factor10

Pipeline patterns and anti-patterns

Things your pipeline should (not) do

Daniel Raniz Raneland
Coding Architect @ factor10

factor10.com
raniz@factor10.com
raniz@mastodon.online
raniz.blog
/in/raneland

Yogendra Singh via Unsplash

Priscilla Du Preez via Unsplash

Blake Wisz via Unsplash

Markus Spiske via Unsplash

Felix M. Dom via Unsplash

Anti-pattern #1

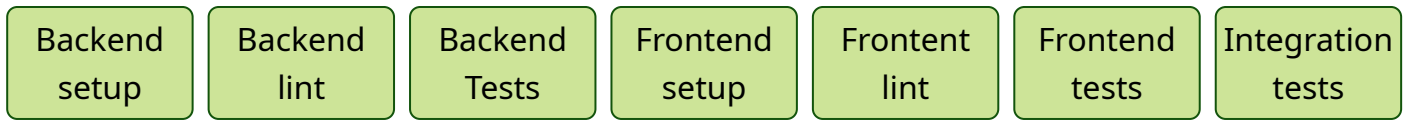
The Ritual

Doing everything all the time

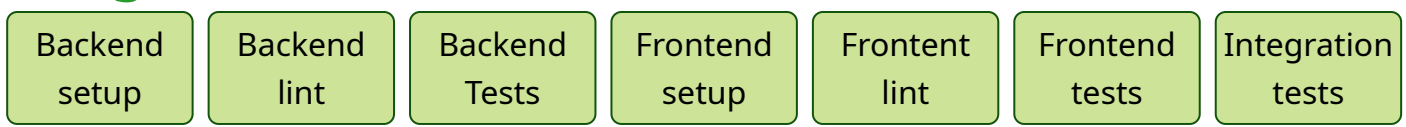
Chinh Le Duc via Unsplash

Antipattern #1: The Ritual

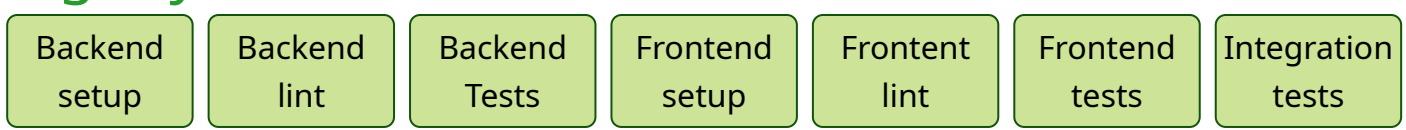
PR



Merge (main)

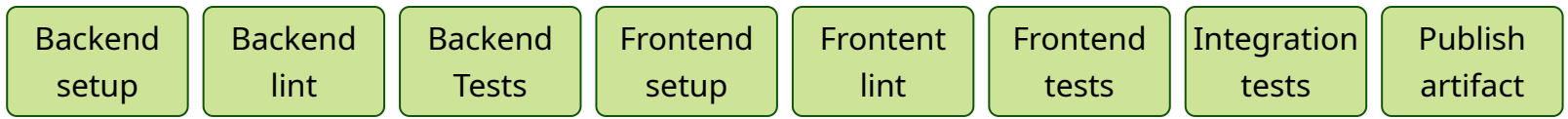


Nightly

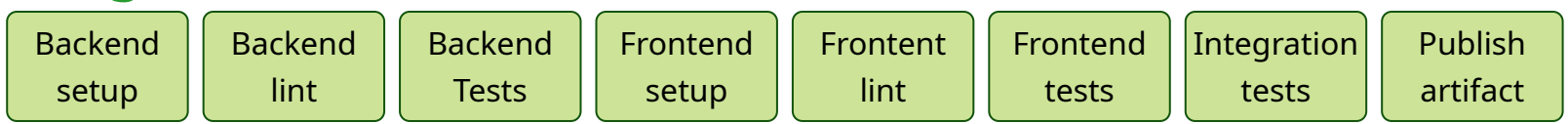


Antipattern #1: The Ritual

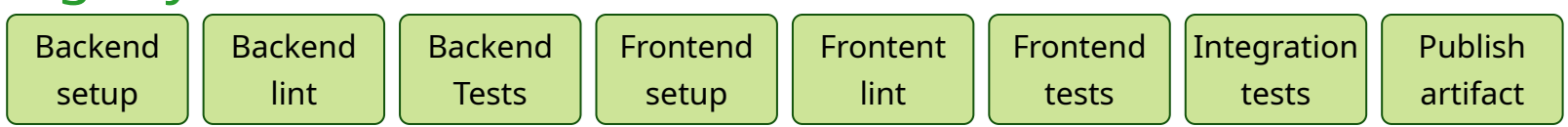
PR



Merge (main)



Nightly



Pattern #1: The right workflow for the job

PR

Backend setup | Backend lint | Backend Tests | Frontend setup | Frontent lint | Frontend tests | Integration tests

Merge (main)

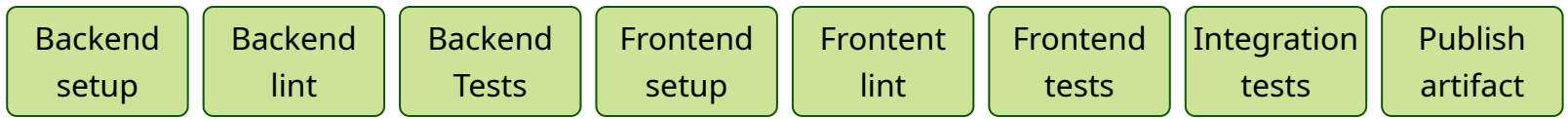
Backend setup | Backend Tests | Frontend setup | Frontend tests | Integration tests | Publish artifact

Nightly

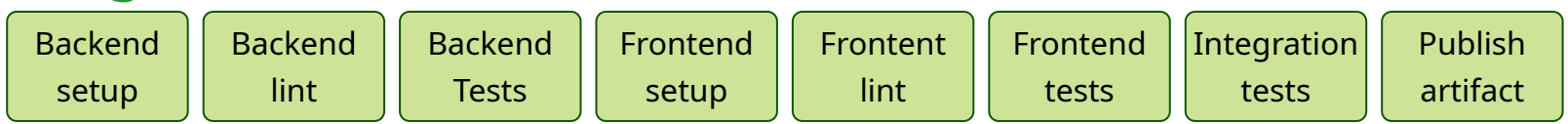
Backend setup | Frontend setup | Integration tests

Pattern #1: The right workflow for the job

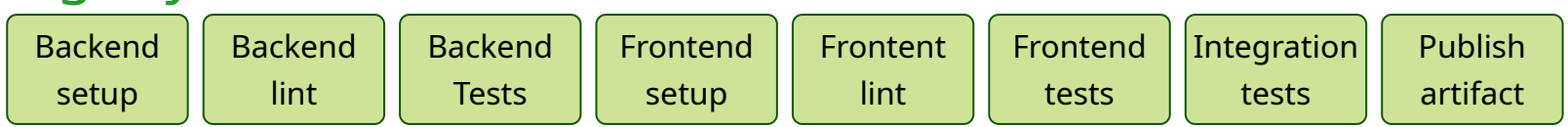
PR



Merge (main)

