Moderne Integration Tests mit Testcontainers

@anderScoreGmbH anderScore.company Java_Meetup_anderscore

Zur unserer Person

- **Daniel Krämer**
- Software-Entwickler, Architekt
- Integration und Migration
- Web Engineering
- Testautomatisierung
- dkraemer-anderscore

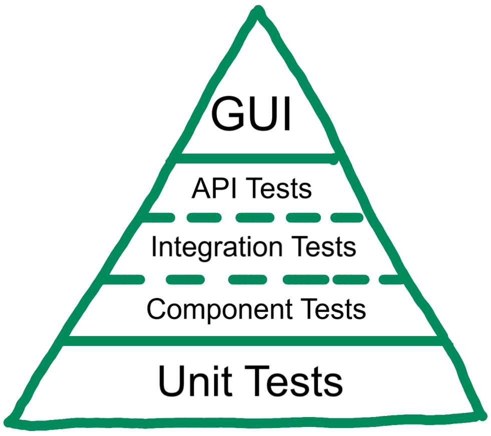
- **Maik Wolf**
- Software-Entwickler
- Fullstack & Devops
- @da_mwolf
- maikwolf

Unternehmen

- Standort: Köln (mit Rheinblick…)
- Individuelle Softwareentwicklung
- Consulting und Festpreis
- Gesamter Application Life Cycle
- Konferenzen und Artikel
- Öffentliche Trainings

- Technologien
  - JEE, Spring
  - Wicket, Angular
  - Docker, Kubernetes, Apache Kafka
  - …
- Goldschmiede@anderScore

Test-Pyramide



Warum Integrationstests?

[unit test vs integration test] | *unit-test-vs-integration-test.gif*

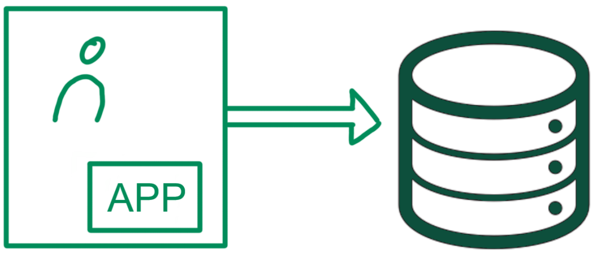
Herausforderung Integrationstest

- **Fremdsysteme**
  - Erreichbarkeit

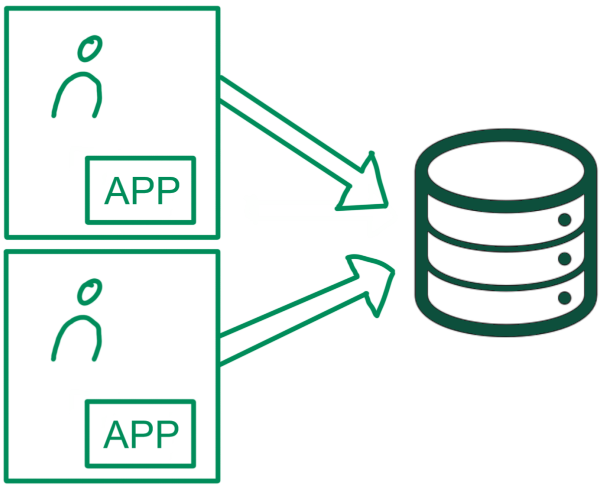
  - Datenhoheit
  - Zustand
  - Performance
  - Nebenläufigkeit

- **Unterschiedliche Umgebungen**
  - Software
  - Versionen
  - Datenbankschema
  - Konfiguration

