Testcontainers in the Real World

Ein Erfahrungsbericht mit Lessons Learned

Speaker

Fabian Gotzen

Software Engineer
@ bLink

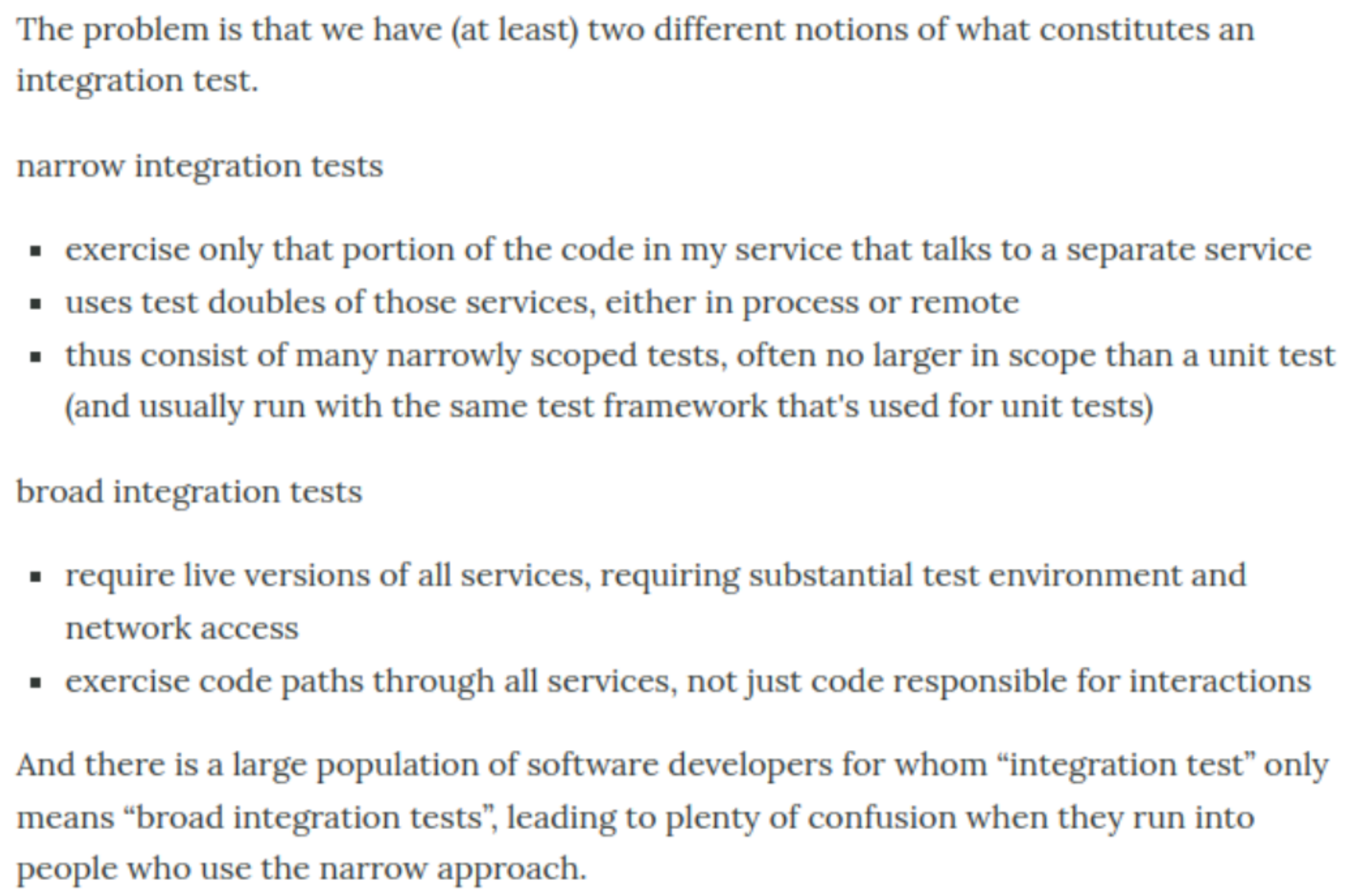
SIX BBS

- Java / Kotlin
- Spring
- Devops

Agenda

Common Ground

- Integration test?
- Broad vs narrow



The problem is that we have (at least) two different notions of what constitutes an integration test.

narrow integration tests

- exercise only that portion of the code in my service that talks to a separate service
- uses test doubles of those services, either in process or remote
- thus consist of many narrowly scoped tests, often no larger in scope than a unit test (and usually run with the same test framework that's used for unit tests)

broad integration tests

- require live versions of all services, requiring substantial test environment and network access
- exercise code paths through all services, not just code responsible for interactions

And there is a large population of software developers for whom “integration test” only means “broad integration tests”, leading to plenty of confusion when they run into people who use the narrow approach.

Introduction to Testcontainers

Testcontainers

Java Go .NET Node.js Python Rust Haskell Ruby Clojure

Star 7.4k

- Open source library
- Provides throwaway instances in containers
- No more mocks or complicated environment configurations
- Define your test dependencies as code

Testcontainers
Community Champions

Josh Long

The first Spring Developer Advocate since 2010. Josh is a Java Champion, author of 7 books (including "Reactive Spring"), open-source contributor, Youtuber, and a podcaster ("A Bootiful Podcast").

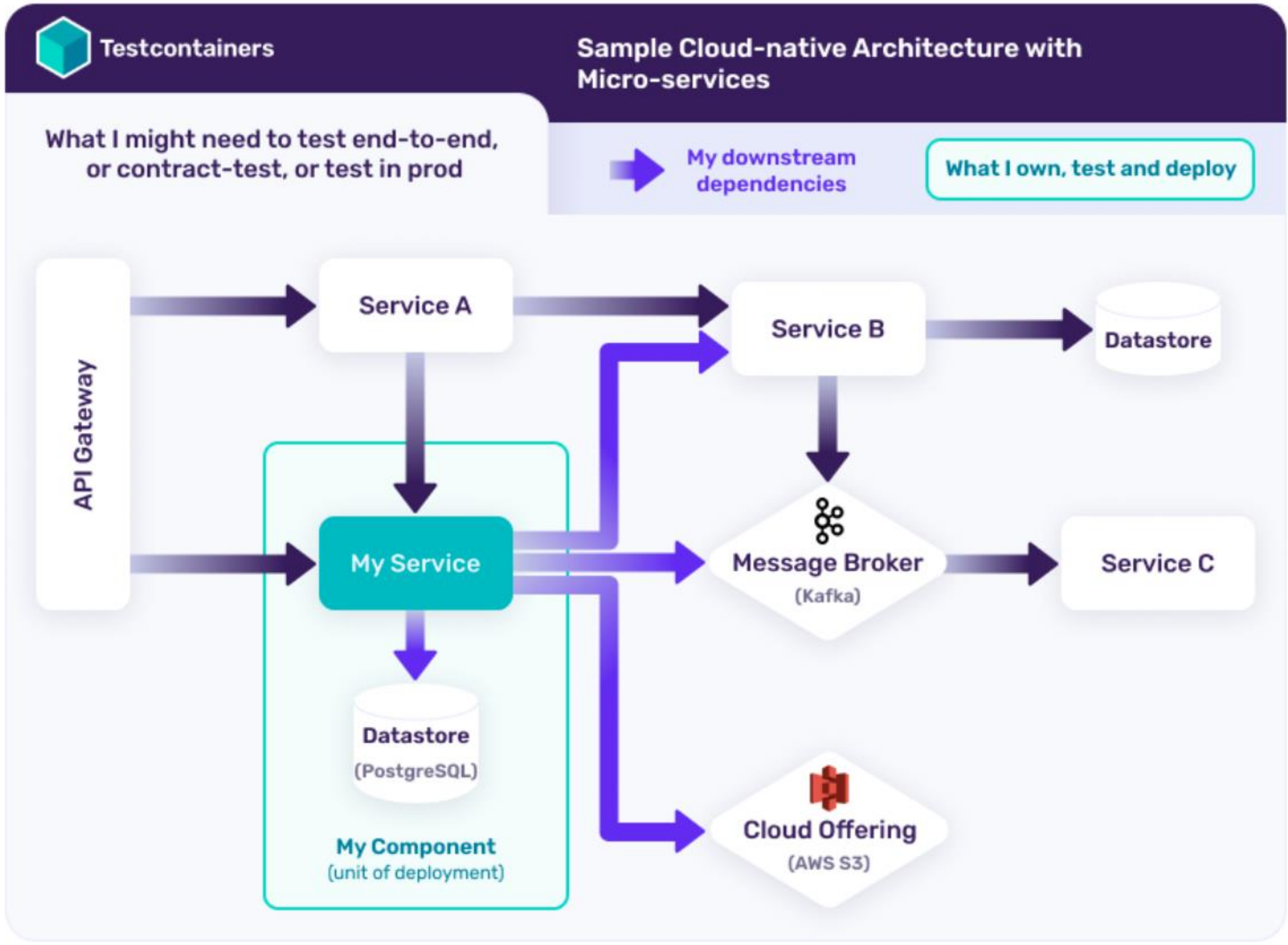
Testcontainers brought to you by

ATOMICJAR

What Problems Does Testcontainers Solve?