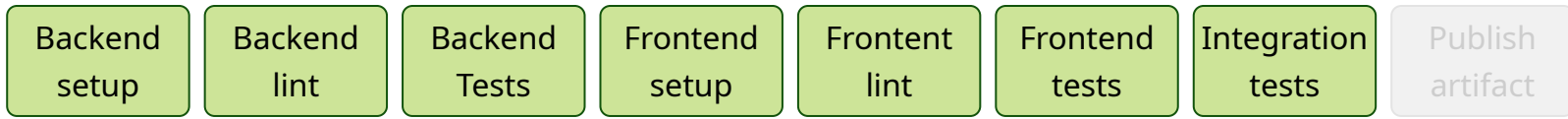


Nightly

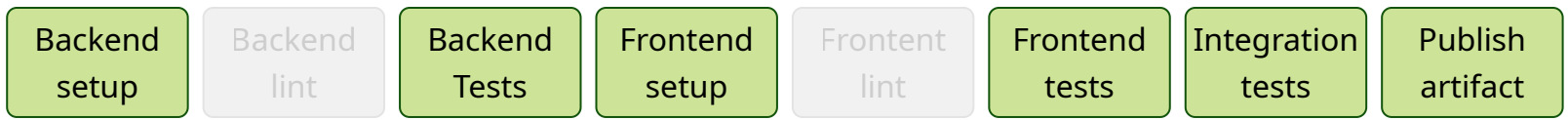


Pattern #2: Conditional workflows

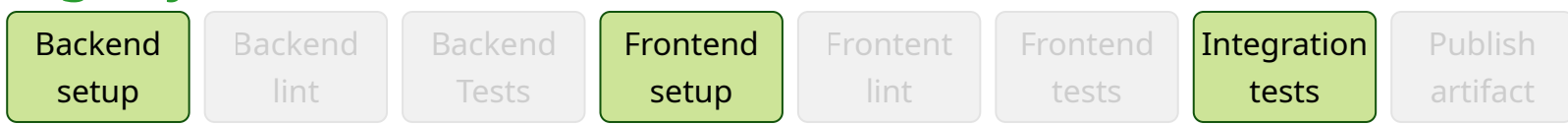
PR



Merge (main)

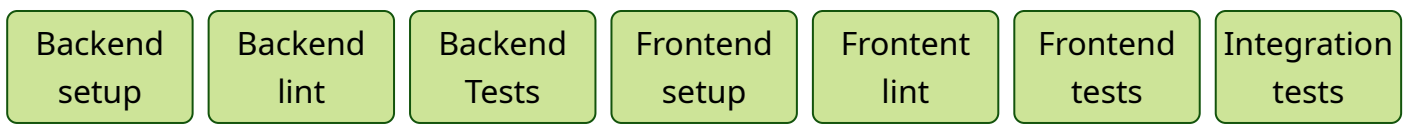


Nightly



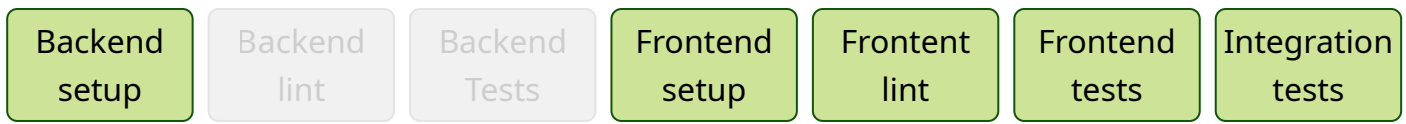
Pattern #2: Conditional workflows

PR

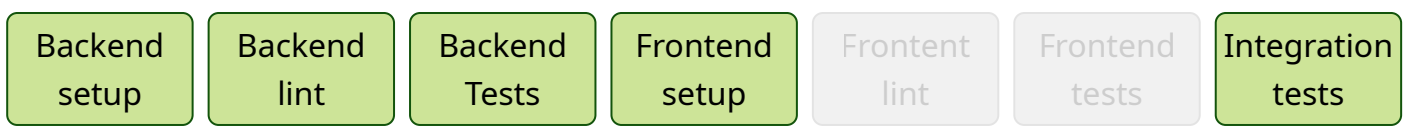


Pattern #2: Conditional workflows

"Hide settings menu when logged out"



"Add index on user birthdate"



Pattern #2: Conditional workflows

- Branch name (pattern)
- Tag
  - Is it a tag?
  - Pattern
- Trigger
  - PR
  - Push
  - Manual
- Custom
  - Changed files
  - Day of week
  - Time of day
  - Your choice