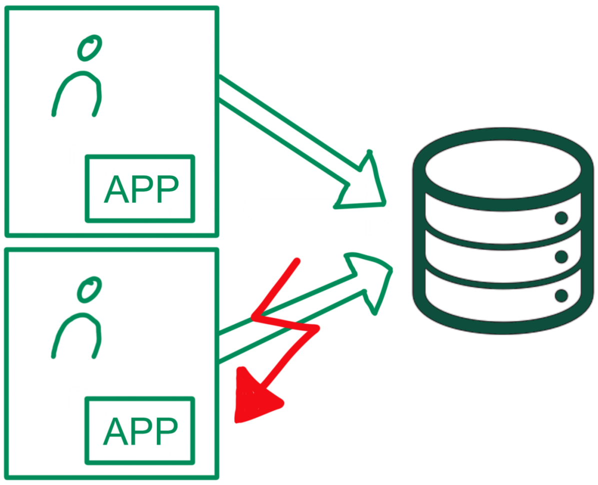
Datenbank-Erreichbarkeit



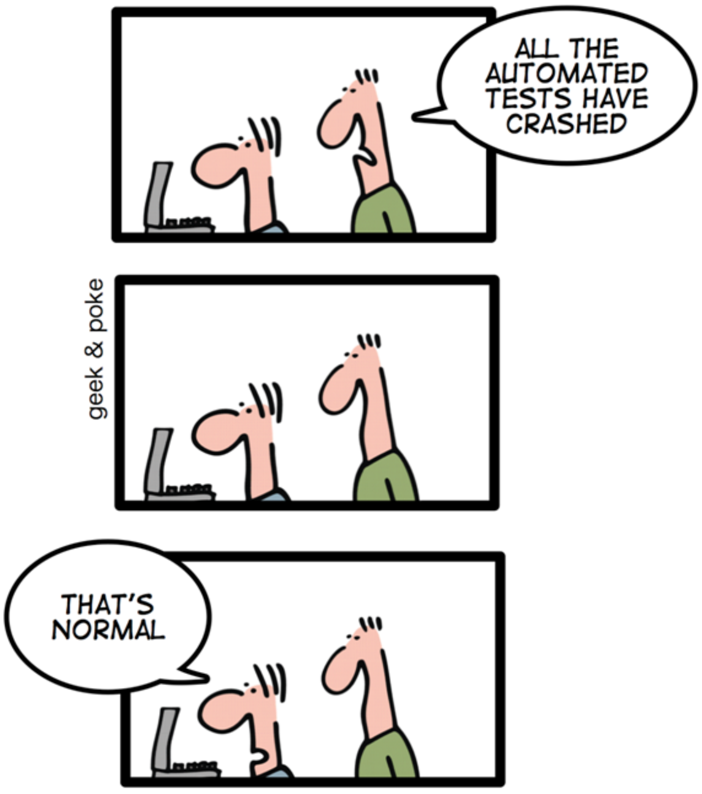
Datenbank-Erreichbarkeit



Datenbank-Erreichbarkeit



Datenbank-Erreichbarkeit



Was würde uns helfen?

- Einheitlichkeit
- Reproduzierbarkeit
- Kontrolle
- Lokalität
- Nähe zur Produktion

Testcontainers

- **Java Library**
- **MIT Lizenz**
- **Einsatzszenarien**
  - Integration Tests (DB, Message Broker, etc.)
  - Anwendungstests (UI, Use Cases)
- **Technologien**
  - Docker
  - JUnit 4/5

Beispiel: Code

```
@SpringJUnitConfig(TestConfig.class)
@ActiveProfiles("test")
@ExtendWith(DbContainerExtension.class)
@TestExecutionListeners({DependencyInjectionTestExecutionListener.class,
FlywayTestExecutionListener.class})
public class SchedulerServiceTest {

    @Inject
    private SchedulerService schedulerService;

    @Test
    @FlywayTest(locationsForMigrate = {"db/scheduler_data"})
    public void findAll() {
        List<Scheduler> result = schedulerService.findAll();

        assertEquals(2, result.size());
    }

    @Test
    public void store() {
        Scheduler scheduler = new Scheduler();
        scheduler.setName("ElBarto");

        schedulerService.store(scheduler);

        Optional<Scheduler> saveScheduler = schedulerService.findByName("ElBarto"
);

        assertTrue(saveScheduler.isPresent());
        assertEquals(saveScheduler.get().getName(), scheduler.getName());
    }
}
```

