PASSWORD" ]; then
    ./gradlew "$@"
  else
    ./gradlew -Puser=$USER -Ppassword=$PASSWORD "$@"
  fi
}
```

A good pipeline is reliable

Isolation

```
FROM node:11.11-stretch

SHELL ["/bin/bash", "-o", "pipefail", "-c"]

RUN apt-get update && \
  apt-get -y install --no-install-recommends \
  # Chrome
  libx11-xcb1 libxcomposite1 libxcursor1 libxdamage1 libxi6 libxtst6 \
  libnss3 libxss1 libcups2 libxrandr2 libasound2 \
  libatk1.0-0 libatk-bridge2.0-0 libgtk-3-0 \
  sudo curl shellcheck unzip rsync jq && \
  apt-get clean && \
  rm -rf /var/lib/apt/lists/*
```

Tradeoff: Isolation and speed

Beware of external systems

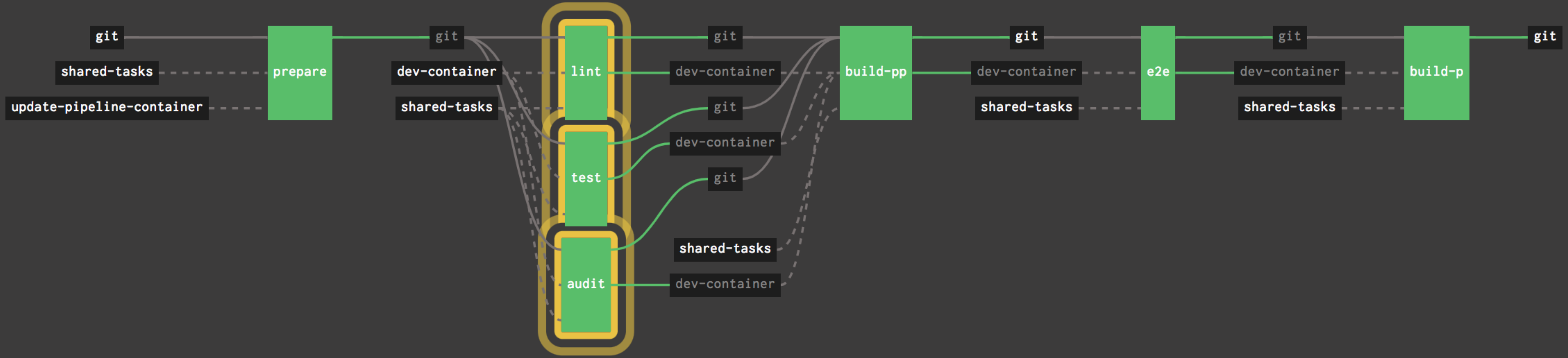
A good pipeline is fast

Throw hardware at the problem

Up-to-date dependencies

PhantomJS

Parallelization


git
shared-tasks
update-pipeline-container
prepare
git
dev-container
shared-tasks
lint
test
audit
git
dev-container
git
dev-container
git
shared-tasks
dev-container
build-pp
git
dev-container
shared-tasks
e2e
git
dev-container
shared-tasks
build-p
git

Avoid repeating steps

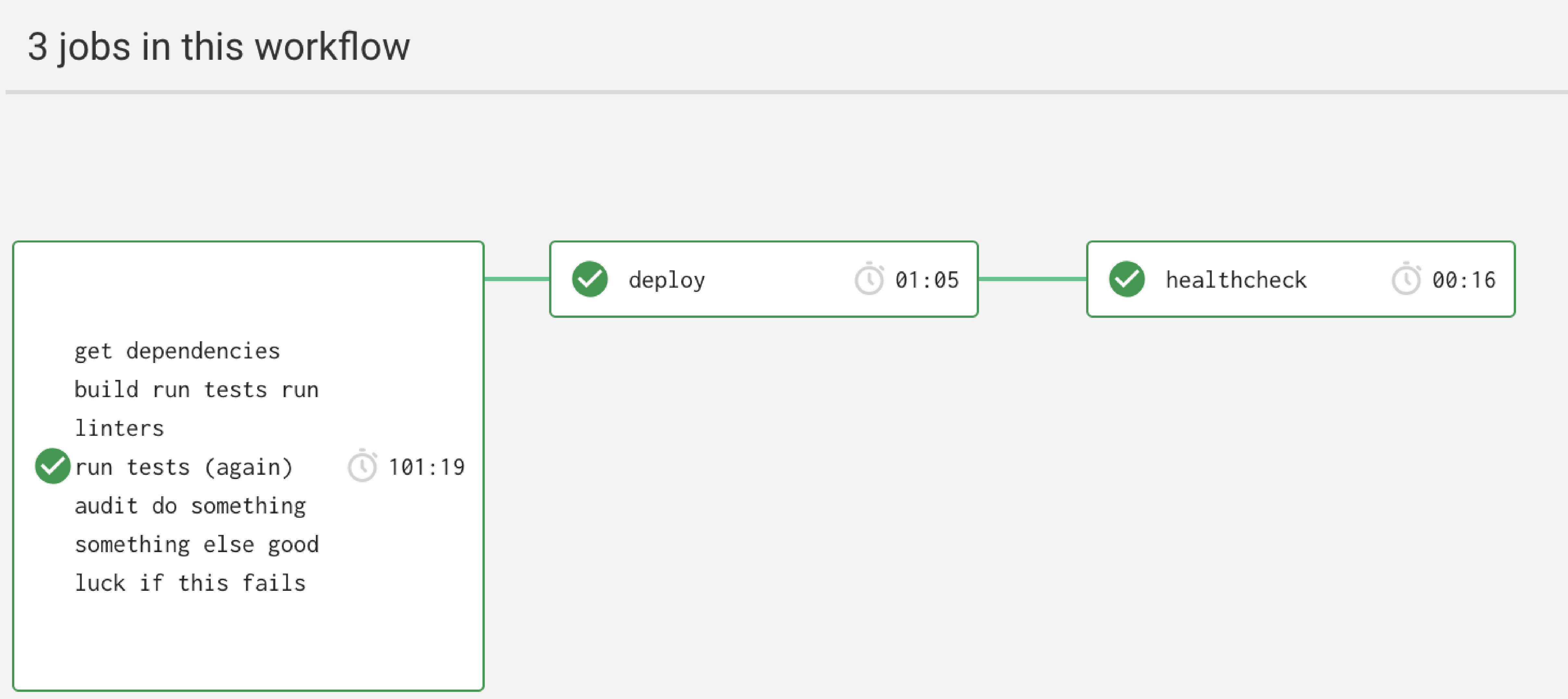
Caching