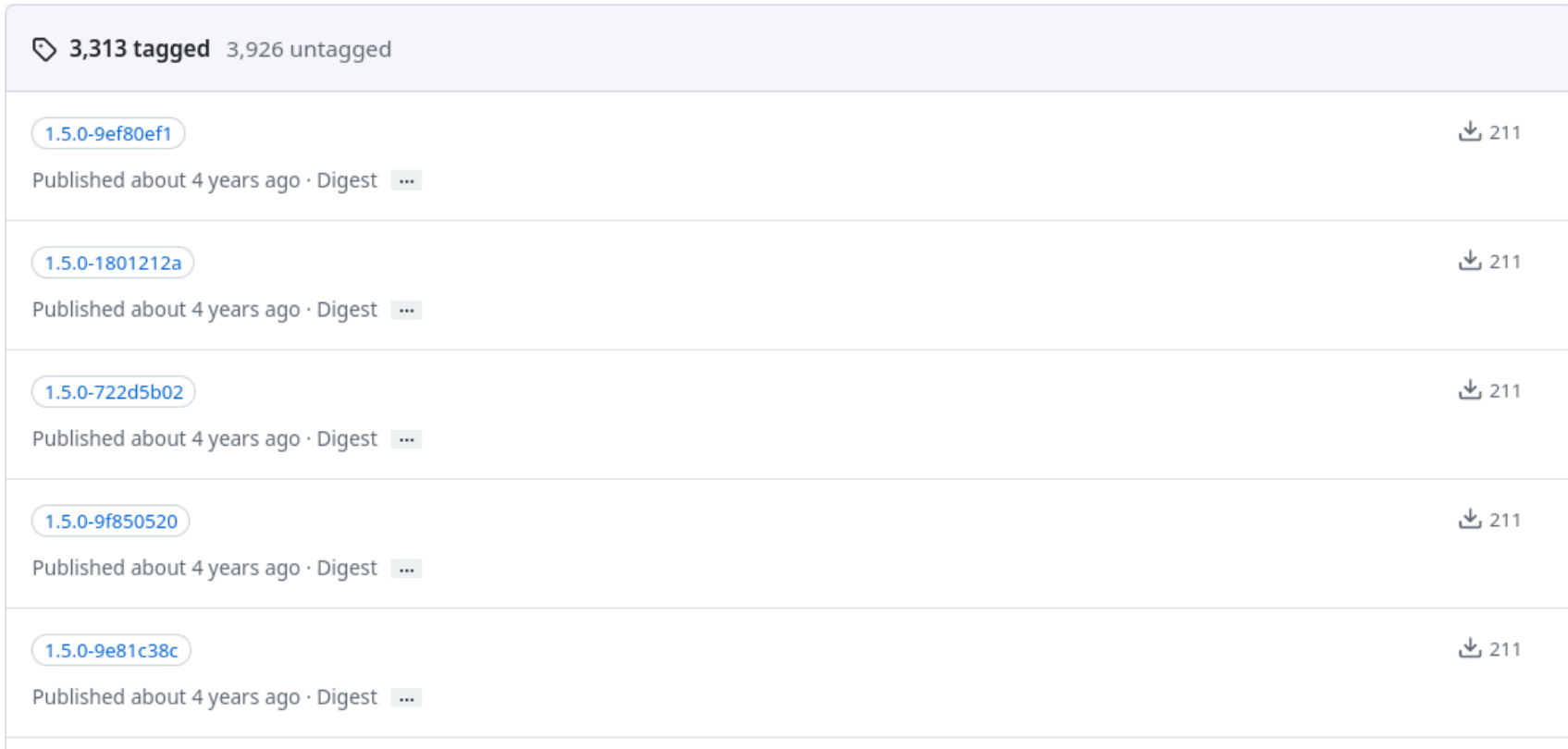
Anti-pattern #2

Hoarding

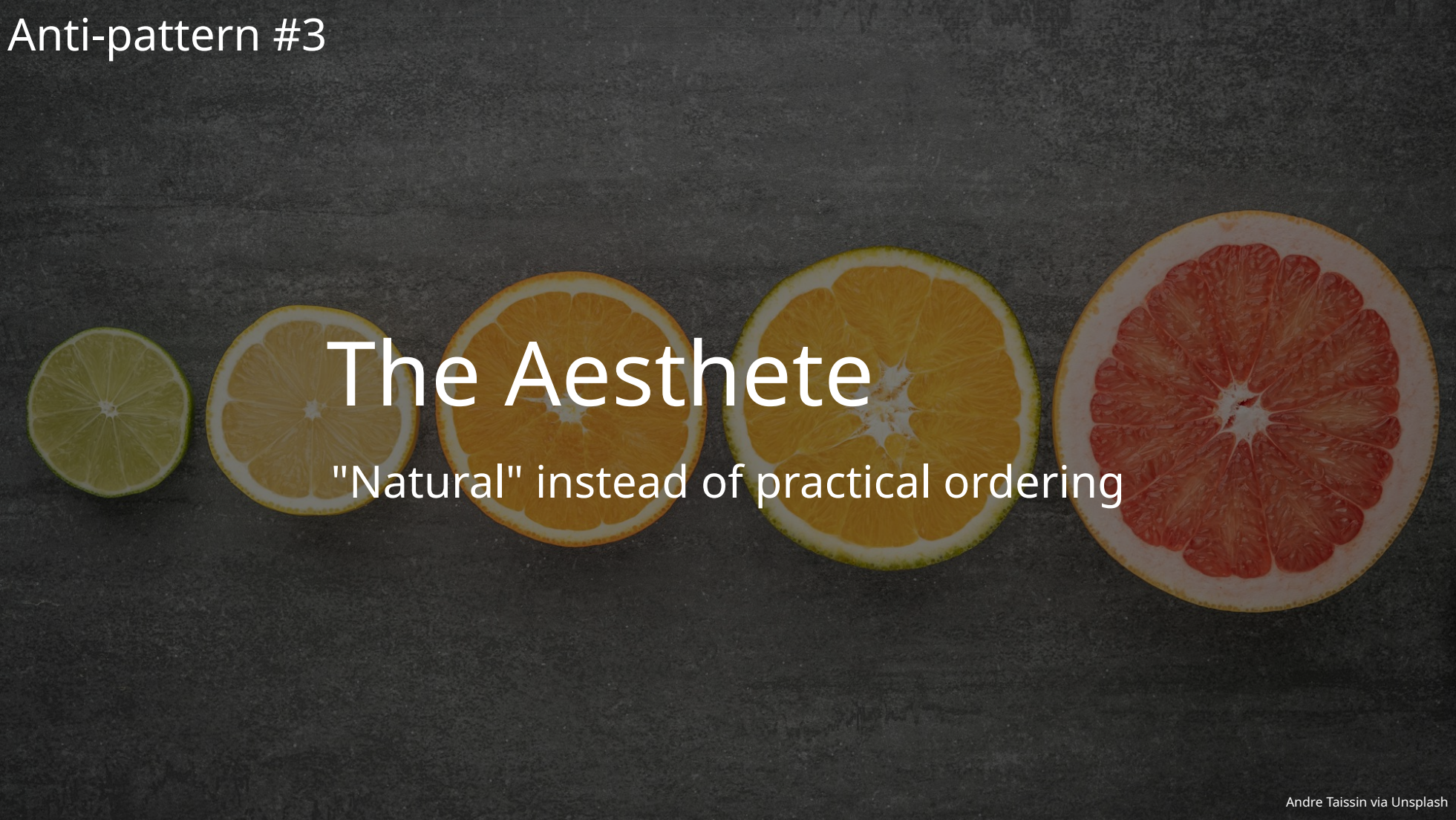
Keeping all artifacts forever

Cristian Gottardi via Unsplash

Antipattern #2: Hoarding



Anti-pattern #3



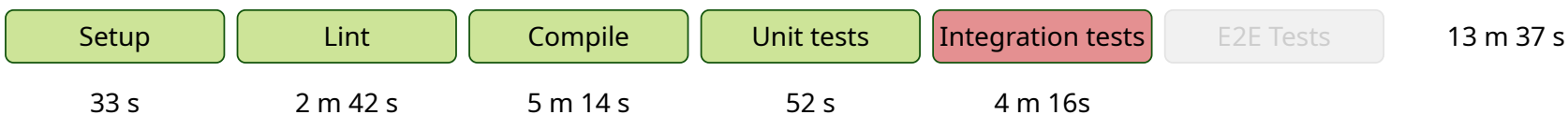
The Aesthete