Beispiel: Docker starten

```
           Docker version should be at least 1.6.0
18:48:32.713 [main] DEBUG com.github.dockerjava.core.command.AbstrDockerCmd - Cmd:
6458137d3a598b22c93a71cf8ae31fb8ae8c9a36ce4e7b73d91d3a1a6c68c848,<null>,true,<null
>,<null>,<null>,<null>,{df,-
P},<null>,<null>,com.github.dockerjava.core.exec.ExecCreateCmdExec@2b4786dd
18:48:32.923 [tc-okhttp-stream-1322484262] DEBUG
com.github.dockerjava.core.command.ExecStartResultCallback - STDOUT: Filesystem
1024-blocks    Used Available Capacity Mounted on
overlay             490691512 113658060 352037972  24% /
tmpfs                   65536         0     65536   0% /dev
tmpfs                 8081808         0   8081808   0% /sys/fs/cgroup
/dev/nvme0n1p2      490691512 113658060 352037972  24% /etc/resolv.conf
/dev/nvme0n1p2      490691512 113658060 352037972  24% /etc/hostname
/dev/nvme0n1p2      490691512 113658060 352037972  24% /etc/hosts
shm                     65536         0     65536   0% /dev/shm
tmpfs                 1616364      3928   1612436   0% /run/docker.sock
tmpfs                 8081808         0   8081808   0% /proc/asound
tmpfs                 8081808         0   8081808   0% /proc/acpi
tmpfs                   65536         0     65536   0% /proc/kcore
tmpfs                   65536         0     65536   0% /proc/keys
tmpfs                   65536         0     65536   0% /proc/timer_list
tmpfs                   65536         0     65536   0% /proc/sched_debug
tmpfs                 8081808         0   8081808   0% /proc/scsi
tmpfs                 8081808         0   8081808   0% /sys/firmware
           Docker environment should have more than 2GB free disk space
18:48:32.955 [main] DEBUG com.github.dockerjava.core.command.AbstrDockerCmd - Cmd:
ListImagesCmdImpl[imageNameFilter=<null>,showAll=false,filters=com.github.dockerja
va.core.util.FiltersBuilder@0,execution=com.github.dockerjava.core.exec.ListImages
CmdExec@22bd2039]
18:48:32.967 [main] DEBUG    [postgres:9.6.12] - Starting container:
postgres:9.6.12
18:48:32.967 [main] DEBUG    [postgres:9.6.12] - Trying to start container:
postgres:9.6.12
18:48:32.967 [main] DEBUG    [postgres:9.6.12] - Trying to start container:
postgres:9.6.12 (attempt 1/1)
18:48:32.967 [main] DEBUG    [postgres:9.6.12] - Starting container:
postgres:9.6.12
18:48:32.967 [main] INFO    [postgres:9.6.12] - Creating container for image:
postgres:9.6.12
```

Beispiel: Container starten

```
   [postgres:9.6.12] - Starting container with ID:
b447ee79d8ab49fb0a92edc05fc571abbbe02291746b5fbef2a3498f33c0c337
com.github.dockerjava.core.command.AbstrDockerCmd - Cmd:
b447ee79d8ab49fb0a92edc05fc571abbbe02291746b5fbef2a3498f33c0c337,com.github.docker
java.core.exec.StartContainerCmdExec@42fcc7e6
   [postgres:9.6.12] - Container postgres:9.6.12 is starting:
b447ee79d8ab49fb0a92edc05fc571abbbe02291746b5fbef2a3498f33c0c337
com.github.dockerjava.core.command.AbstrDockerCmd - Cmd:
b447ee79d8ab49fb0a92edc05fc571abbbe02291746b5fbef2a3498f33c0c337,false,com.github.
dockerjava.core.exec.InspectContainerCmdExec@5da7cee2
com.github.dockerjava.core.exec.InspectContainerCmdExec - GET:
OkHttpWebTarget(okHttpClient=org.testcontainers.shaded.okhttp3.OkHttpClient@19569e
bd, baseUrl=http://docker.socket/,
path=[/containers/b447ee79d8ab49fb0a92edc05fc571abbbe02291746b5fbef2a3498f33c0c337
/json], queryParams={})
com.github.dockerjava.core.command.AbstrDockerCmd - Cmd:
b447ee79d8ab49fb0a92edc05fc571abbbe02291746b5fbef2a3498f33c0c337,false,com.github.
dockerjava.core.exec.InspectContainerCmdExec@120ddb7
com.github.dockerjava.core.exec.InspectContainerCmdExec - GET:
OkHttpWebTarget(okHttpClient=org.testcontainers.shaded.okhttp3.OkHttpClient@19569e
bd, baseUrl=http://docker.socket/,
path=[/containers/b447ee79d8ab49fb0a92edc05fc571abbbe02291746b5fbef2a3498f33c0c337
/json], queryParams={})
```