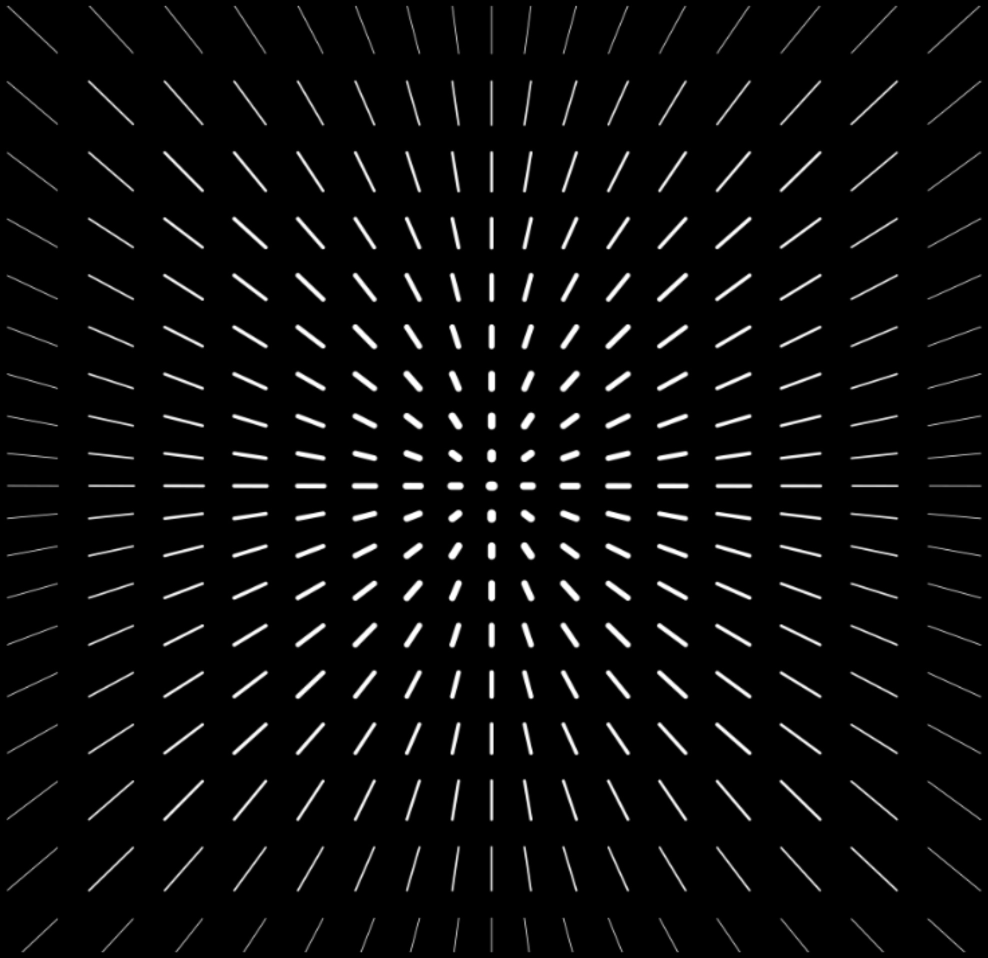
Before running tests...

- up and running
- desired state
- problems with shared resources (non-deterministic, data corruption, configuration drift)


Testcontainers
Sample Cloud-native Architecture with Micro-services
What I might need to test end-to-end, or contract-test, or test in prod
My downstream dependencies
What I own, test and deploy
API Gateway
Service A
Service B
Datastore
My Service
Message Broker
(Kafka)
Service C
Datastore
(PostgreSQL)
My Component
(unit of deployment)
Cloud Offering
(AWS S3)

Chapter 1 →

Why?



External Factors

- New service
- Message broker integration

Goals

01 Test close to production

02 Testing made easy

03 Declarative setup

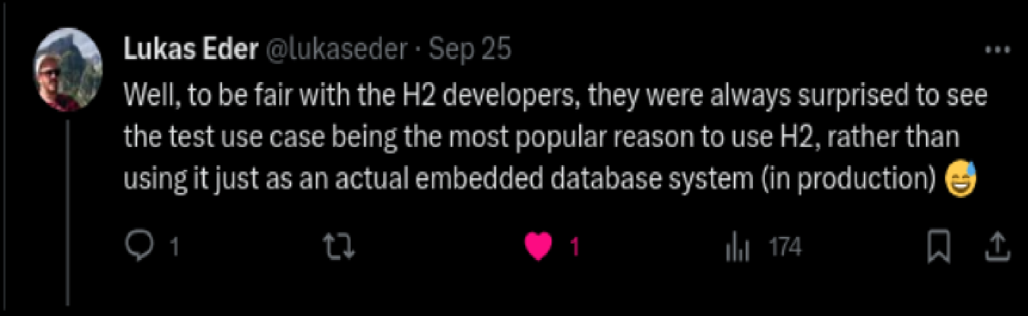
04 Parallel execution

Starting Situation

- In-memory solutions (H2, embedded Kafka)
- Test system (shared Oracle DB)

In-memory (H2)

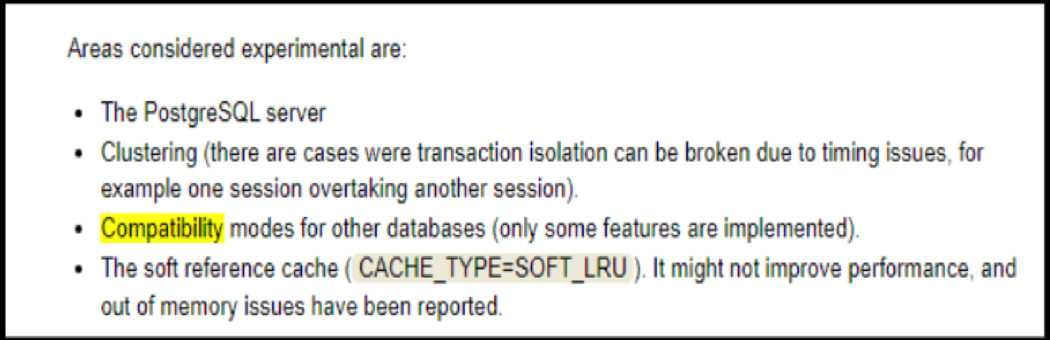
+ Fast & simple
+ No Docker required
- Different behaviour & syntax



Lukas Eder @lukaseder · Sep 25

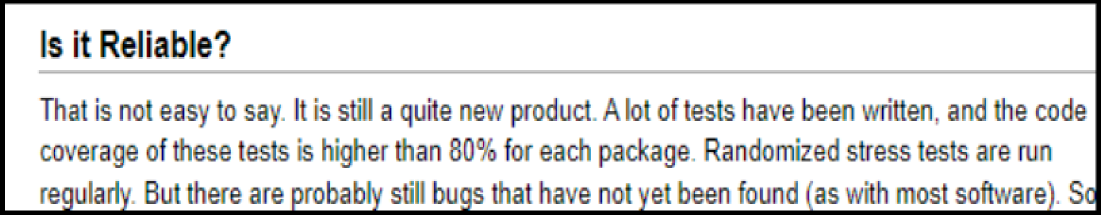
Well, to be fair with the H2 developers, they were always surprised to see the test use case being the most popular reason to use H2, rather than using it just as an actual embedded database system (in production) 😅

1 1 174



Areas considered experimental are:

- The PostgreSQL server
- Clustering (there are cases were transaction isolation can be broken due to timing issues, for example one session overtaking another session).
- Compatibility modes for other databases (only some features are implemented).
- The soft reference cache (`CACHE_TYPE=SOFT_LRU`). It might not improve performance, and out of memory issues have been reported.

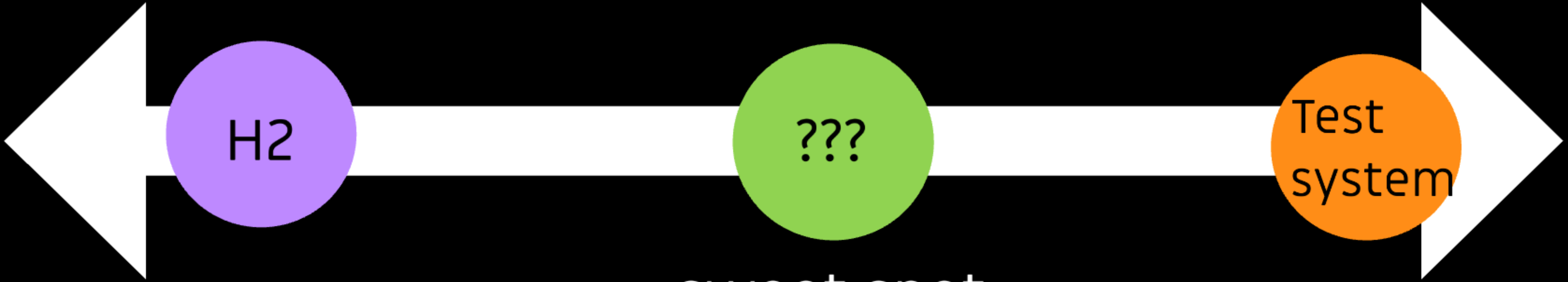


Is it Reliable?