"Natural" instead of practical ordering

Andre Taissin via Unsplash

Antipattern #3: The Aesthete

Antipattern #3: The Aesthete



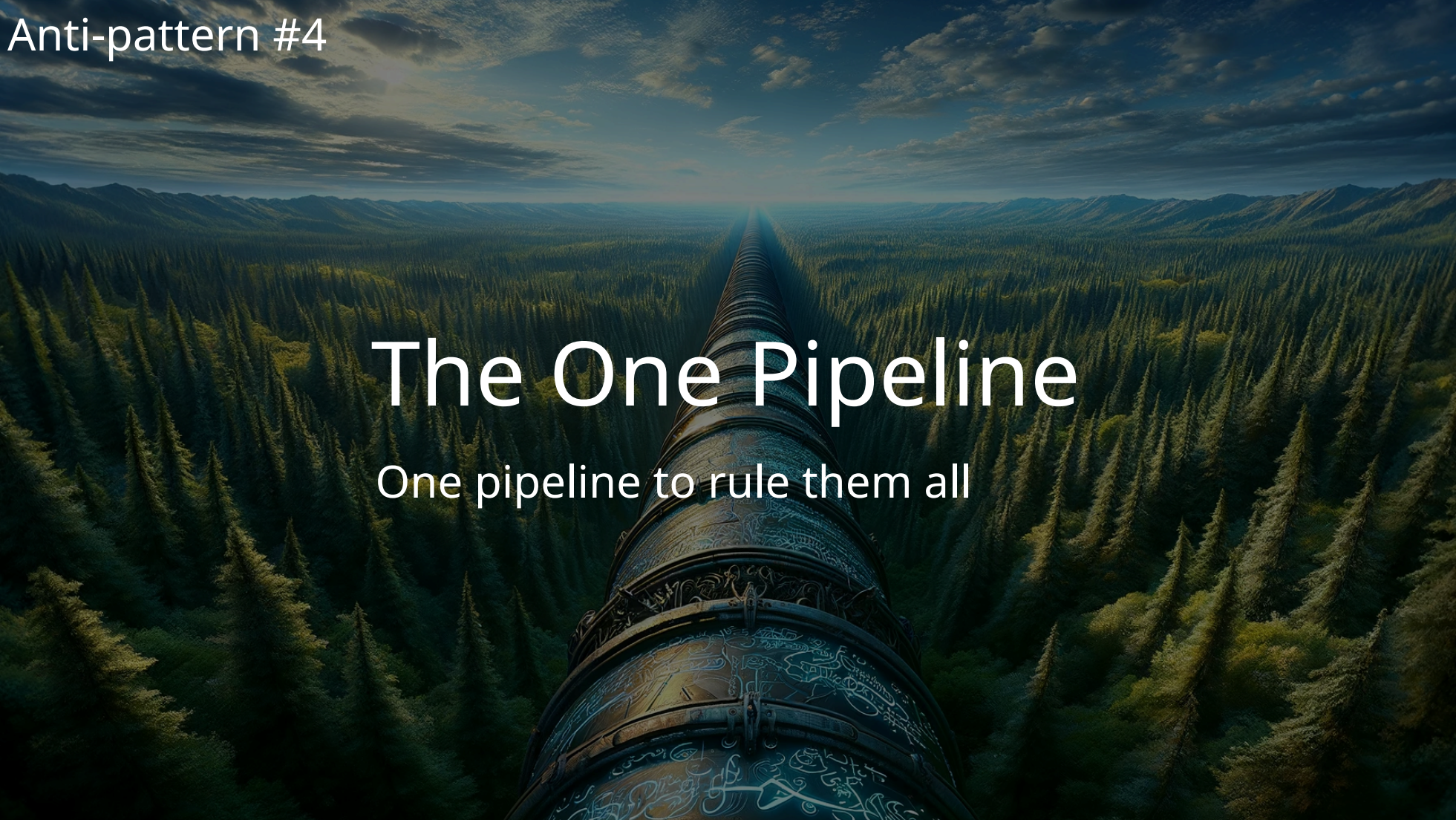
Antipattern #3: The Aesthete

Which steps fails?

Step	Caused failure
Setup	1 %
Lint	6 %
Compile	4 %
Unit tests	21 %
Integration tests	68 %