Beispiel: Hikari Konfiguration

```
com.zaxxer.hikari.HikariConfig - HikariPool-1 - configuration:
com.zaxxer.hikari.HikariConfig -
dataSourceProperties............{password=<masked>}
com.zaxxer.hikari.HikariConfig -
driverClassName................."org.postgresql.Driver"
com.zaxxer.hikari.HikariConfig - healthCheckProperties...........{}
com.zaxxer.hikari.HikariConfig - healthCheckRegistry.............none
com.zaxxer.hikari.HikariConfig - idleTimeout.....................600000
com.zaxxer.hikari.HikariConfig - initializationFailTimeout.......1
com.zaxxer.hikari.HikariConfig - isolateInternalQueries..........false
com.zaxxer.hikari.HikariConfig -
jdbcUrl.........................jdbc:postgresql://localhost:32781/scheduler?logger
Level=OFF
com.zaxxer.hikari.HikariConfig - leakDetectionThreshold..........0
com.zaxxer.hikari.HikariConfig - maxLifetime.....................1800000
com.zaxxer.hikari.HikariConfig - maximumPoolSize.................10
com.zaxxer.hikari.HikariConfig - metricRegistry..................none
com.zaxxer.hikari.HikariConfig - metricsTrackerFactory...........none
com.zaxxer.hikari.HikariConfig - minimumIdle.....................10
com.zaxxer.hikari.HikariConfig - password........................<masked>
com.zaxxer.hikari.HikariConfig - poolName........................"HikariPool-1"
com.zaxxer.hikari.HikariConfig - username........................"postgres"
com.zaxxer.hikari.HikariConfig - validationTimeout...............5000
com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
com.zaxxer.hikari.pool.HikariPool - HikariPool-1 - Added connection
org.postgresql.jdbc.PgConnection@2b917fb0
com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
```

Beispiel: Flyway

```
org.flywaydb.core.internal.scanner.classpath.ClassPathScanner - Found resource:
db/migration/V001__init_scheduler_table.sql
org.flywaydb.core.internal.scanner.classpath.ClassPathScanner - Scanning for
classes at classpath:db/migration
org.flywaydb.core.internal.callback.SqlScriptCallbackFactory - Scanning for SQL
callbacks ...
org.flywaydb.core.internal.util.FeatureDetector - Spring Jdbc available: true
org.flywaydb.core.internal.command.DbValidate - Validating migrations ...
org.flywaydb.core.internal.sqlscript.SqlScript - Parsing
V001__init_scheduler_table.sql ...
org.flywaydb.core.internal.sqlscript.SqlScript - Found statement at line 1: CREATE
TABLE scheduler
(
  id        serial              NOT NULL,
  name      varchar(255)        NOT NULL,
  PRIMARY KEY (id)
)
org.flywaydb.core.internal.sqlscript.SqlScript - Found statement at line 8:
comment on table scheduler
  is 'Scheduler that runs to create tickets in wekan'
[main] DEBUG org.flywaydb.core.internal.sqlscript.SqlScript - Found statement at
line 11: create index scheduler_name_index
  on scheduler (name)
org.flywaydb.core.internal.scanner.Scanner - Filtering out resource:
db/migration/V001__init_scheduler_table.sql (filename:
V001__init_scheduler_table.sql)
org.flywaydb.core.internal.command.DbValidate - Successfully validated 1 migration
(execution time 00:00.019s)
org.flywaydb.core.internal.command.DbSchemas - Schema "public" already exists.
Skipping schema creation.
org.flywaydb.core.internal.schemahistory.JdbcTableSchemaHistory - Creating Schema
History table: "public"."flyway_schema_history"
org.flywaydb.core.internal.sqlscript.SqlScript - Parsing createMetaDataTable.sql
...
org.flywaydb.core.internal.sqlscript.SqlScript - Found statement at line 17:
CREATE TABLE "public"."flyway_schema_history" (
    "installed_rank" INT NOT NULL,
    "version" VARCHAR(50),
    "description" VARCHAR(200) NOT NULL,
    "type" VARCHAR(20) NOT NULL,
    "script" VARCHAR(1000) NOT NULL,
    "checksum" INTEGER,
    "installed_by" VARCHAR(100) NOT NULL,
    "installed_on" TIMESTAMP NOT NULL DEFAULT now(),
    "execution_time" INTEGER NOT NULL,
    "success" BOOLEAN NOT NULL
)
```