That is not easy to say. It is still a quite new product. A lot of tests have been written, and the code coverage of these tests is higher than 80% for each package. Randomized stress tests are run regularly. But there are probably still bugs that have not yet been found (as with most software). So

Test system (Oracle)

+ As real as it gets
- Setup & maintenance
- Initial state
- Parallel execution on shared resource



sweet spot

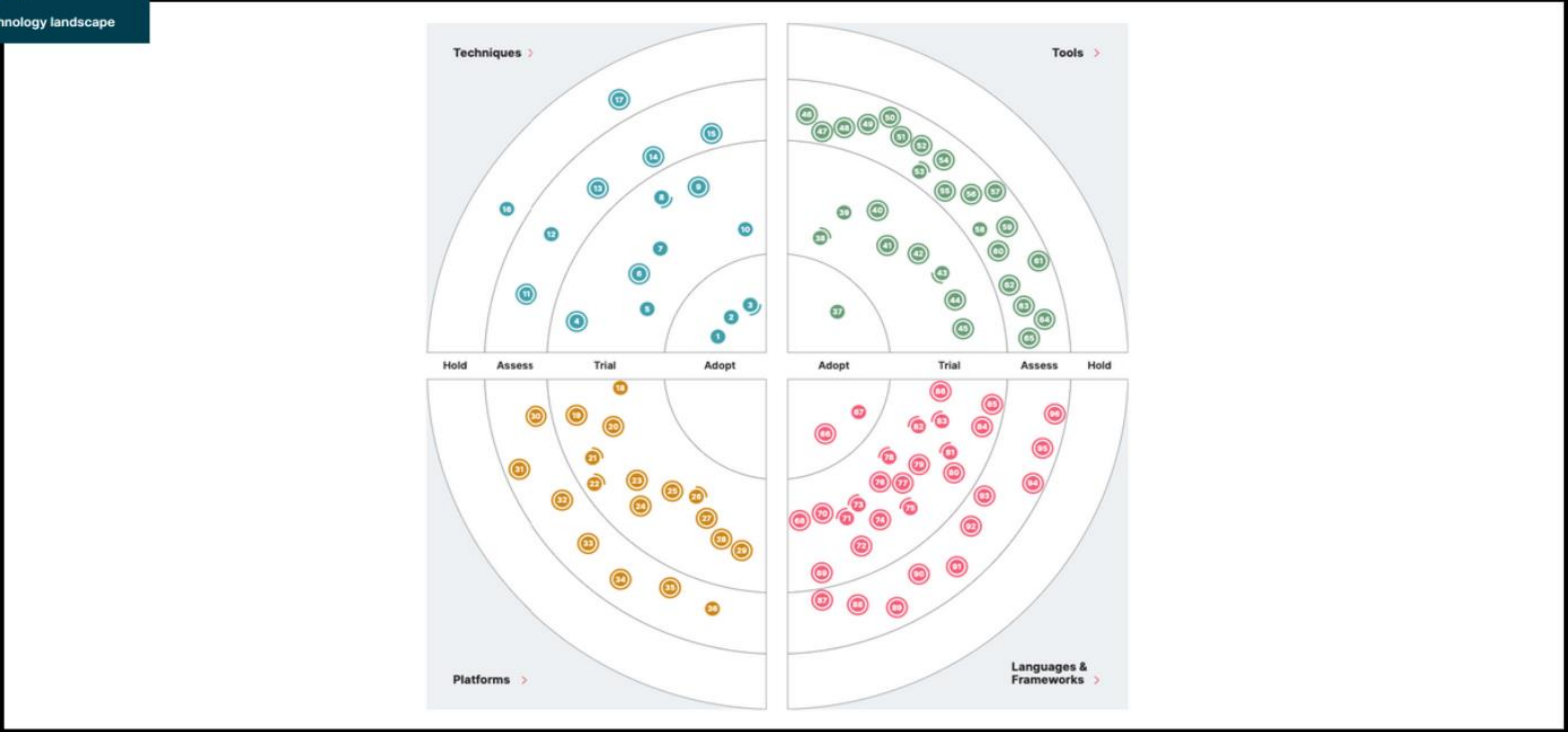
fast & simple
different technology

close to production
rigid

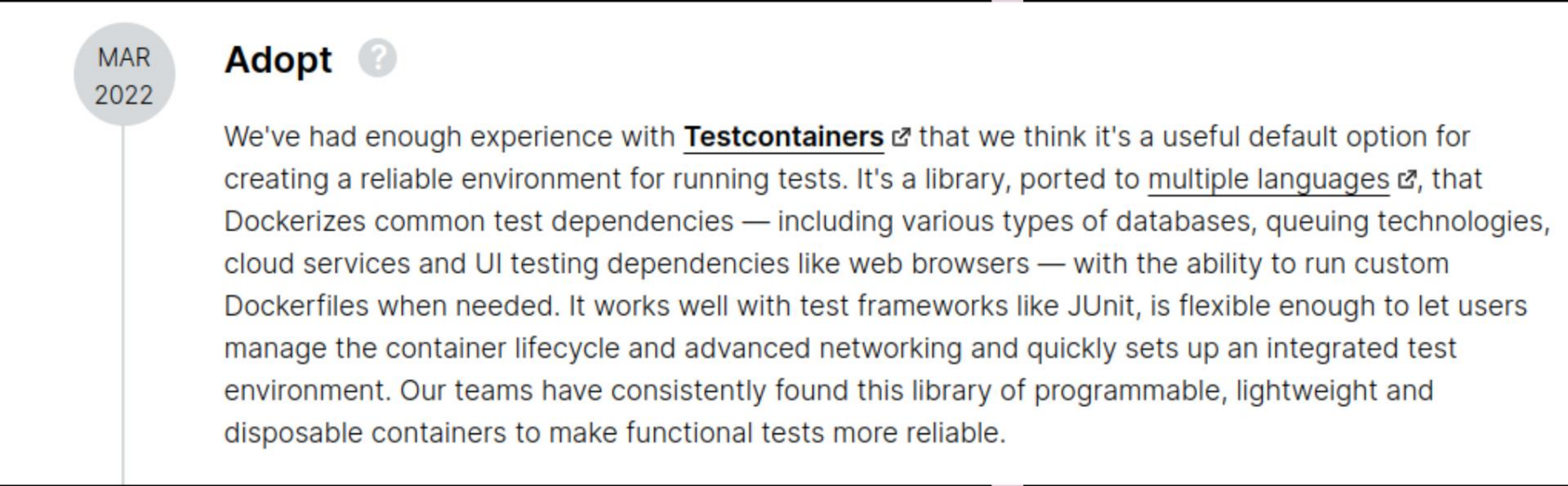
In search of a solution...

Technology Radar

An opinionated guide to today's technology landscape



https://www.thoughtworks.com/radar



MAR
2022

Adopt

We've had enough experience with **Testcontainers** that we think it's a useful default option for creating a reliable environment for running tests. It's a library, ported to multiple languages, that Dockerizes common test dependencies — including various types of databases, queuing technologies, cloud services and UI testing dependencies like web browsers — with the ability to run custom Dockerfiles when needed. It works well with test frameworks like JUnit, is flexible enough to let users manage the container lifecycle and advanced networking and quickly sets up an integrated test environment. Our teams have consistently found this library of programmable, lightweight and disposable containers to make functional tests more reliable.

Testcontainers

+ Consistent environment
+ Isolated
+ Flexibility (services)
- Overhead
- Complexity