```yaml
platform: linux
inputs:
  - name: git
  - name: shared-tasks
caches:
  - path: git/node_modules
```

```yaml
platform: linux
inputs:
  - name: git
caches:
  - path: gradle
params:
  GRADLE_USER_HOME: ../gradle
```

A good pipeline is visual

The god-step



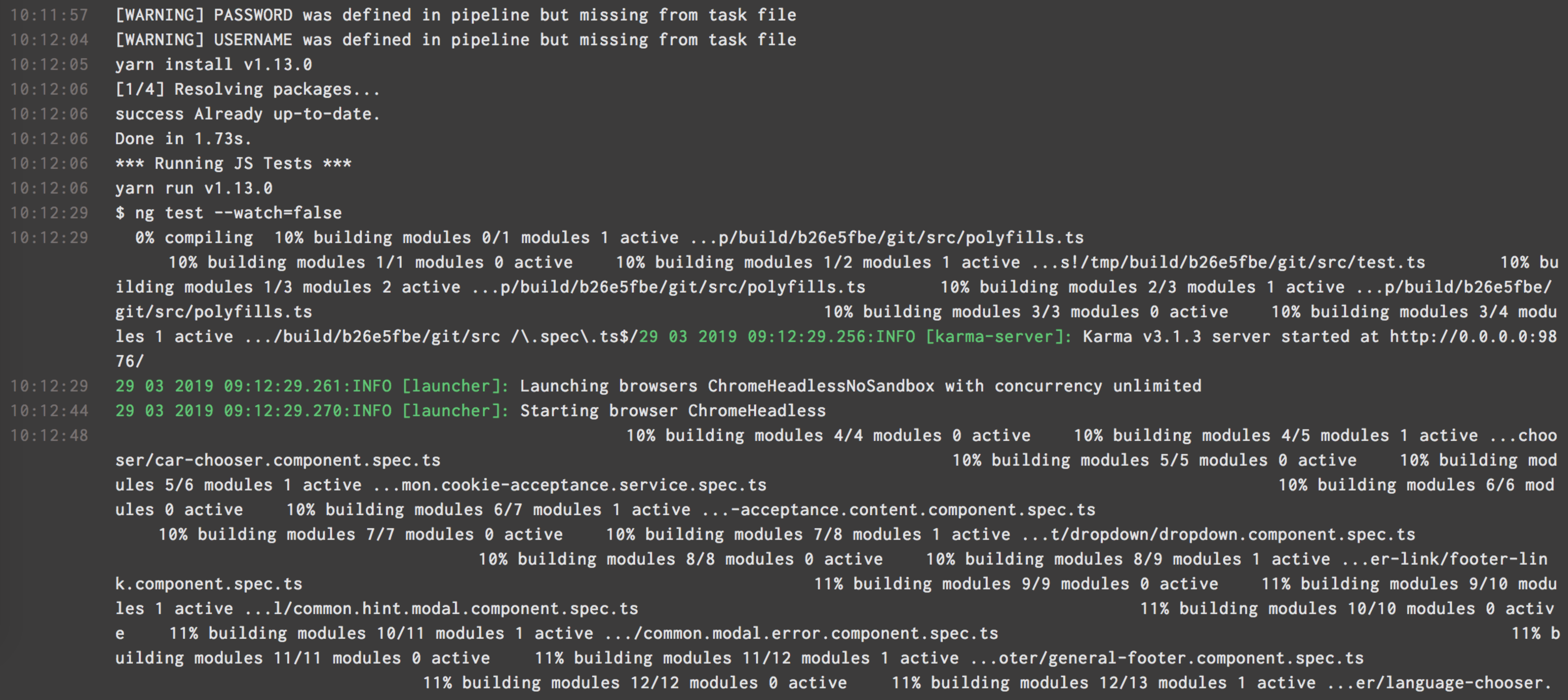
3 jobs in this workflow