Beispiel: JUnit-Test

```
org.hibernate.hql.internal.ast.QueryTranslatorImpl - HQL: select generatedAlias0
from com.anderscore.testcontainers.data.Scheduler as generatedAlias0 where
generatedAlias0.name=:name
org.hibernate.hql.internal.ast.QueryTranslatorImpl - SQL: select scheduler0_.id as
id1_0_, scheduler0_.name as name2_0_ from Scheduler scheduler0_ where
scheduler0_.name=?
org.hibernate.hql.internal.ast.ErrorTracker - throwQueryException() : no errors
com.zaxxer.hikari.pool.HikariPool - HikariPool-1 - Added connection
org.postgresql.jdbc.PgConnection@4c341efc
com.zaxxer.hikari.pool.HikariPool - HikariPool-1 - After adding stats (total=9,
active=0, idle=9, waiting=0)
com.zaxxer.hikari.pool.HikariPool - HikariPool-1 - Added connection
org.postgresql.jdbc.PgConnection@645706f8
com.zaxxer.hikari.pool.HikariPool - HikariPool-1 - After adding stats (total=10,
active=0, idle=10, waiting=0)
org.hibernate.SQL - select scheduler0_.id as id1_0_, scheduler0_.name as name2_0_
from Scheduler scheduler0_ where scheduler0_.name=?
Hibernate: select scheduler0_.id as id1_0_, scheduler0_.name as name2_0_ from
Scheduler scheduler0_ where scheduler0_.name=?
```

Was kann Testcontainers sonst noch?

- **GenericContainer**
    - Fertiges Docker Image
    - On-the-fly Docker Image (Dockerfile DSL)
    - Ausführen oder Überschreiben von Commands

- **Fluent API**
    - Ports, Environment Variables
    - Shell Commands
    - Log Consumer

```
GenericContainer alpine =
    new GenericContainer("alpine:3.2")
            .withExposedPorts(80)
            .withEnv("MAGIC_NUMBER", "42")
            .withCommand("/bin/sh", "-c",
            "while true; do echo \"$MAGIC_NUMBER\" | nc -l -p 80; done");
```

Was kann Testcontainers sonst noch?

- **Netzwerk**
    - Externe Ports zufällig
    - Zugang zu Host Ports möglich
    - Container-Netzwerke

```
Network network = Network.newNetwork();

GenericContainer foo = new GenericContainer()
    .withNetwork(network)
    .withNetworkAliases("foo");

GenericContainer bar = new GenericContainer()
    .withNetwork(network);
```

Was kann Testcontainers sonst noch?

