Escaping integration testing hell with Testcontainers

Jure Repe
Software Engineer

OUTFIT7

Hello there 👋

Software engineer @ Outfit7

Backend department

Cloud infrastructure, build tools, internal tooling*

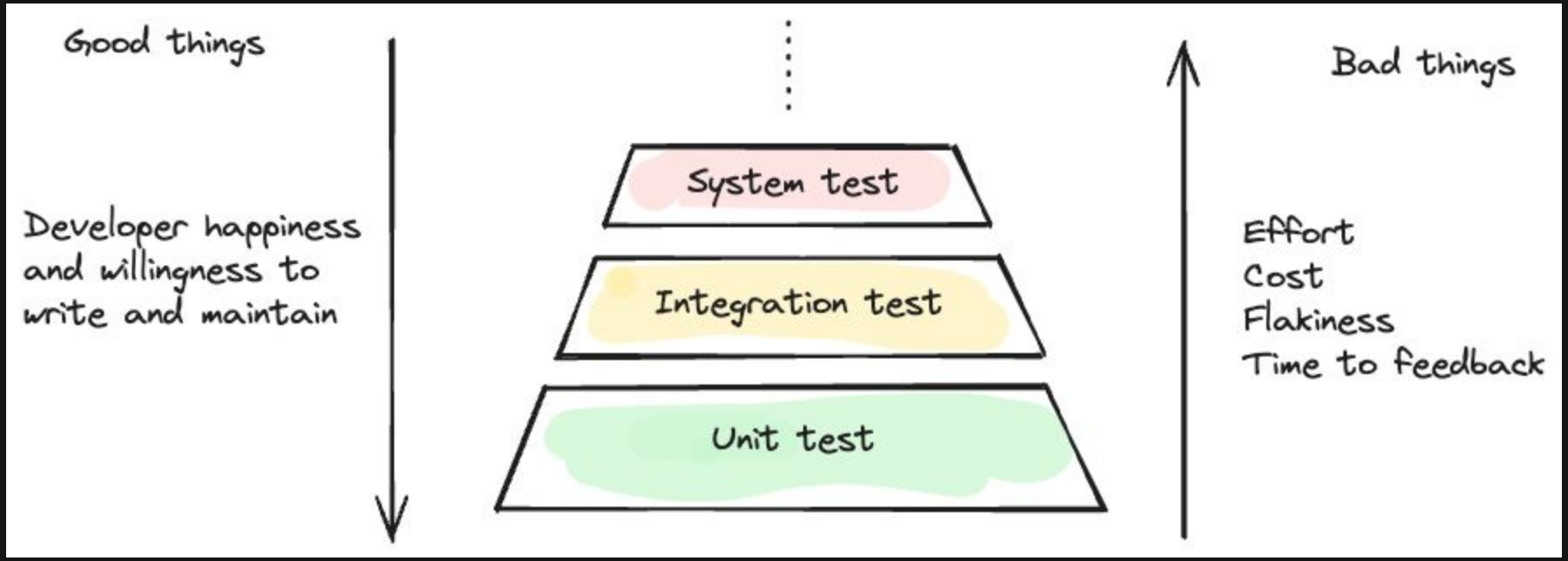
* Improving existing processes for easier development

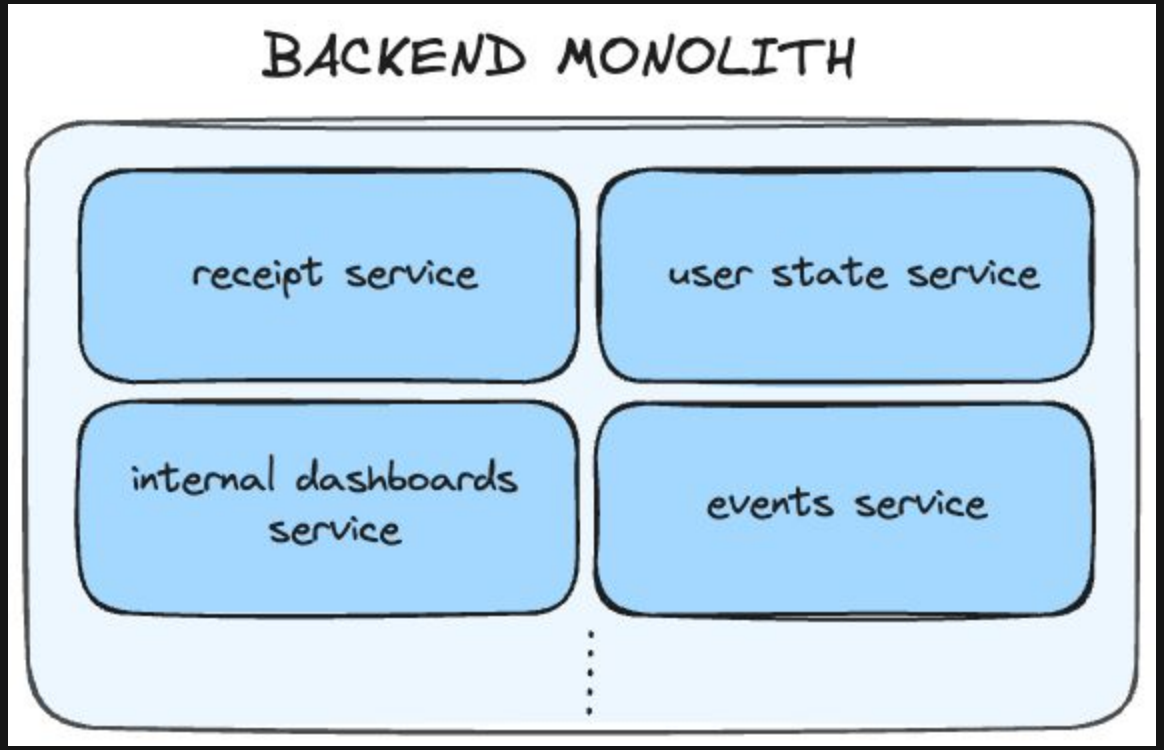
Existing integration testing infrastructure