```
get dependencies
build run tests run
linters
run tests (again)        101:19
audit do something
something else good
luck if this fails
```

```
deploy        01:05
```

```
healthcheck        00:16
```

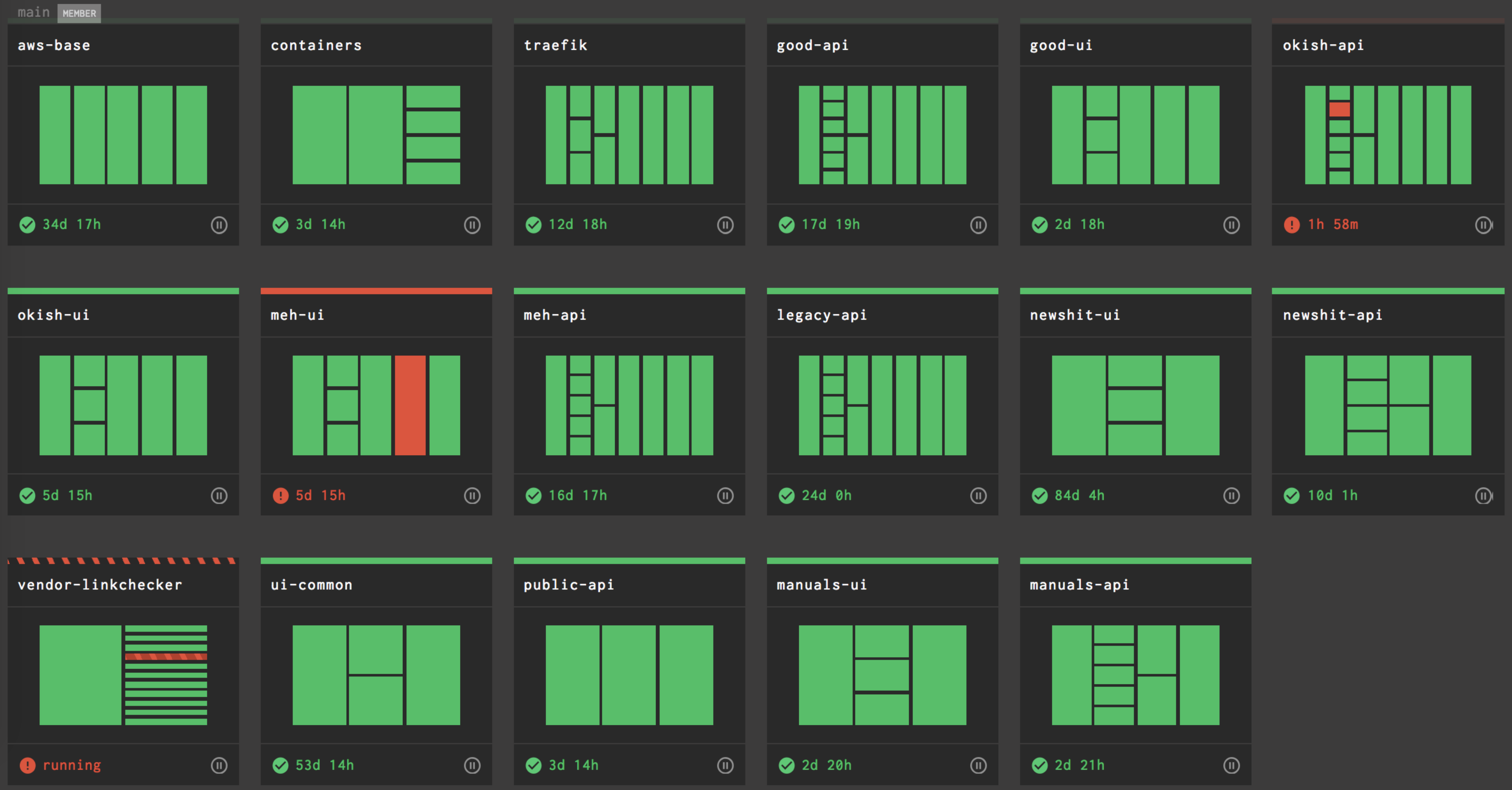
Logging overdose