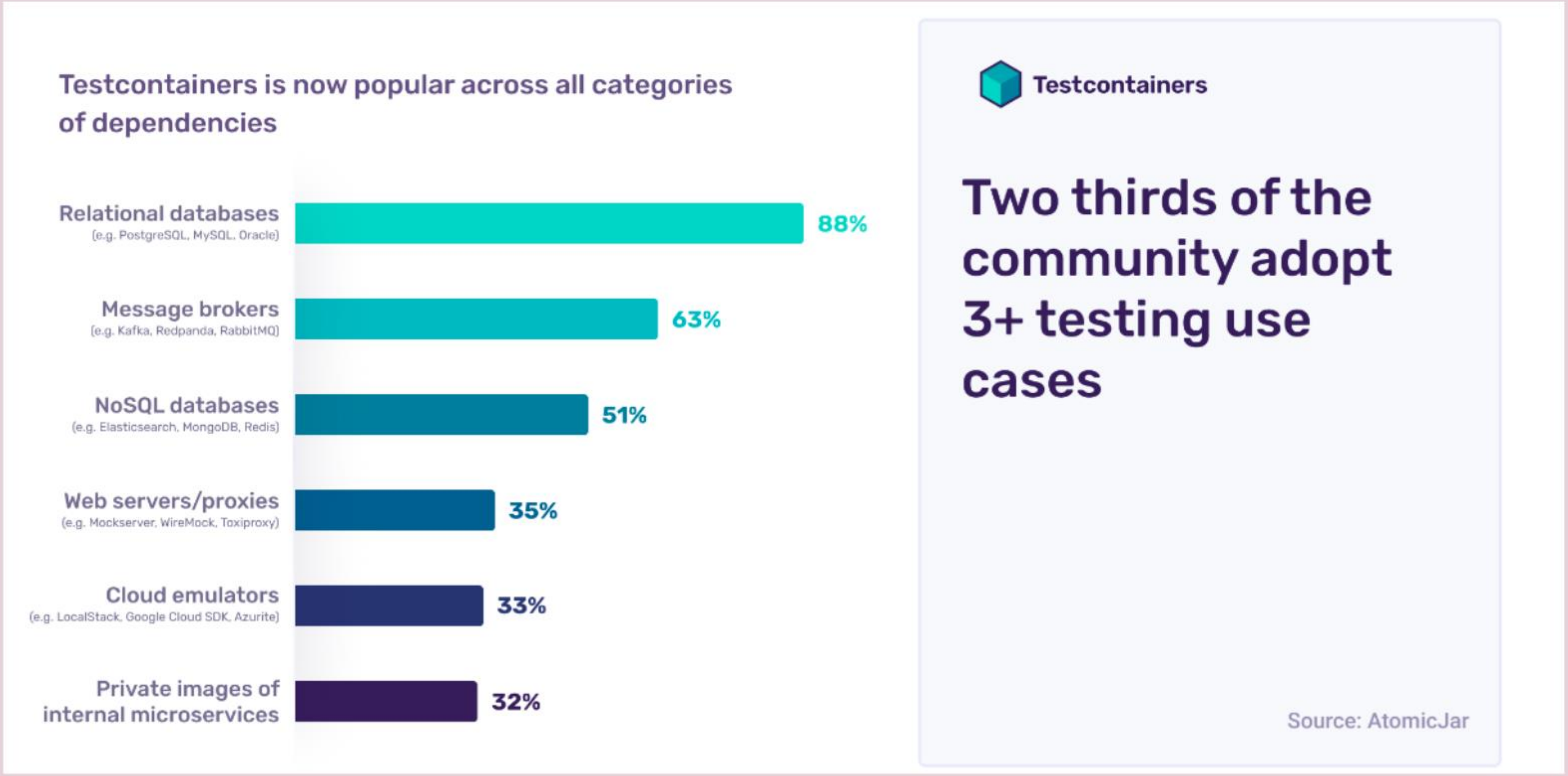
AtomicJar Testcontainers

REPORT

State of Local Development and Testing 2023

By Oleg Šelajev

Dependencies

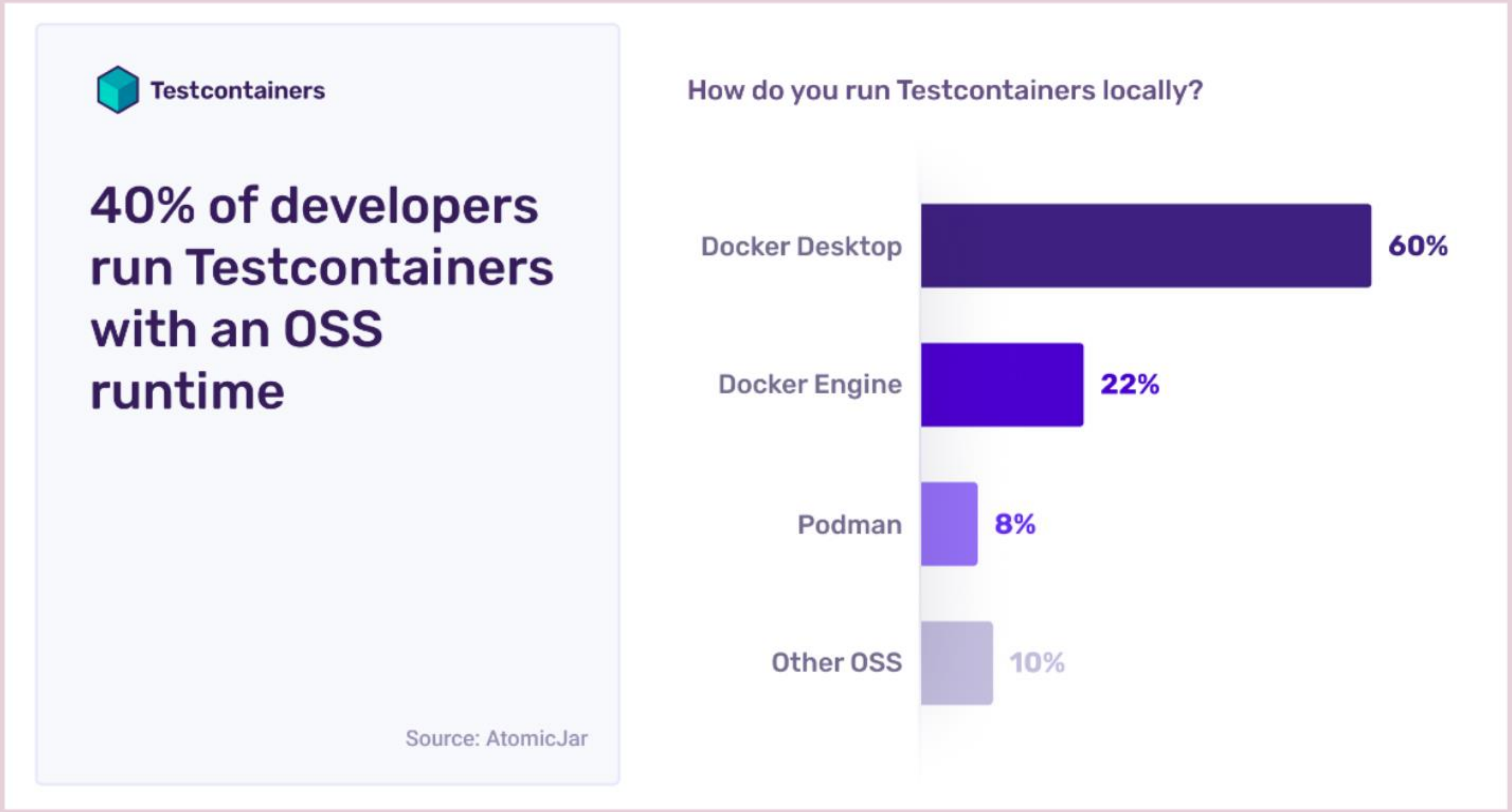

Testcontainers is now popular across all categories of dependencies

Testcontainers

Two thirds of the community adopt 3+ testing use cases

Source: AtomicJar

OSS



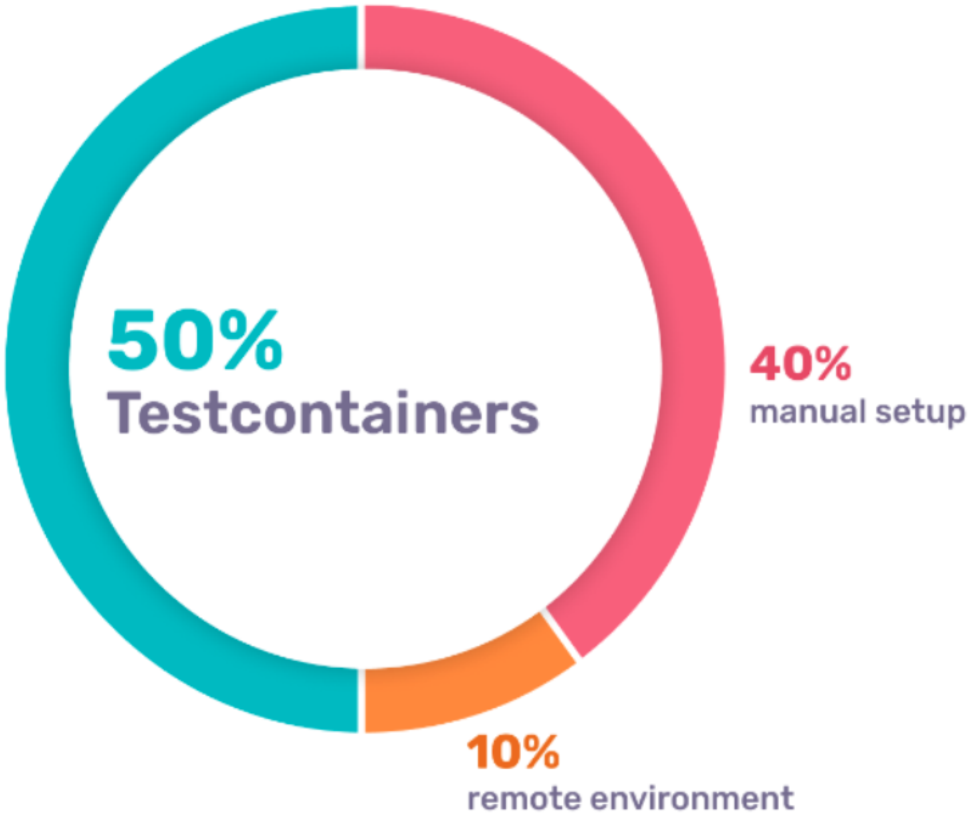
Testcontainers

40% of developers run Testcontainers with an OSS runtime

Source: AtomicJar

Dev-Time

How do you create the environment for running your project locally during development?