- **Logs**
  - Auslesen (stdout, stderr)
  - Streamen

```
Slf4jLogConsumer logConsumer = new Slf4jLogConsumer(LOGGER);
container.followOutput(logConsumer);
```

Module

- DBMS (u.a. PostgreSQL, MySQL, DB2, Cassandra, Neo4j)
- ElasticSearch
- Kafka
- Nginx
- Webdriver (Selenium, inkl. VNC)

Datenbank - Extension

```
@SpringJUnitConfig(TestConfig.class)
@ActiveProfiles("test")
@ExtendWith(DbContainerExtension.class)
@TestExecutionListeners({DependencyInjectionTestExecutionListener.class,
FlywayTestExecutionListener.class})
public class SchedulerServiceTest {
```

```
public class DbContainerExtension implements AfterEachCallback {

    @Override
    public void afterEach(ExtensionContext extensionContext) throws Exception {
        DatabaseContainerHolder containerHolder = SpringExtension
.getApplicationContext(extensionContext)
                .getBean(DatabaseContainerHolder.class);

        Flyway flyWay = SpringExtension.getApplicationContext(extensionContext)
.getBean(Flyway.class);

        containerHolder.refresh();
        flyWay.migrate();
    }
}
```

Konfiguration

```
@Configuration
@Profile("test")
@Import(TestPersistenceConfig.class)
@ComponentScan(basePackageClasses = {SchedulerMapper.class, SchedulerService.
class})
@EnableJpaRepositories(basePackageClasses = SchedulerRepository.class)
public class TestConfig {
}
```

Konfiguration

```java
@ComponentScan(basePackageClasses = DatabaseContainerHolder.class)
public class TestPersistenceConfig {

    @Inject
    private Environment env;

    @Bean
    public Flyway flyway(DatabaseContainerHolder containerHolder) {
        JdbcDatabaseContainer<?> dbContainer = containerHolder.get();

        Flyway flyway = Flyway.configure().dataSource(dbContainer.getJdbcUrl(),
                dbContainer.getUsername(),
                dbContainer.getPassword())
                .load();

        flyway.migrate();

        return flyway;
    }

    @Bean
    public DataSource dataSource(DatabaseContainerHolder containerHolder) {
        JdbcDatabaseContainer<?> dbContainer = containerHolder.get();

        HikariConfig hikariConfig = new HikariConfig();
        hikariConfig.setJdbcUrl(dbContainer.getJdbcUrl());
        hikariConfig.setUsername(dbContainer.getUsername());
        hikariConfig.setPassword(dbContainer.getPassword());
        hikariConfig.setDriverClassName(env.getProperty("jdbc.driverClassName"));

        return new HikariDataSource(hikariConfig);
    }
}
```

Konfiguration

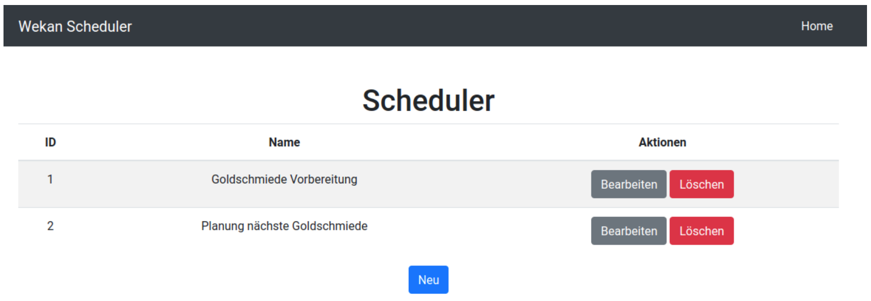
DatabaseContainerHolder.java

```
    private JdbcDatabaseContainer<?> newContainer(){
        JdbcDatabaseContainer<?> container = new PostgreSQLContainer<>()
                .withDatabaseName(DB_SCHEMA)
                .withUsername(DB_USER)
                .withPassword(DB_PASSWD);

        if (hostDbPort > 0){
            container.setPortBindings(asList(hostDbPort + ":" + CONTAINER_DB_PORT
));
        }

        container.start();

        return container;
    }
```

DatabaseContainerHolder.java

```
    public void refresh(){
        if (instance != null && instance.isRunning()){
            instance.stop();
        }

        instance = newContainer();
        hostDbPort = instance.getMappedPort(CONTAINER_DB_PORT);
    }
```

Front-End - Wicket



Scheduler

ID	Name	Aktionen
1	Goldschmiede Vorbereitung	Bearbeiten Löschen
2	Planung nächste Goldschmiede	Bearbeiten Löschen