Pattern #3: Failing fast

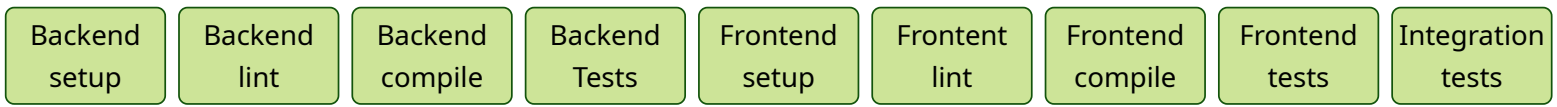
Anti-pattern #4



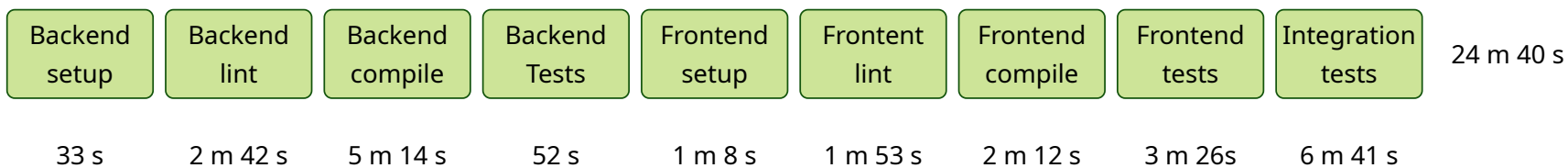
The One Pipeline

One pipeline to rule them all

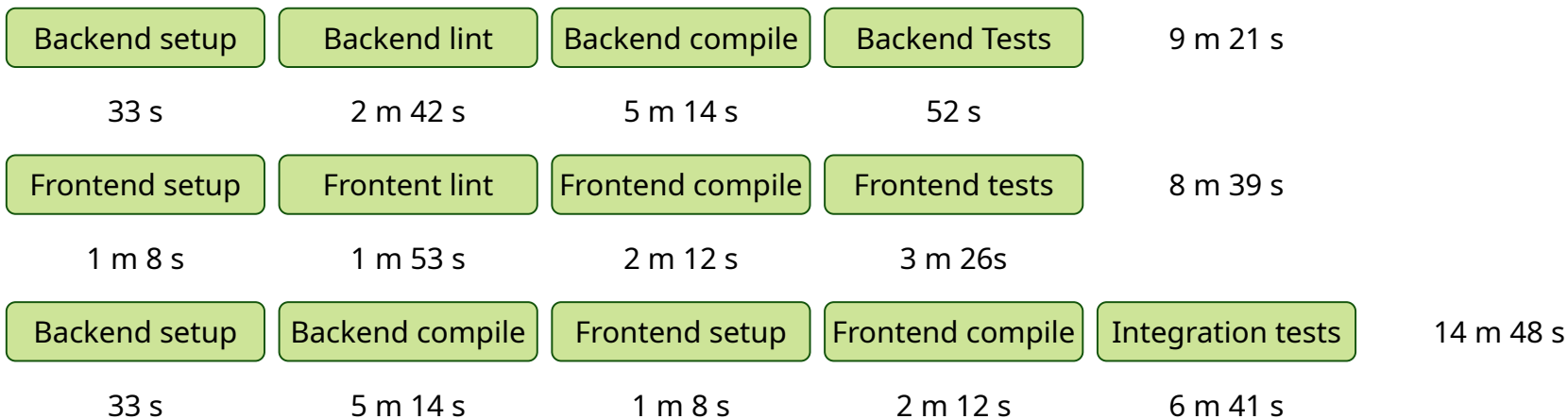
Antipattern #4: The One Pipeline



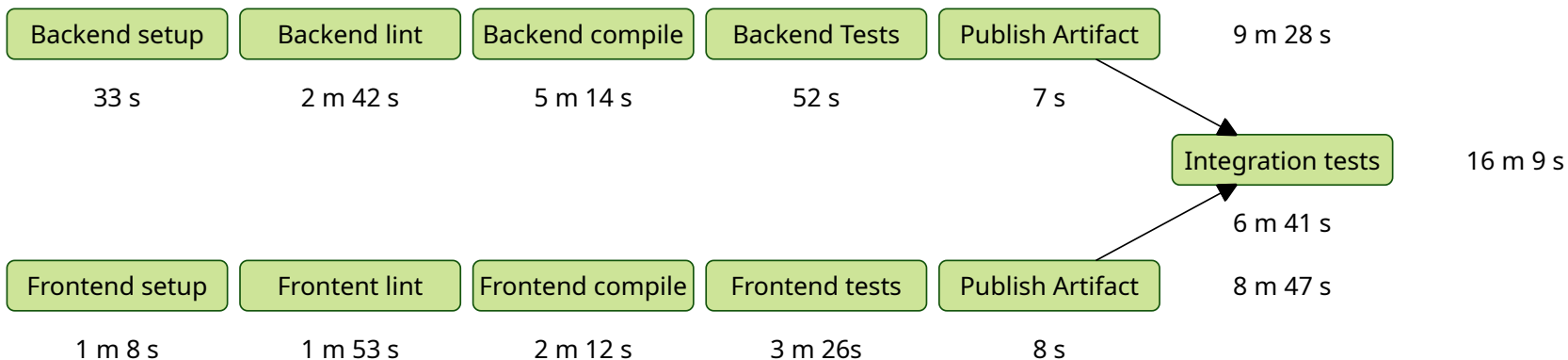
Antipattern #4: The One Pipeline



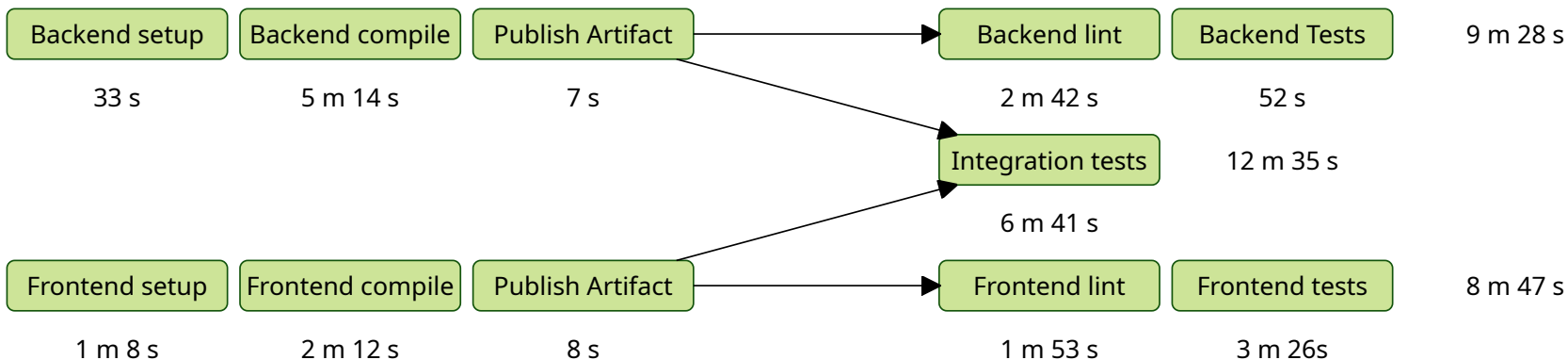
Pattern #4: Parallel jobs



Pattern #4: Parallel jobs



Pattern #4: Parallel jobs



Anti-pattern #5

Gift wrapping

Using wrapper tasks to execute command line tools

Kira ouf der Heide via Unsplash

Don’t

```yaml
# Build project
- uses: actions/run-cargo@9
  with:
    command: build
    target: release
    workspace: true
    all-features: true
```

Don’t

```yaml
# Build project
- uses: actions/run-cargo@9
  with:
    command: build
    target: release
    workspace: true
    all-features: true
    arguments: --jobs 8
```

Do

```yaml
# Build project
- run: >
    cargo build
    --target release
    --workspace
    --all-features
    --jobs 8
```

Do

```yaml
# Set up Python
- uses: actions/setup-python@v4
  with:
    python-version: 3.11

# Set up Poetry
- uses: Gr1N/setup-poetry@v8

# Install Python dependencies
- run: poetry install
```

Anti-pattern #6

It’s complicated

The pipeline is its own software project

Antipattern #6: It’s complicated

Jenkinsfile

```
@Library("pipeline-library@4.3")
import com.acme.pipeline.Pipeline

Pipeline.pipeline()
```

pipeline.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<pipeline>
    <testFramework>junit4</testFramework>
    <integrationTests>true</integrationTests>
    <javaVersion>1.8</javaVersion>
    <configurationDirectory>configurations/</configurationDirectory>
    <deploy branch="main">
        <server>production.acme.com</server>
    </deploy>
    <deploy branch="develop">
        <server>integration.acme.com</server>
    </deploy>
</pipeline>
```

Pattern #5: Composable pipeline library

```yaml
jobs:
  build_rust:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: factor10/actions/compile-rust@v1.2

      - uses: factor10/actions/test-rust@v1.2

      - uses: factor10/actions/publish-artifact@v1.2
        with:
          path: target/release/auth-server
```

Pattern #5: Composable pipeline library

```
jobs:
  build_rust:
    uses: factor10/workflows/rust-workflow@v2.3
    with:
      artifact_path: target/release/auth-server
```

Anti-pattern #6.5

Entangled

How do we get out of here?

Leo Bayard via Unsplash

Antipattern #6.5: Entangled

```
$ cat .github/**/*.yaml | wc -l
12858
```

Pattern #6: Scripting

```yaml
name: PR Checks
on: [pull_request]
jobs:
  CI:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      # Set up Python
      - uses: actions/setup-python@v4
        with:
          python-version: 3.9

      # Set up Poetry
      - uses: Gr1N/setup-poetry@v8

      # Init the project
      - run: make init

      # Run the CI checks
      - run: make ci
```

Pattern #6: Scripting

```
init:
  poetry install

lint:
  poetry run ruff check src tests

stylecheck:
  poetry run black --check src/ tests/

format:
  poetry run black src/ tests/

typecheck:
  MYPYPATH="src/" poetry run mypy --namespace-packages --explicit-package-bases src/

test:
  PYTHONPATH="src/" poetry run pytest --cov --cov-branch --cov-report term-missing