Testcontainers

Not just tests: 50% of developers use Testcontainers at dev-time

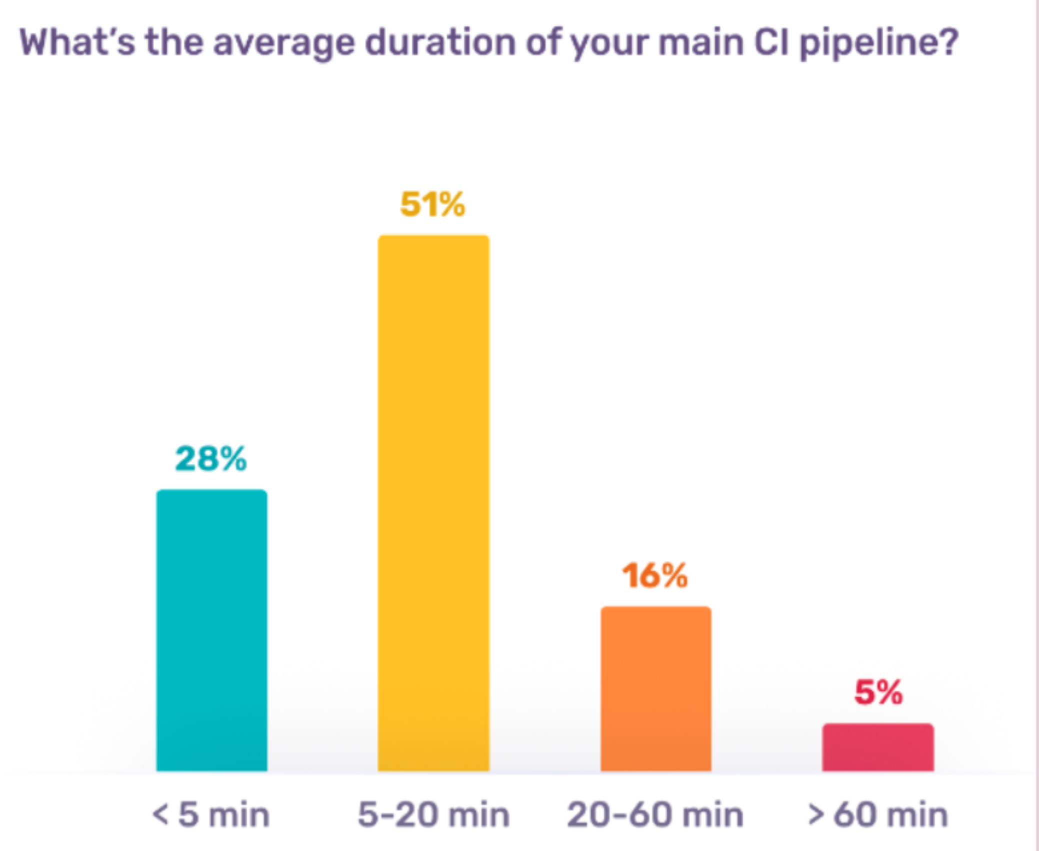
Source: AtomicJar

Pipeline

Testcontainers

79% of Testcontainers-enabled CI pipelines run under 20 min

Source: AtomicJar

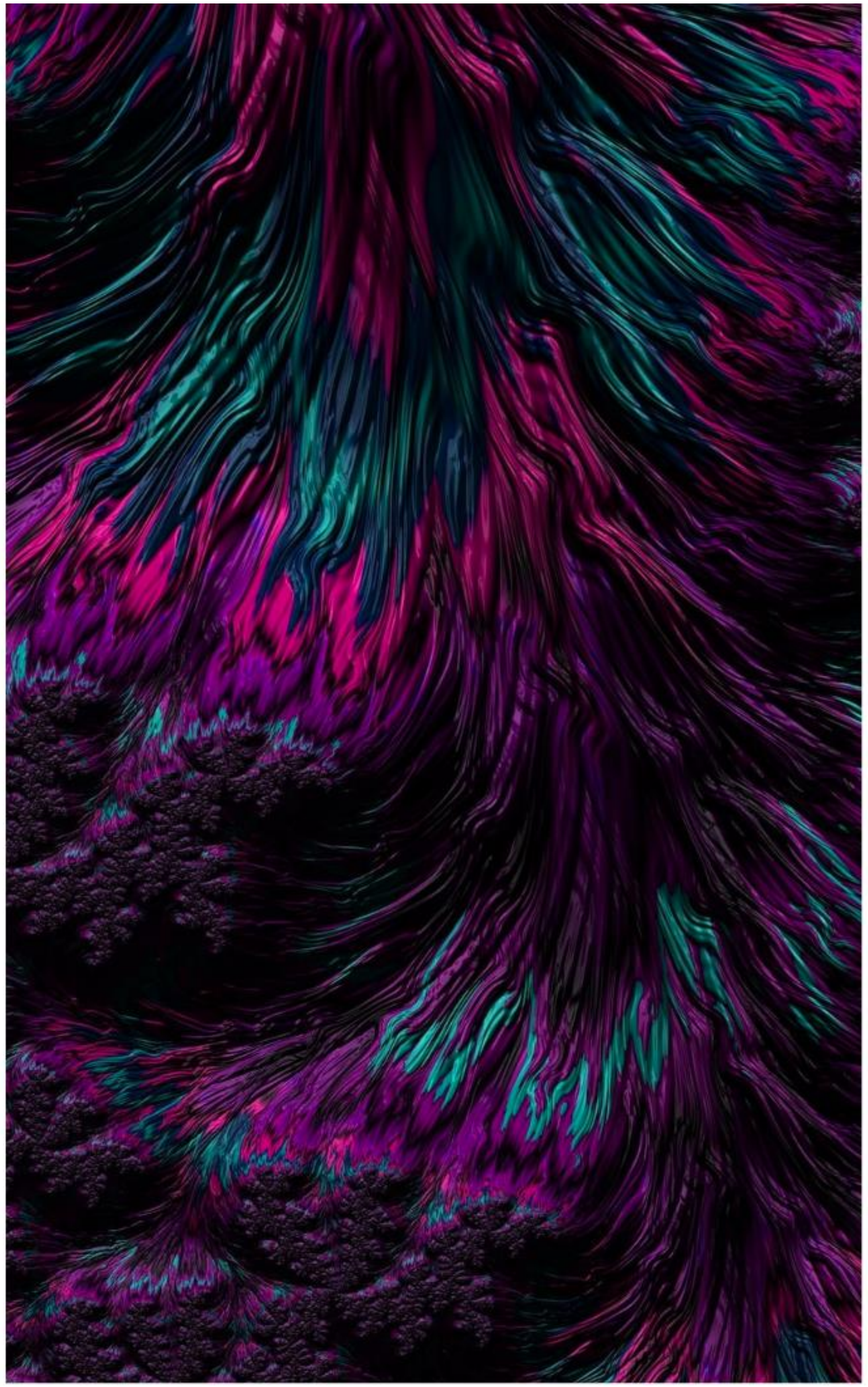


Chapter 2

The Good, the Bad and the Ugly

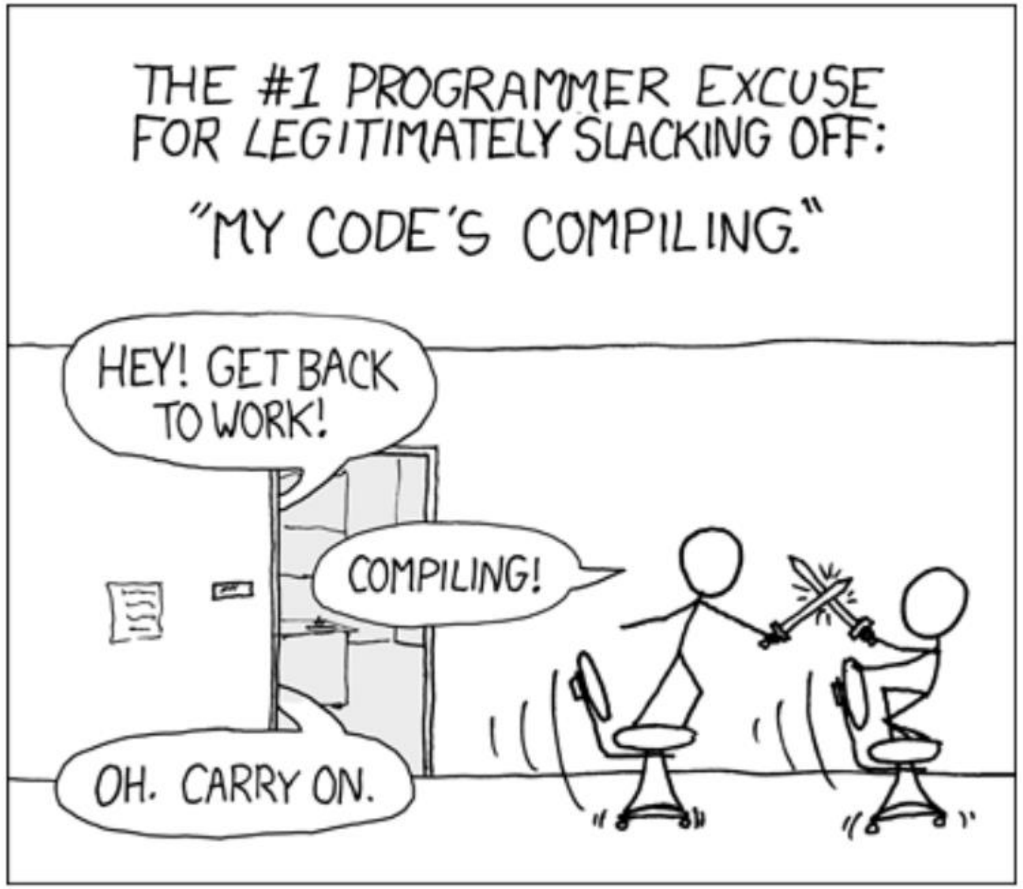
Introduction & Setup

- Textbook example



Compiling

|< < Prev Random Next > >|

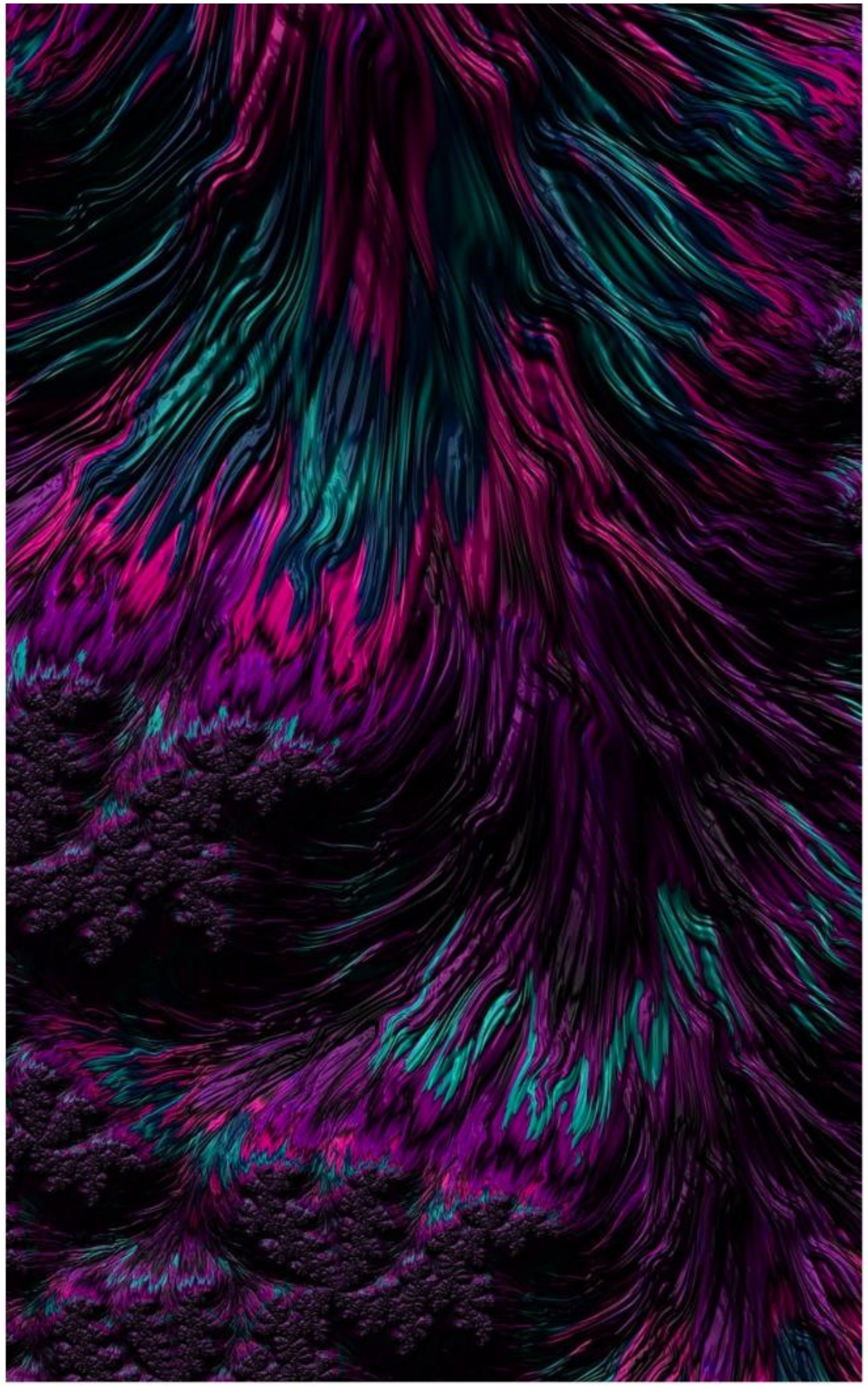


|< < Prev Random Next > >|

Permanent link to this comic: https://xkcd.com/303/
Image URL (for hotlinking/embedding): https://imgs.xkcd.com/comics/compiling.png

Lifecycle Management

- Singleton container
- Started once for several test classes



State Management

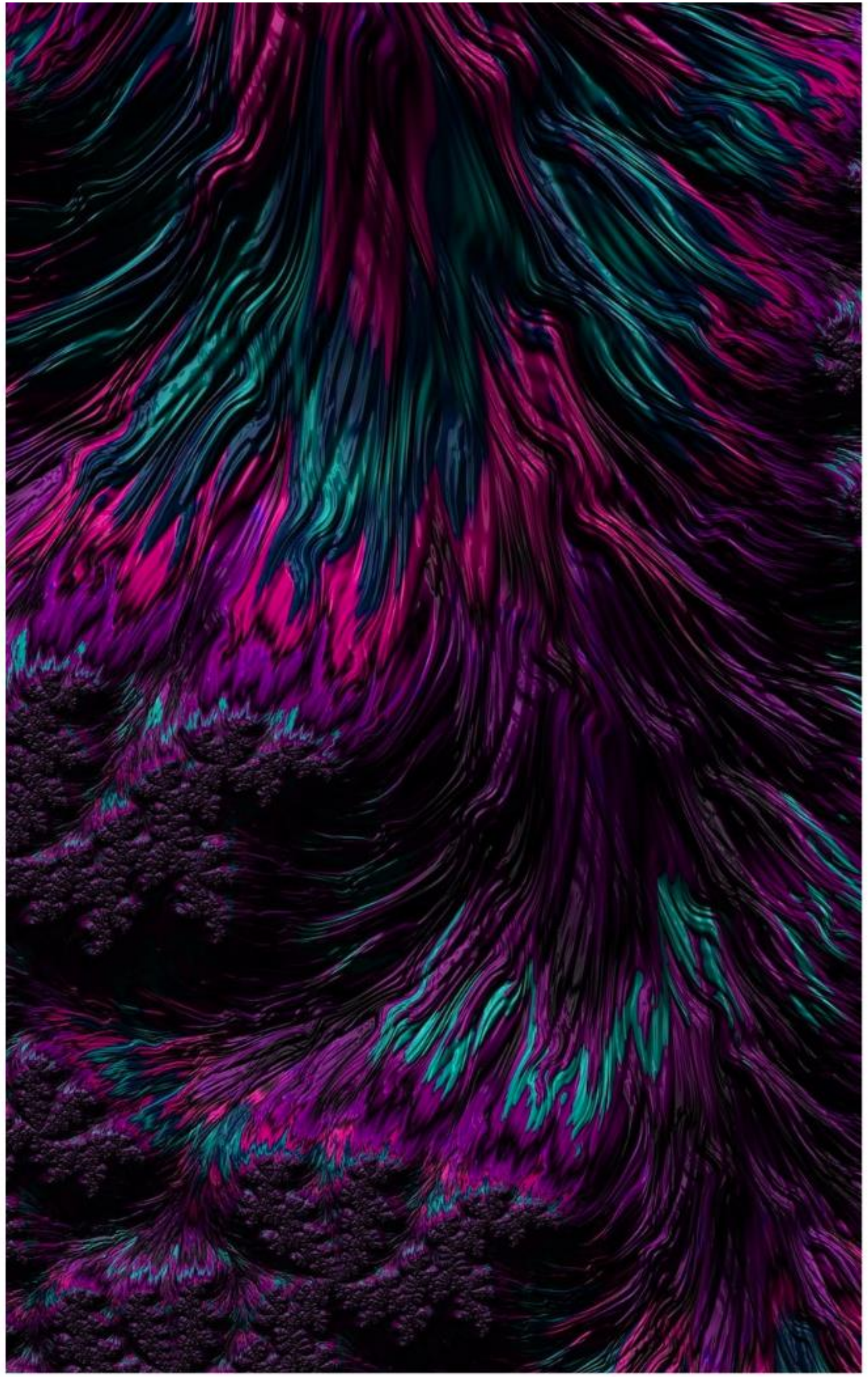
- Clean & migrate databases
- Junit extension

Working with Spring

- Wait.. how does Spring know where to connect?

- @DynamicPropertySource / ApplicationContextInitializer / @ServiceConnection

- Passing properties to individual tests



Developer Experience

Reuse existing containers
- Caveat: Experimental & not suited for CI

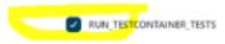