```
10:11:57  [WARNING] PASSWORD was defined in pipeline but missing from task file
10:12:04  [WARNING] USERNAME was defined in pipeline but missing from task file
10:12:05  yarn install v1.13.0
10:12:06  [1/4] Resolving packages...
10:12:06  success Already up-to-date.
10:12:06  Done in 1.73s.
10:12:06  *** Running JS Tests ***
10:12:06  yarn run v1.13.0
10:12:29  $ ng test --watch=false
10:12:29    0% compiling  10% building modules 0/1 modules 1 active ...p/build/b26e5fbe/git/src/polyfills.ts
               10% building modules 1/1 modules 0 active    10% building modules 1/2 modules 1 active ...s!/tmp/build/b26e5fbe/git/src/test.ts       10% bu
          ilding modules 1/3 modules 2 active ...p/build/b26e5fbe/git/src/polyfills.ts       10% building modules 2/3 modules 1 active ...p/build/b26e5fbe/
          git/src/polyfills.ts                                                    10% building modules 3/3 modules 0 active    10% building modules 3/4 modu
          les 1 active .../build/b26e5fbe/git/src /\.spec\.ts$/29 03 2019 09:12:29.256:INFO [karma-server]: Karma v3.1.3 server started at http://0.0.0.0:98
          76/
10:12:29  29 03 2019 09:12:29.261:INFO [launcher]: Launching browsers ChromeHeadlessNoSandbox with concurrency unlimited
10:12:44  29 03 2019 09:12:29.270:INFO [launcher]: Starting browser ChromeHeadless
10:12:48                                                     10% building modules 4/4 modules 0 active    10% building modules 4/5 modules 1 active ...choo
          ser/car-chooser.component.spec.ts                                                 10% building modules 5/5 modules 0 active    10% building mod
          ules 5/6 modules 1 active ...mon.cookie-acceptance.service.spec.ts                                                       10% building modules 6/6 mod
          ules 0 active    10% building modules 6/7 modules 1 active ...-acceptance.content.component.spec.ts
              10% building modules 7/7 modules 0 active    10% building modules 7/8 modules 1 active ...t/dropdown/dropdown.component.spec.ts
                                              10% building modules 8/8 modules 0 active    10% building modules 8/9 modules 1 active ...er-link/footer-lin