Testcontainers & hope for improvement

Testcontainers in practice

Part 1: Existing infrastructure


Good things
Developer happiness and willingness to write and maintain
System test
Integration test
Unit test
Bad things
Effort
Cost
Flakiness
Time to feedback


BACKEND MONOLITH
receipt service
user state service
internal dashboards service
events service

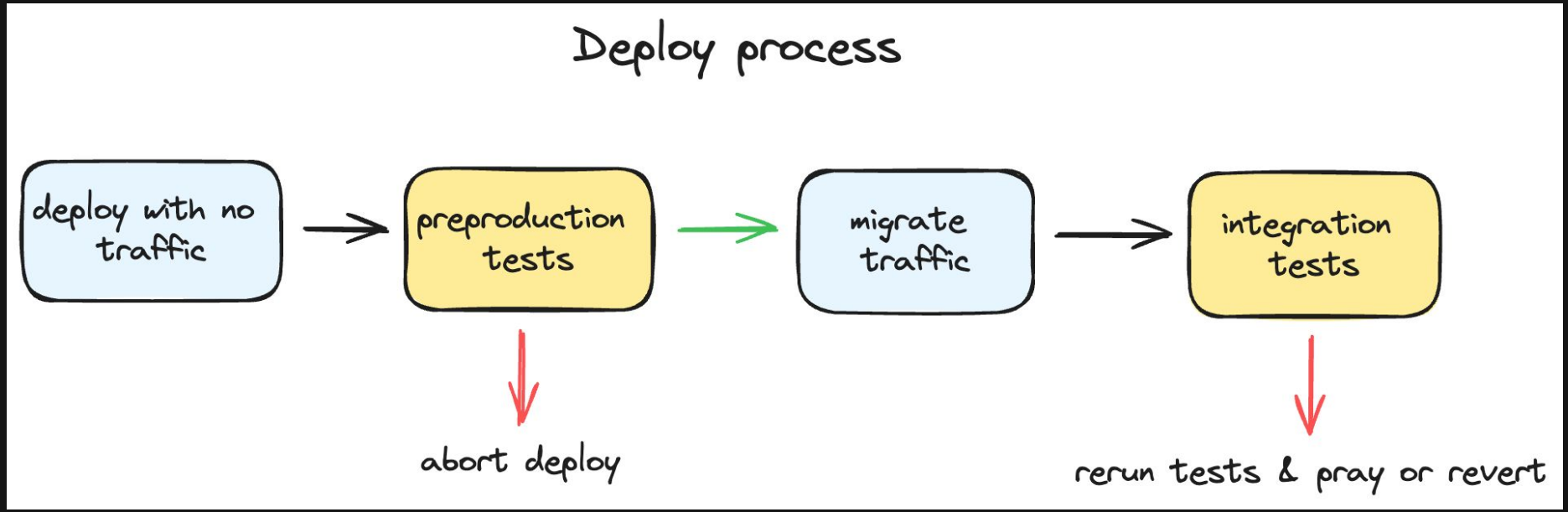
Integration tests: version 1

Groovy & Spock framework, separate repository

Can be triggered on-demand during development

Runs before traffic migration and after deploy *

* In the darkest of times, we ran them every 3 hours


Deploy process
deploy with no traffic
preproduction tests
migrate traffic
integration tests
abort deploy
rerun tests & pray or revert

Integration tests: version 1

Is written in Groovy and Spock - super specific

Requires full application deploy - time consuming

Flakier than corn flakes - eventual consistency, countless combinations

Integration tests: version 2

Kotlin, same repository

Can be triggered on-demand during development

Runs before traffic migration and after deploy

Integration tests: version 2

Is written in Kotlin - specific, ...AGAIN

Requires full application deploy - time consuming, ...AGAIN

Flakier than corn flakes - eventual consistency, ...AGAIN

Huge backlog of tests to rewrite - ain't nobody got time for that

Integration tests: version 2

Same repository, so easier change tracking

Moved away from unfamiliar framework (Spock)

Let’s evaluate

Instead of one, now there are two

Tests are hard to write

No confidence in tests

No clear owner of the project

There was another underlying problem...