New Solutions to old Problems

Testcontainers module for Shopify’s Toxiproxy. This TCP proxy can be used to simulate network failure conditions.

CI/CD & Jenkins

Wenn die Testcontainer wiedermal klemmen...

tl;dr: Die Integration-Tests, welche Testcontainers benötigen können geskipped werden:



✓ RUN_TESTCONTAINER_TESTS

mehr...

Problem

- Die Testcontainers auf dem Jenkins brauchen ziemlich viel Ressourcen
- Wir bauen bLink mit mehreren Threads, d.h. es können mehrere Module gleichzeitig Testcontainers starten, und es werden auch noch mehrere CIs parallel ausgeführt. Daher kann es gut sein, dass mal 50 Containers am laufen sind.
- Gleichzeitig brauchen [redacted] und [redacted] ebenfalls ziemlich viel Ressourcen (tw. ebenfalls für Testcontainers, aber auch für anderes)

→ führt dazu, dass wir oft Probleme im Test Setup haben, z.B. Timeouts oder auch andere Fehler beim Starten

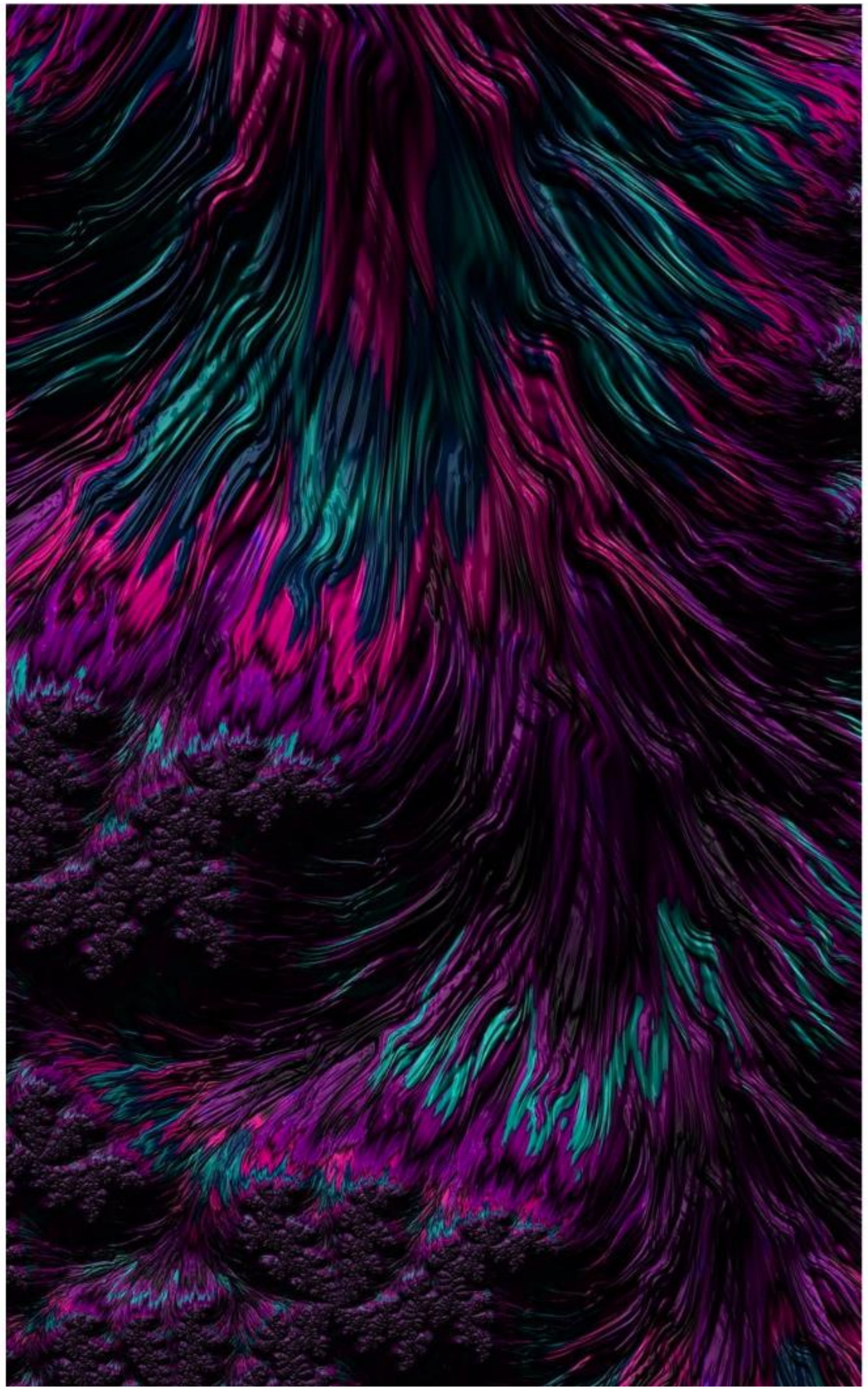
Massnahme

- Timeout wurde bereits auf 3min erhöht, und Anzahl erlaubte parallele CIs in bLink auf max. 3 beschränkt
- Wenn alle Stricke reissen kann man die Tests nun auch skippen (siehe Parameter im Screenshot)