ci: stylecheck lint typecheck test
```

Anti-pattern #7

Groundhog day

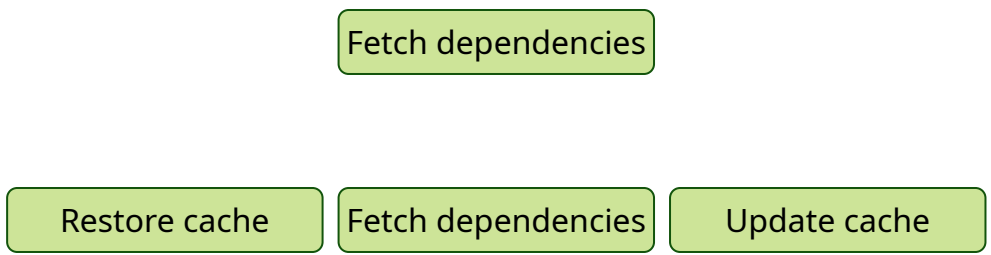
Starting from scratch all the time

Jimmy Conover via Unsplash

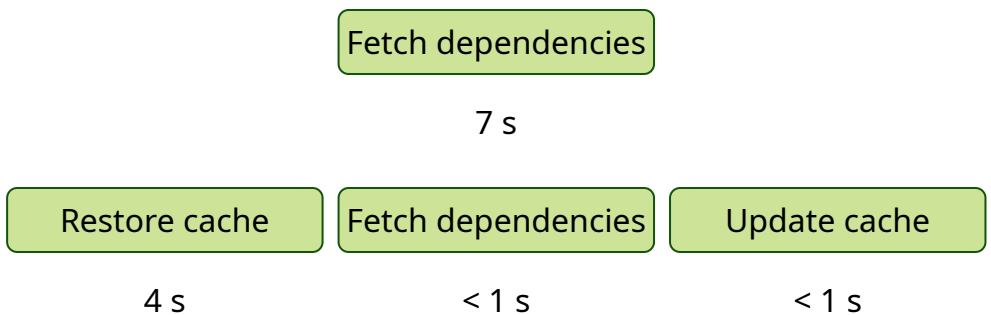
Antipattern #7: Groundhog day

Pattern #7

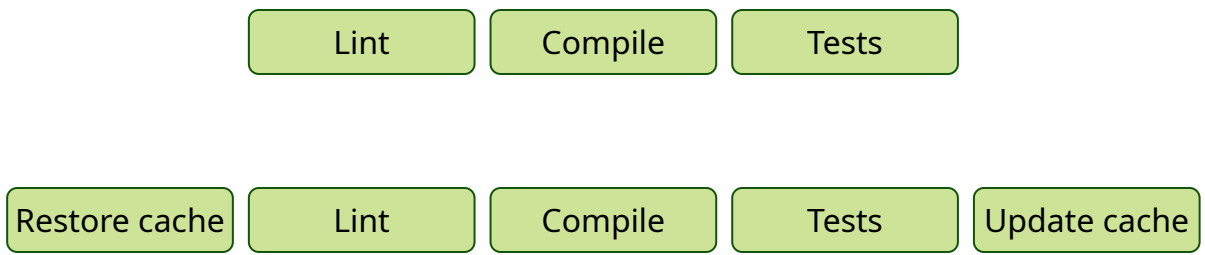
Caching dependencies



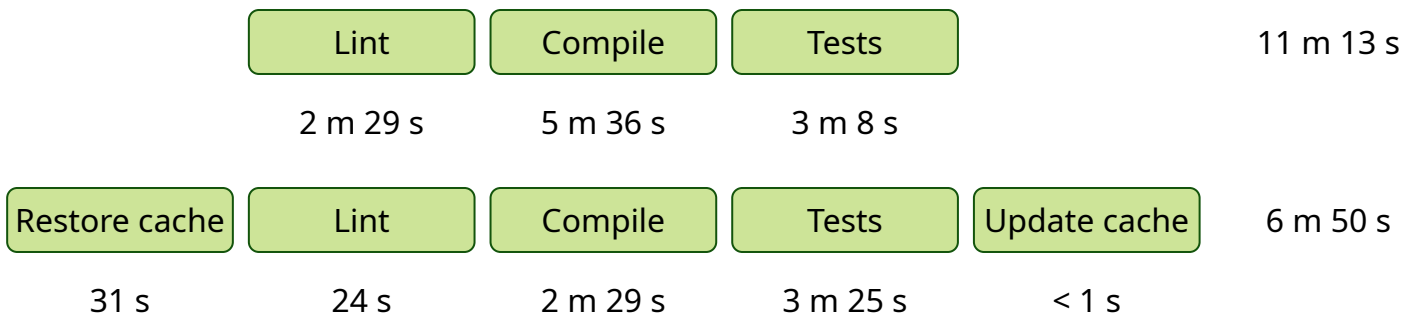
Caching dependencies



Caching build output



Caching build output

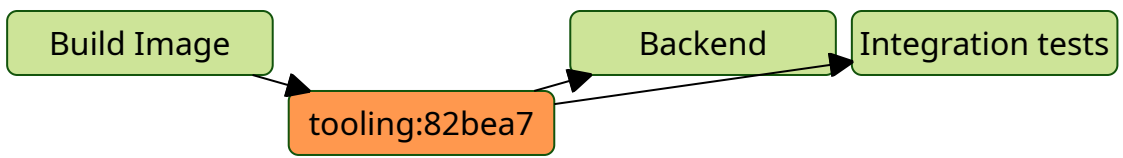


Caching container images

```
docker buildx build \
  --push \
  --tag my/image:latest \
  --cache-from type=registry,ref=my.registry/my/repo-cache:latest \
  --cache-to type=registry,ref=my.registry/my/repo-cache:latest \
  .
```

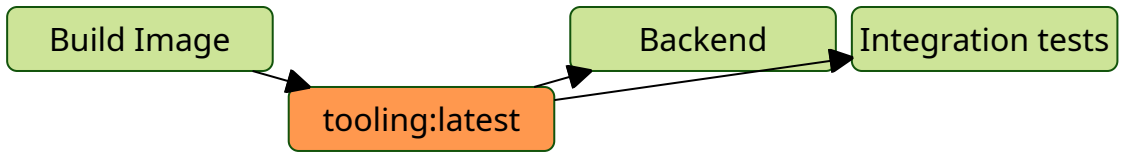
```
#7 importing cache manifest from my/repo-cache:latest
#7 inferred cache manifest type: application/vnd.oci.image.index.v1+json done
#7 DONE 1.2s

#8 [3/4] RUN cargo install --locked --git https://github.com/leptos-rs/cargo-leptos cargo-leptos
#8 extracting sha256:b0a0cf830b12453b7e15359a804215a7bcccd3788e2bcecff2a03af64bbd4df7 1.3s done
#8 extracting sha256:e62e6e54ca92f1c0e35a38988a00f7753207afcb1fc9106d95893324d7505fb5 4.3s done
#8 DONE 7.9s