Front-End - Test

```java
@SpringBootTest(webEnvironment = RANDOM_PORT)
@ActiveProfiles("test")
@ExtendWith(DbContainerExtension.class)
@ExtendWith(WebDriverContainerExtension.class)
@ExtendWith(ServletContainerContextParameterResolver.class)
@ExtendWith(WebDriverParameterResolver.class)
public class WekanSchedulerTest {

  @Test
  public void createScheduler(ServletContainerContext context, RemoteWebDriver
webDriver) {
      webDriver.get(context.getHttpUrl());

      // SchedulerOverviewPage
      assertEquals("Scheduler", webDriver.findElement(By.tagName("h1")).getText()
);
      webDriver.findElement(By.id("new")).click();

      // SchedulerCreationPage
      assertEquals("Scheduler anlegen", webDriver.findElement(By.tagName("h1"))
.getText());
      WebElement nameInput = webDriver.findElement(By.id("name"));
      nameInput.sendKeys("TestScheduler");
      nameInput.submit();

      // SchedulerOverviewPage
      assertEquals("Scheduler", webDriver.findElement(By.tagName("h1")).getText()
);
      assertEquals("1", webDriver.findElement(By.cssSelector("td:nth-child(1)
span")).getText());
      assertEquals("TestScheduler", webDriver.findElement(By.cssSelector("td:nth-
child(2) span")).getText());
  }
}
```

Konfiguration

```
public class WebDriverContainerExtension implements BeforeEachCallback,
AfterEachCallback {

    @Override
    public void beforeEach(ExtensionContext extensionContext) throws Exception {
        String serverPort = SpringExtension.getApplicationContext(
extensionContext).getEnvironment()
                .getProperty("local.server.port");

        Testcontainers.exposeHostPorts(parseInt(serverPort));

        BrowserWebDriverContainer<?> container = new BrowserWebDriverContainer<>()
                .withCapabilities(new ChromeOptions())
                .withRecordingMode(RECORD_FAILING, new File("./target/"));

        container.start();

        extensionContext.getStore(GLOBAL).put(BrowserWebDriverContainer.class
.getSimpleName(), container);
    }

    @Override
    public void afterEach(ExtensionContext extensionContext) throws Exception {
        BrowserWebDriverContainer<?> container = extensionContext.getStore(GLOBAL)
                .get(BrowserWebDriverContainer.class.getSimpleName(),
BrowserWebDriverContainer.class);

        container.stop();
    }
}
```

Pitfalls

- Zufällige Ports
- Fixe DataSource im SpringContext
- Kommunikation mit anderen Containern und Host

The good, the bad & the ugly

The good

- Einfache Verwendung
- Übersichtliche API
- Verständliche Dokumentation
- Gute Integration mit Junit 4 und 5
- Vielfältig einsetzbar

The bad

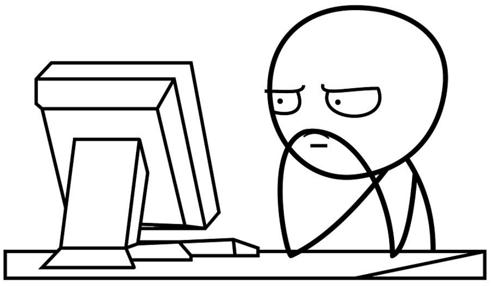
- Performance
- Eingeschränkte Integration mit Spring
- Keine parallele Testausführung

The ugly

- Internes Datenmodell

```
public class PostgreSQLContainer<SELF extends PostgreSQLContainer<SELF>> extends
JdbcDatabaseContainer<SELF> {
[...]
}
```

```
JdbcDatabaseContainer<?> container = new PostgreSQLContainer<>();
```



Links

- Testcontainers Dokumentation: https://www.testcontainers.org
- Docker Dokumentation: https://docs.docker.com
- Vortrag Sergei Egorov (Mitentwickler Testcontainers): https://www.youtube.com/watch?v=rv-NxOTMvDQ (Russisch)
- Folien + Demo: https://github.com/anderscore-gmbh/Testcontainers-EducationQuickie

Ende

Vielen Dank!

@anderScoreGmbH anderScore.company Java_Meetup_anderscore