→ einzige Regel: Know what you are doing ⚠️

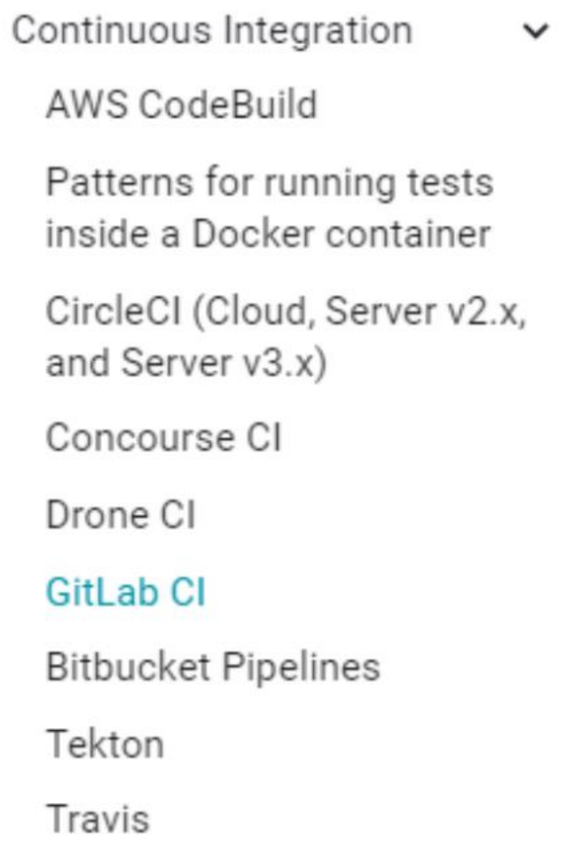
CI/CD & Jenkins

- Devtools support needed (Dockersocket on Jenkins)
- Additional resources needed (load on Jenkins affected test success)



Alternatives

- 'Docker wormhole' pattern
- Docker-in-Docker


Continuous Integration

AWS CodeBuild

Patterns for running tests inside a Docker container

CircleCI (Cloud, Server v2.x, and Server v3.x)