#9 [4/4] RUN cargo install --locked cross
#9 0.092     Updating crates.io index
#9 0.360  Downloading crates ...
#9 0.681   Downloaded cross v0.2.5
#9 0.713   Installing cross v0.2.5
```

Pattern #8: Tooling images

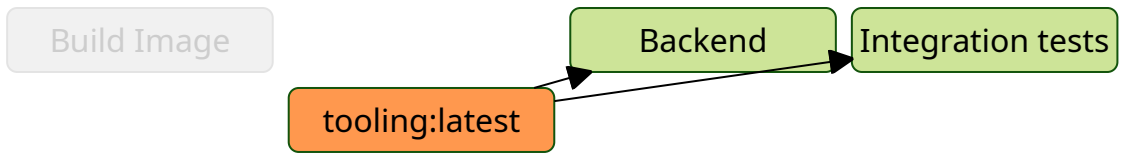


Pattern #8: Tooling images

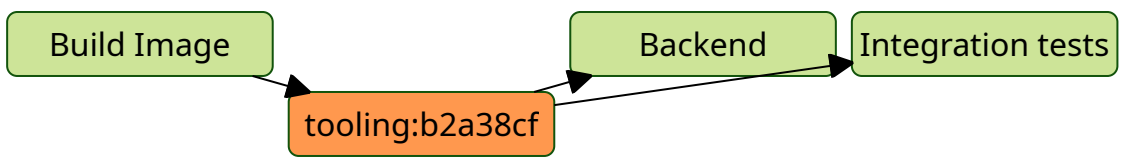
main



PR #387: Add index on user birthdates



PR #392: Upgrade cross in the build pipeline



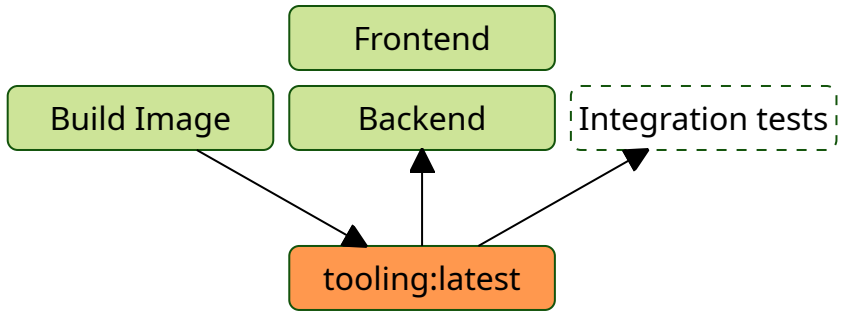
Anti-pattern #8

Interference

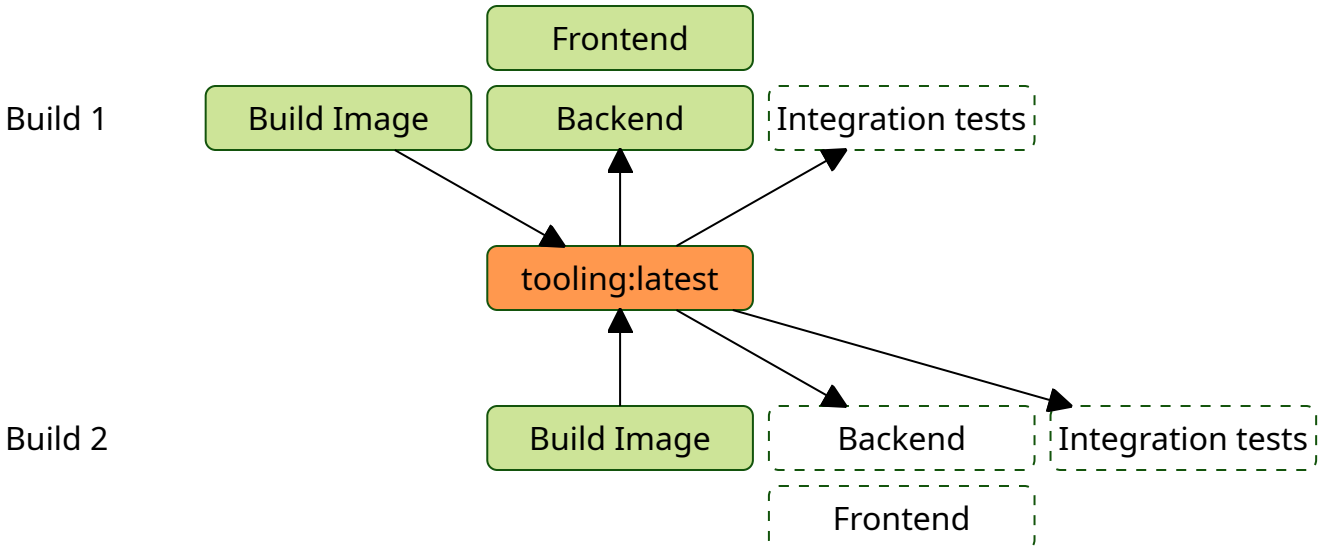
Colliding with other pipelines

Uriel Soberanes via Unsplash

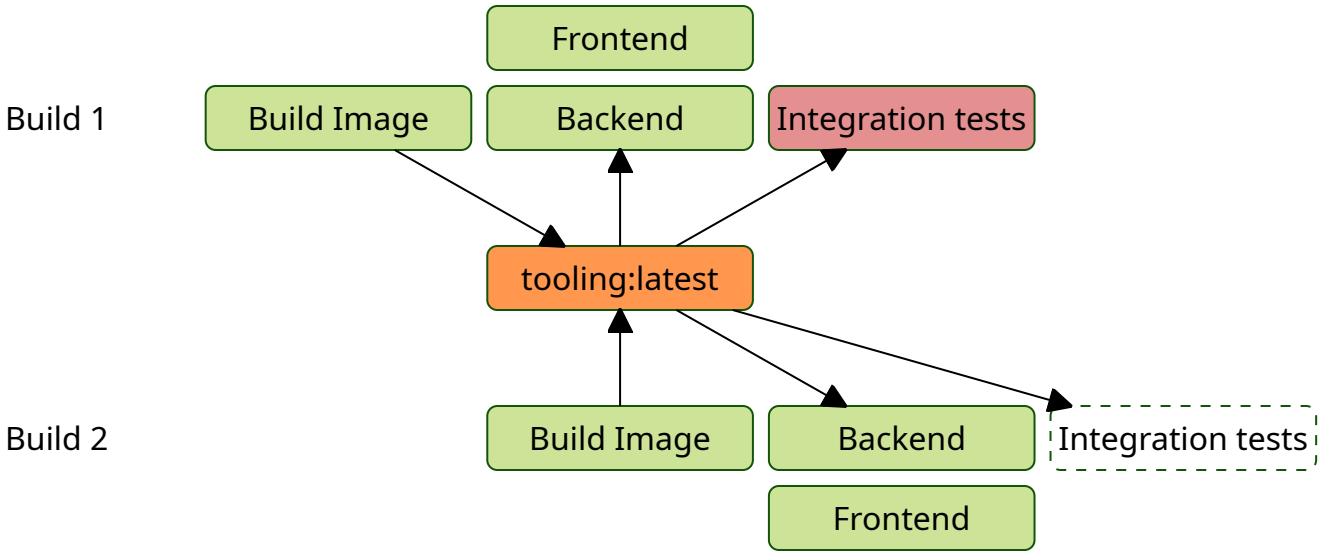
Antipattern #8: Interference



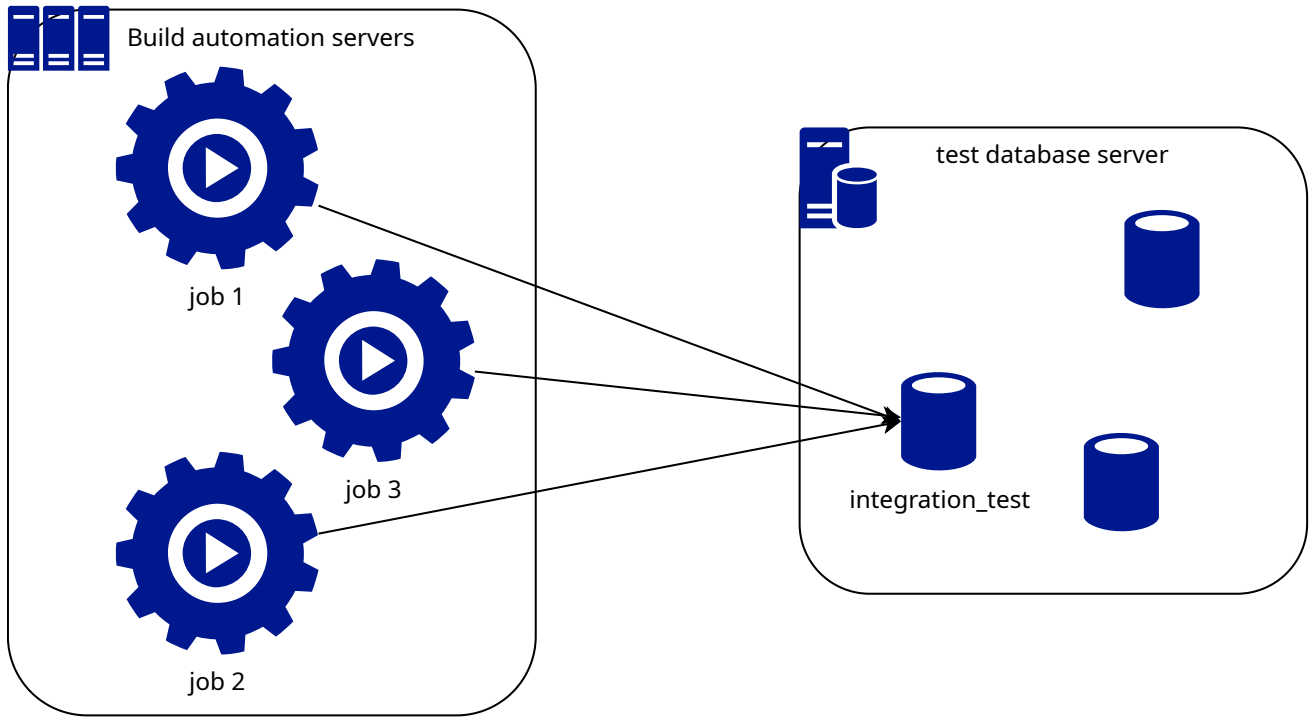
Antipattern #8: Interference



Antipattern #8: Interference



Antipattern #8: Interference



Anti-pattern #9

Overloaded

Running on underpowered machines