Part 2: Enter Testcontainers

Testcontainers?

Testcontainers is an open source framework for providing throwaway, lightweight instances of databases, message brokers, web browsers, or just about anything that can run in a Docker container

If you can mash it into a container, you can run it.

What it hopefully solves

Easily provision 3rd party services we need in containers

Fast, simple, familiar workflow - just like unit tests

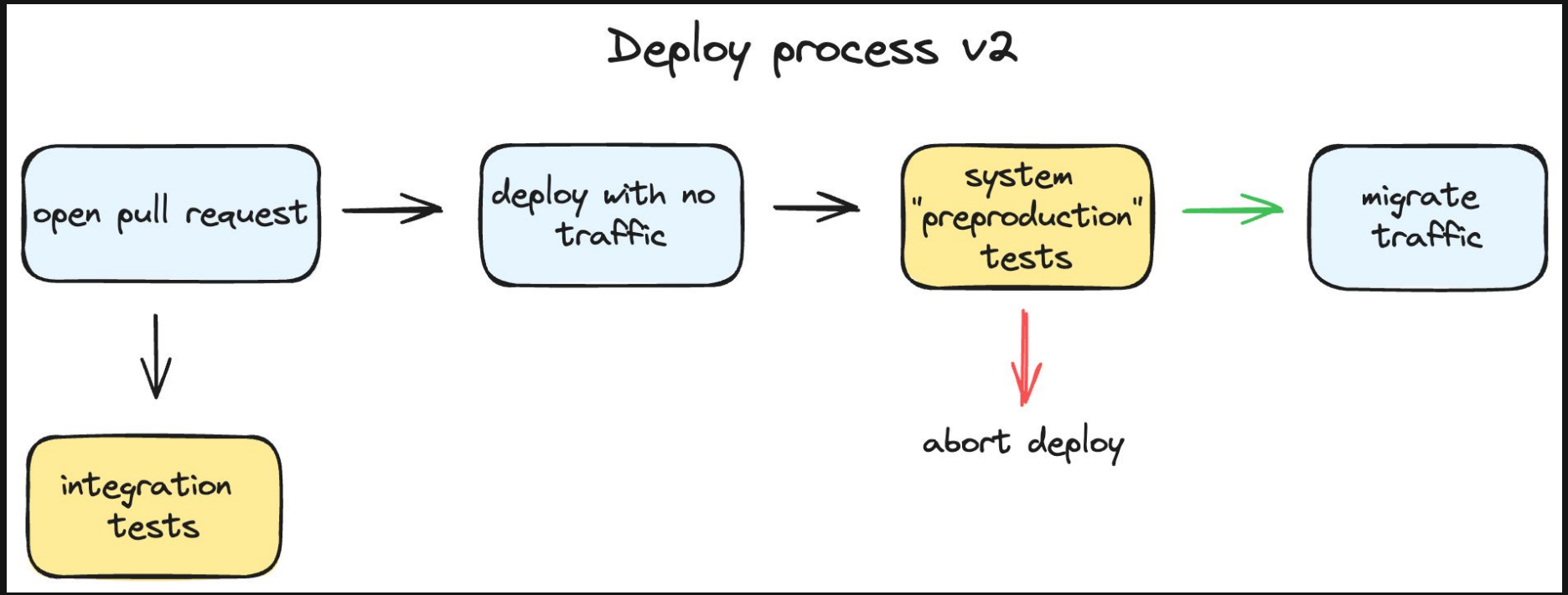
Controlled environment

Migrating is still challenging

Bending the will of the monolith is hard

We still have a large backlog of tests to be rewritten

Preloading data can be a pain


Deploy process v2
open pull request
deploy with no traffic
system "preproduction" tests
migrate traffic
integration tests
abort deploy

Comparison with v1 & v2

Language & framework we don't use - good old Java and jUnit

Hard to write and maintain - much easier, like unit tests

Flakiness ? - a more controlled environment

Time consuming process - fast, quick feedback loop

Huge backlog of tests to rewrite - same as before

Part 3: Testcontainers in practice

(ノ °□°)ノ ⌒ ┻━━┻

Key takeaways

Minimal friction when writing integration tests

Fast feedback loop

Confidence in tests is key to success

Testcontainers project is awesome :)

Jure Repe @ LinkedIn

jure.repe@outfit7.com

Breaking free at last,
Containers spin, tests now pass—
Integration peace.