Concourse CI

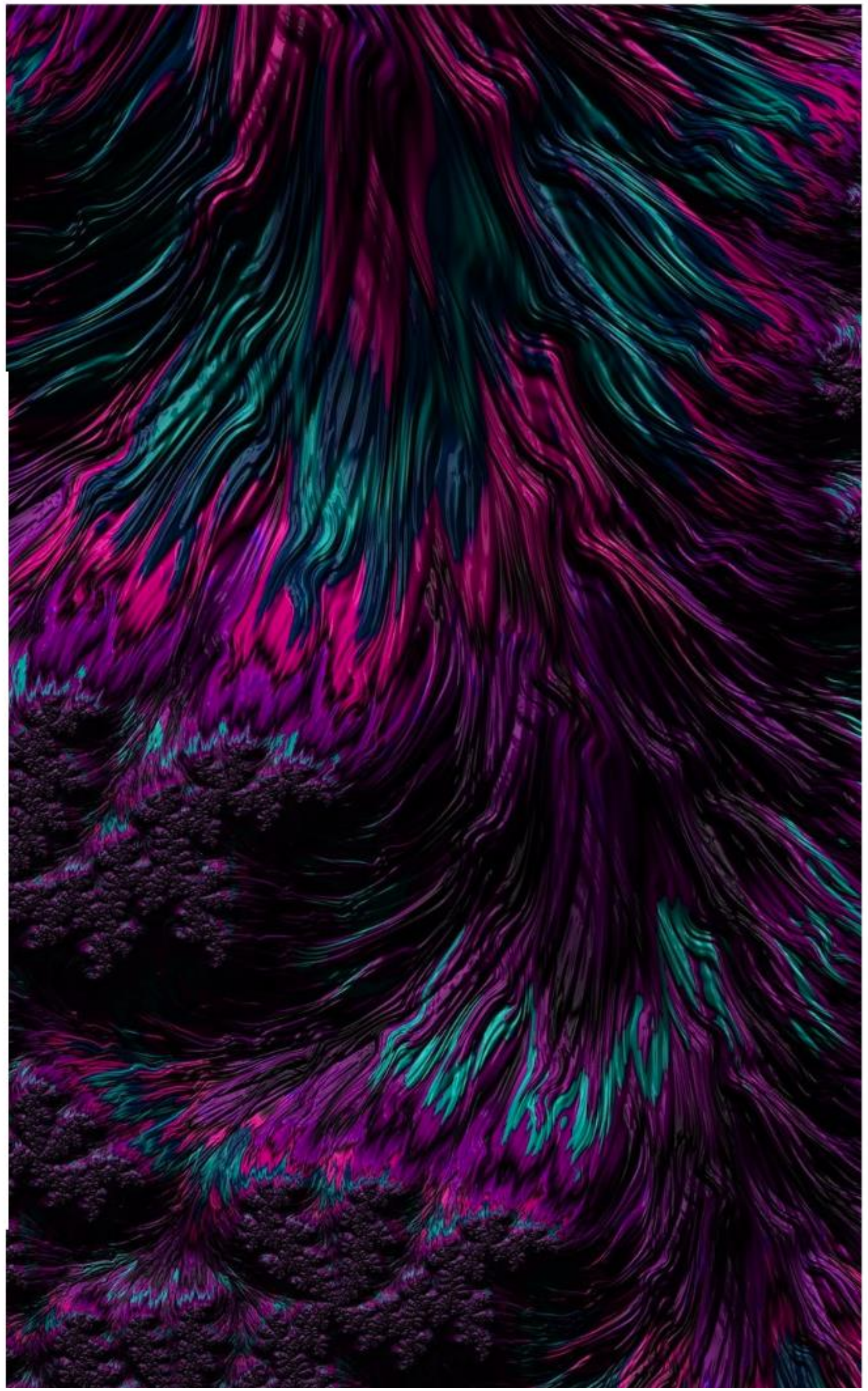
Drone CI

GitLab CI

Bitbucket Pipelines

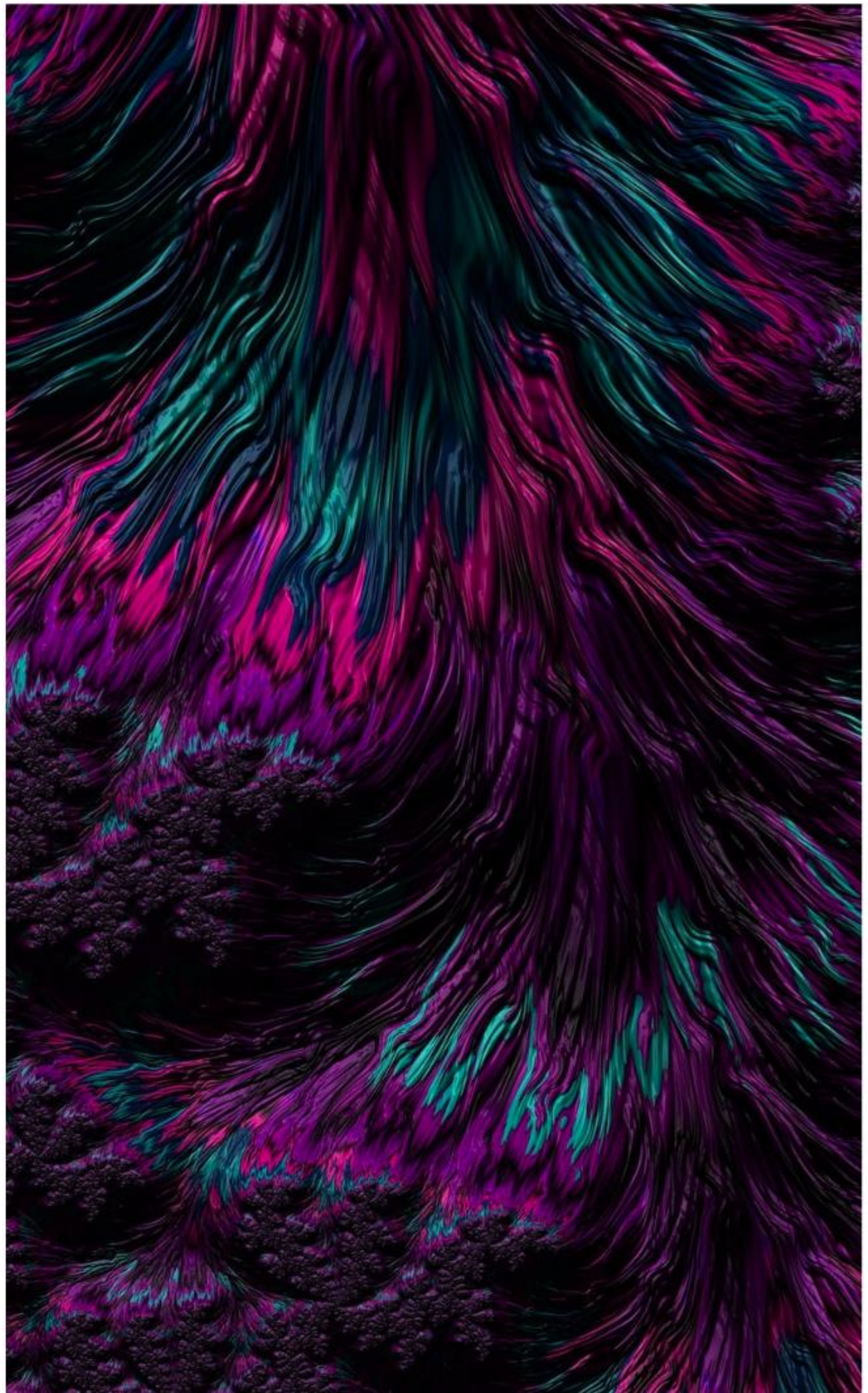
Tekton

Travis



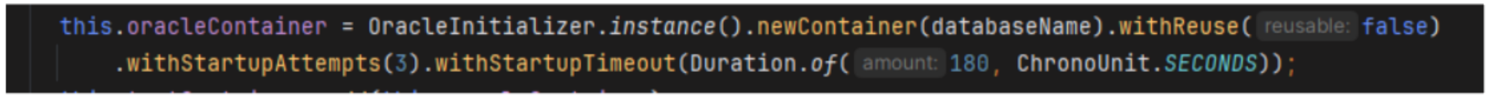
Parallel Execution

- currently averaging three containers per module
- adds about 5 minutes to build time to a total of 10 minutes.

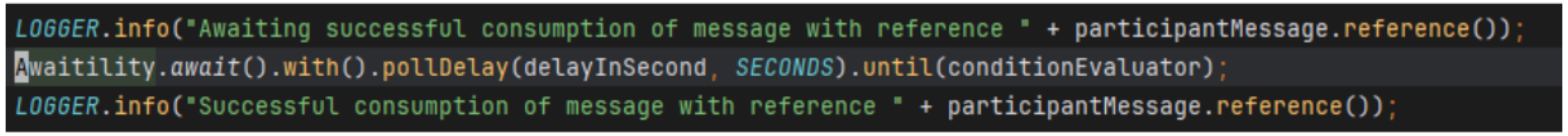


Dealing with Flakiness

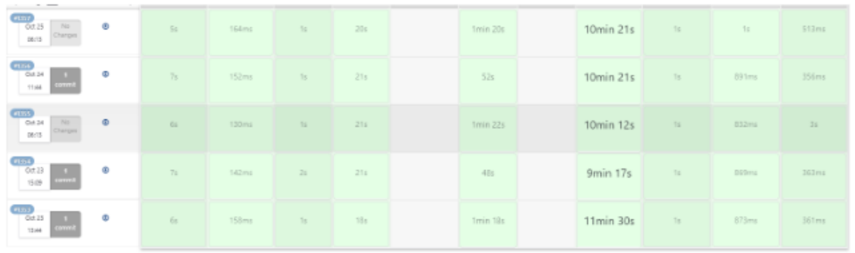
- Awaitility
- Timeouts
- Retries



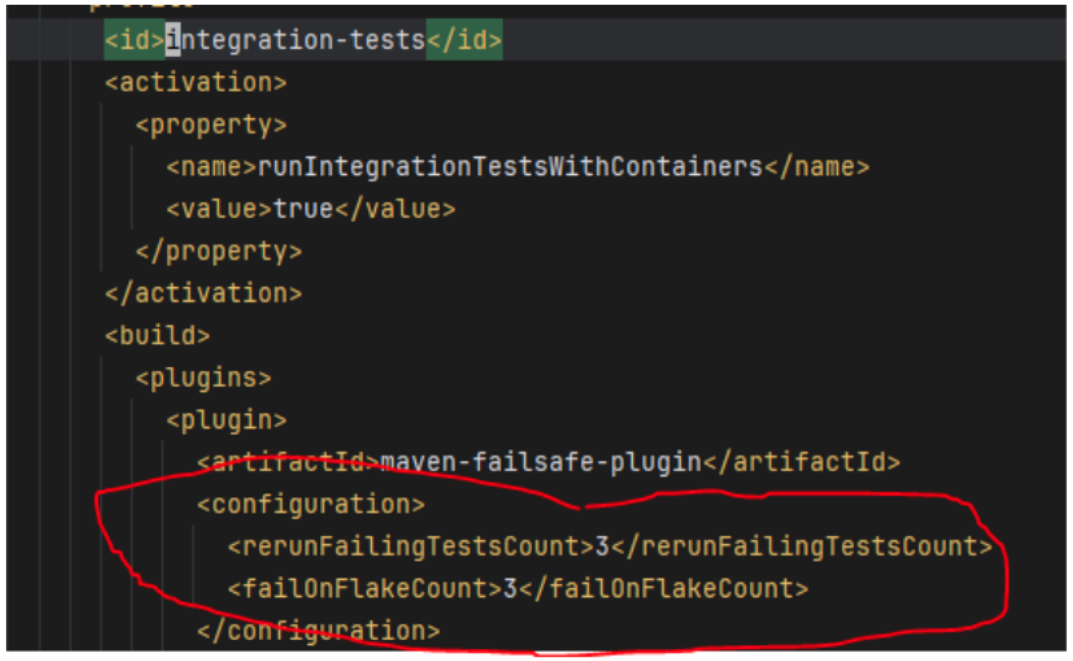
```
this.oracleContainer = OracleInitializer.instance().newContainer(databaseName).withReuse( reusable: false)
    .withStartupAttempts(3).withStartupTimeout(Duration.of( amount: 180, ChronoUnit.SECONDS));
```



```
LOGGER.info("Awaiting successful consumption of message with reference " + participantMessage.reference());
Awaitility.await().with().pollDelay(delayInSecond, SECONDS).until(conditionEvaluator);
LOGGER.info("Successful consumption of message with reference " + participantMessage.reference());
```



#1357 Oct 25 08:15 No Changes	5s	164ms	1s	20s		1min 20s		10min 21s	1s	1s	513ms
#1356 Oct 24 11:44 1 commit	7s	152ms	1s	21s		52s		10min 21s	1s	891ms	356ms
#1355 Oct 24 08:15 No Changes	6s	130ms	1s	21s		1min 22s		10min 12s	1s	832ms	3s
#1354 Oct 23 15:09 1 commit	7s	142ms	2s	21s		48s		9min 17s	1s	869ms	363ms
#1353 Oct 23 13:44 1 commit	6s	158ms	1s	18s		1min 18s		11min 30s	1s	873ms	361ms



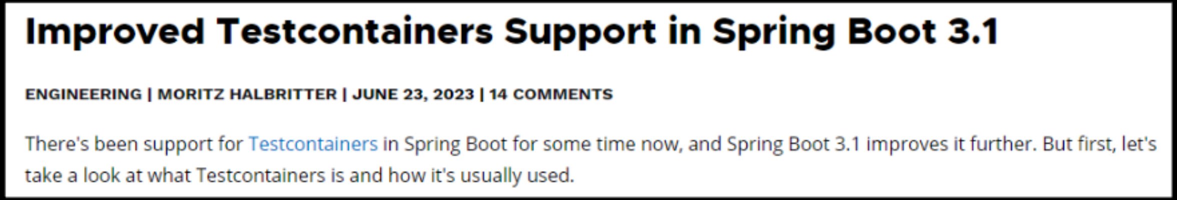
```
    <id>integration-tests</id>
    <activation>
      <property>
        <name>runIntegrationTestsWithContainers</name>
        <value>true</value>
      </property>
    </activation>
    <build>
      <plugins>
        <plugin>
          <artifactId>maven-failsafe-plugin</artifactId>
          <configuration>
            <rerunFailingTestsCount>3</rerunFailingTestsCount>
            <failOnFlakeCount>3</failOnFlakeCount>
          </configuration>
```

Complexity & Debugging

- If/when there are problems, they are usually not solved before your first coffee.
- "hmm, thats funny"
- Lots of moving parts
- Works locally, fails on Jenkins

Chapter 3

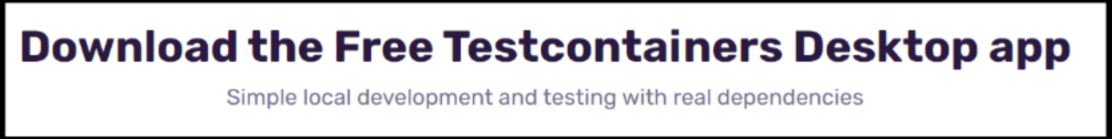
What's next?


Improved Testcontainers Support in Spring Boot 3.1

ENGINEERING | MORITZ HALBRITTER | JUNE 23, 2023 | 14 COMMENTS

There's been support for Testcontainers in Spring Boot for some time now, and Spring Boot 3.1 improves it further. But first, let's take a look at what Testcontainers is and how it's usually used.

- Increased support in Spring Boot 3
  - @ServiceConnection
- Desktop application to analyze test sessions
- High Activity, new modules


Download the Free Testcontainers Desktop app

Simple local development and testing with real dependencies

Conclusion

Pros	Cons
Adoption	Complexity with containers
Isolated & close to production	High resource usage (which can lead to flakiness)
Flexibility & new use cases	
Solid Spring integration	