          k.component.spec.ts                                                     11% building modules 9/9 modules 0 active    11% building modules 9/10 modu
          les 1 active ...l/common.hint.modal.component.spec.ts                                                  11% building modules 10/10 modules 0 activ
          e    11% building modules 10/11 modules 1 active .../common.modal.error.component.spec.ts                                                    11% b
          uilding modules 11/11 modules 0 active    11% building modules 11/12 modules 1 active ...oter/general-footer.component.spec.ts
                                         11% building modules 12/12 modules 0 active    11% building modules 12/13 modules 1 active ...er/language-chooser.
```

Visualize dependencies

A good pipeline is scalable


main MEMBER

aws-base
34d 17h

containers
3d 14h

traefik
12d 18h

good-api
17d 19h

good-ui
2d 18h

okish-api
1h 58m

okish-ui
5d 15h

meh-ui
5d 15h

meh-api
16d 17h

legacy-api
24d 0h

newshit-ui
84d 4h

newshit-api
10d 1h

vendor-linkchecker
running

ui-common
53d 14h

public-api
3d 14h

manuals-ui
2d 20h

manuals-api
2d 21h

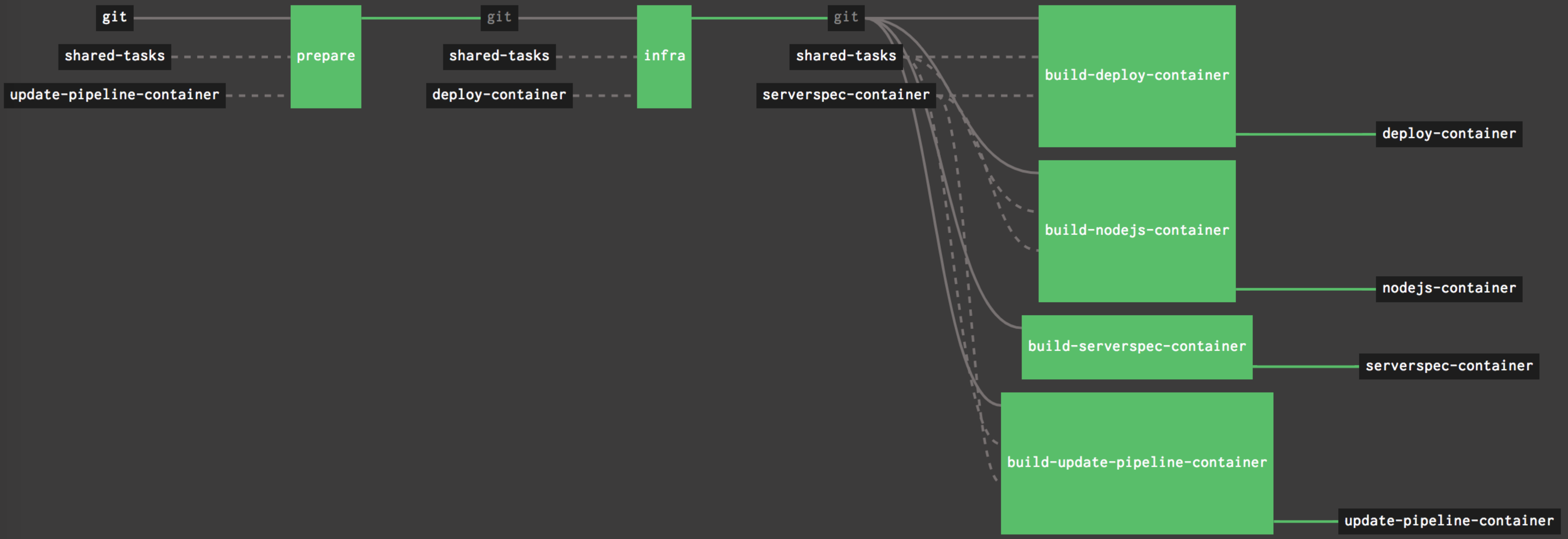
Tradeoff: Reuse vs Coupling

Follow a similar structure

Parametrized steps

```yaml
platform: linux
inputs:
  - name: git
  - name: shared-tasks
caches:
  - path: git/node_modules
params:
  TARGET:
run:
  path: sh
  dir: git
  args:
  - -ec
  - |
    ../shared-tasks/scripts/install-yarn-packages.sh
    ./go linter-${TARGET}
```

Shared containers


git
shared-tasks
update-pipeline-container
prepare
git
shared-tasks
deploy-container
infra
git
shared-tasks
serverspec-container
build-deploy-container
deploy-container
build-nodejs-container
nodejs-container
build-serverspec-container
serverspec-container
build-update-pipeline-container
update-pipeline-container

Summary

A good pipeline is ...

- code
- maintainable
- reliable
- fast
- visual
- scalable

Tell me what CI/CD tool to use!

It is not that important

It won't be for lack of options

Top Continuous Integration Tools: 51 Tools to Streamline Your Development Process, Boost Quality, and Enhance Accuracy

Let's finish with a quote

> You know what I love? Spending more time retrying jobs until the build is green than building the *damn* feature itself

Nobody, ever

Mario Fernandez

Lead Developer

ThoughtWorks