Jainath Ponnala via Unsplash

Raniz’ rules-of-thumb for build agent sizing

1. If the build is slower on the build agents than on developer machines: beef up the build agents.
2. If the build fails because of a lack of resources: beef up the build agents.
3. If you have issues with queue times, get more servers
4. If the available build agent configurations aren’t sufficient: roll your own.

Should you run your own agents?

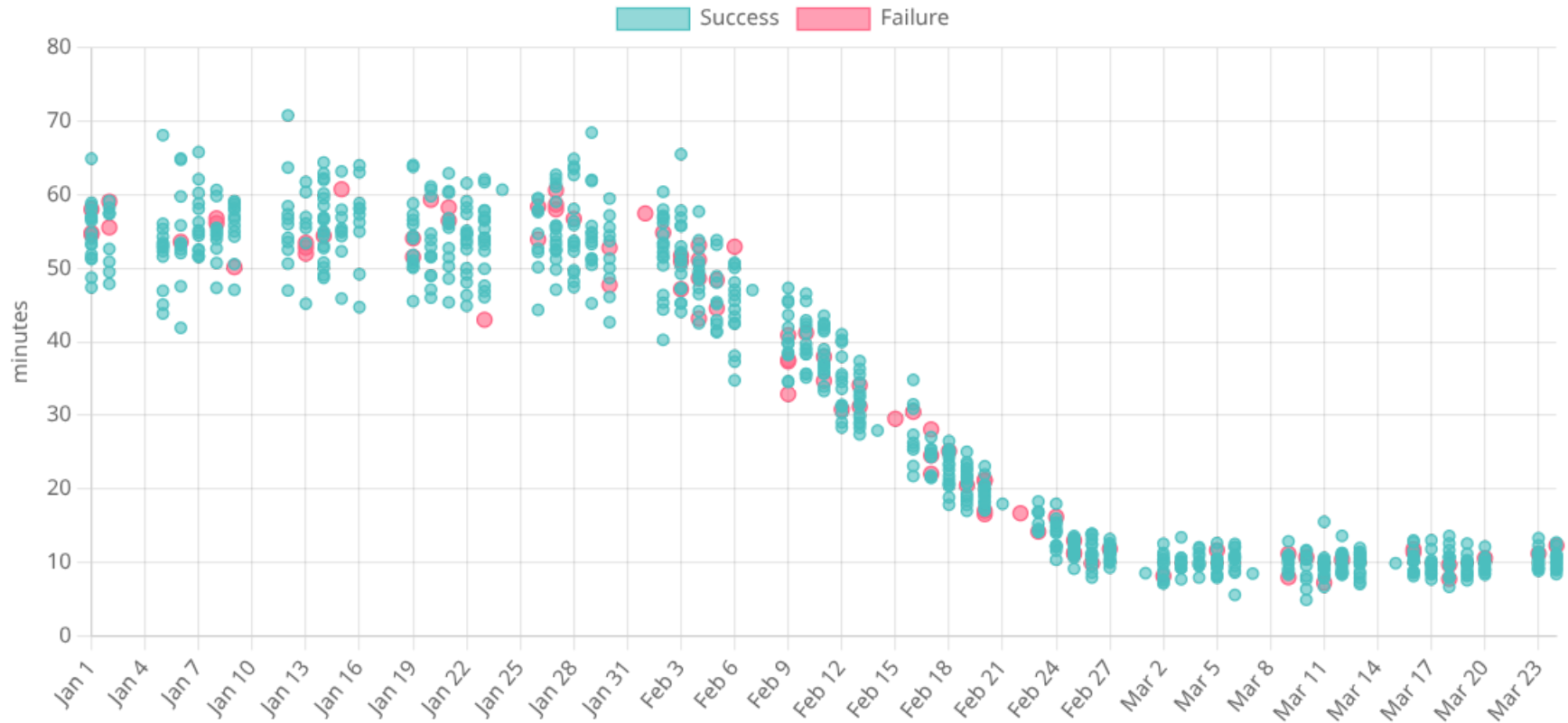
Pros

+ Full control
+ Right-sizing
+ Cost

Cons

- Maintenance
- Scaling
- Cost

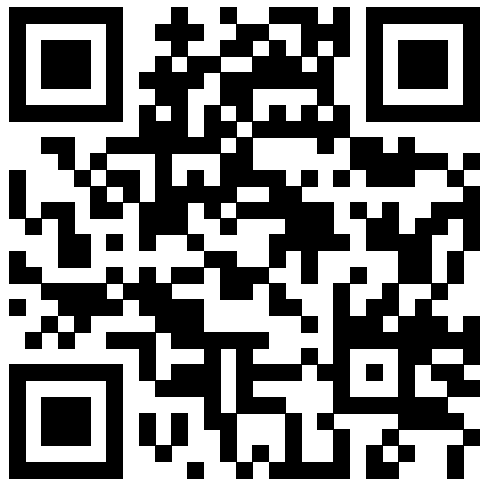
Is it worth the effort?



factor10

Pipeline patterns and anti-patterns

Things your pipeline should (not) do



Daniel Raniz Raneland
Coding Architect @ factor10

factor10.com
raniz@factor10.com
raniz@mastodon.online
about.me/raniz
raniz.blog
/in/raneland