

#WISSENTEILEN  
 @\_openKnowledge

# TESTING WITH CONTAINERS

Stephan Müller, Christian Wansart | open knowledge GmbH

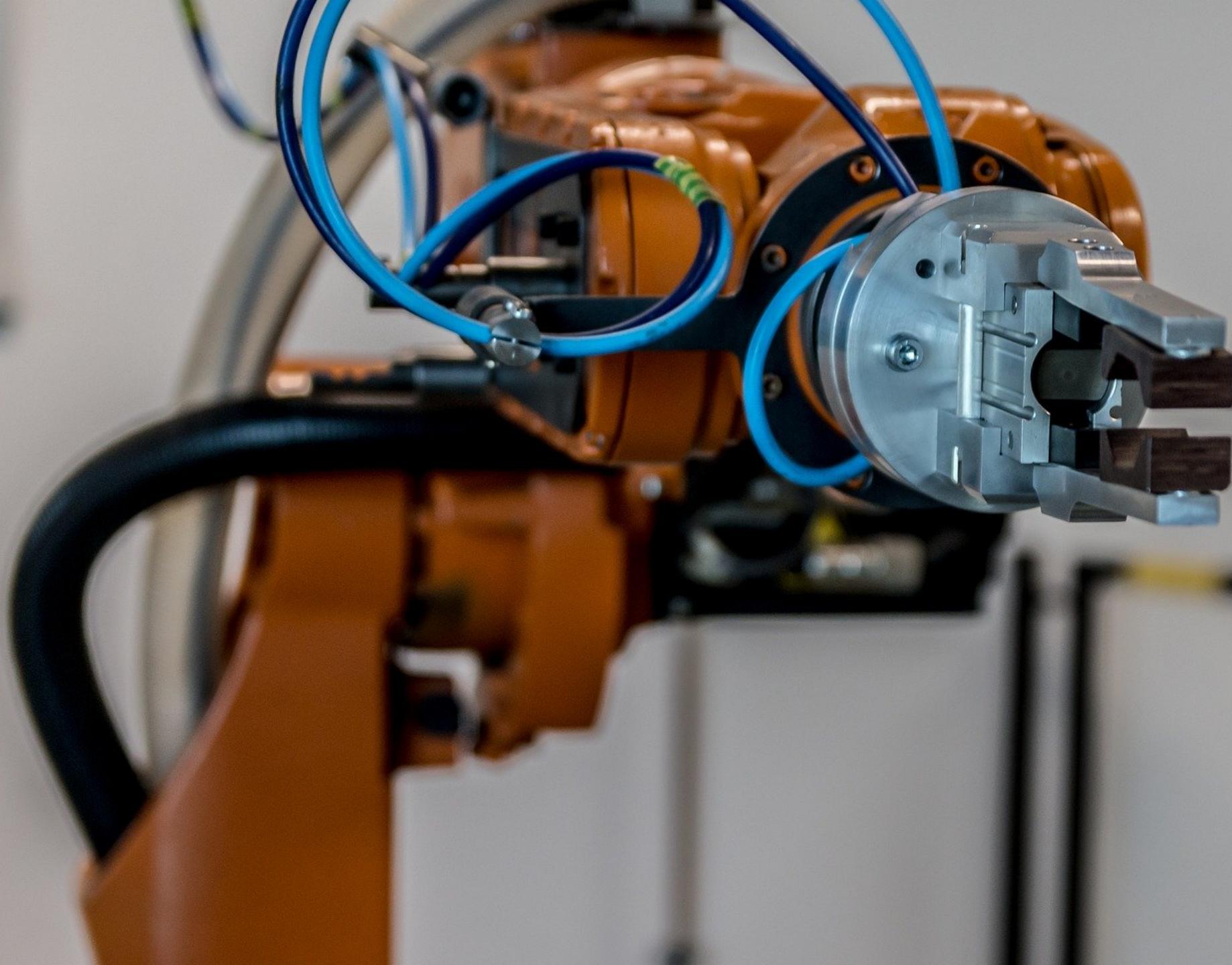
# Demo application & exercises



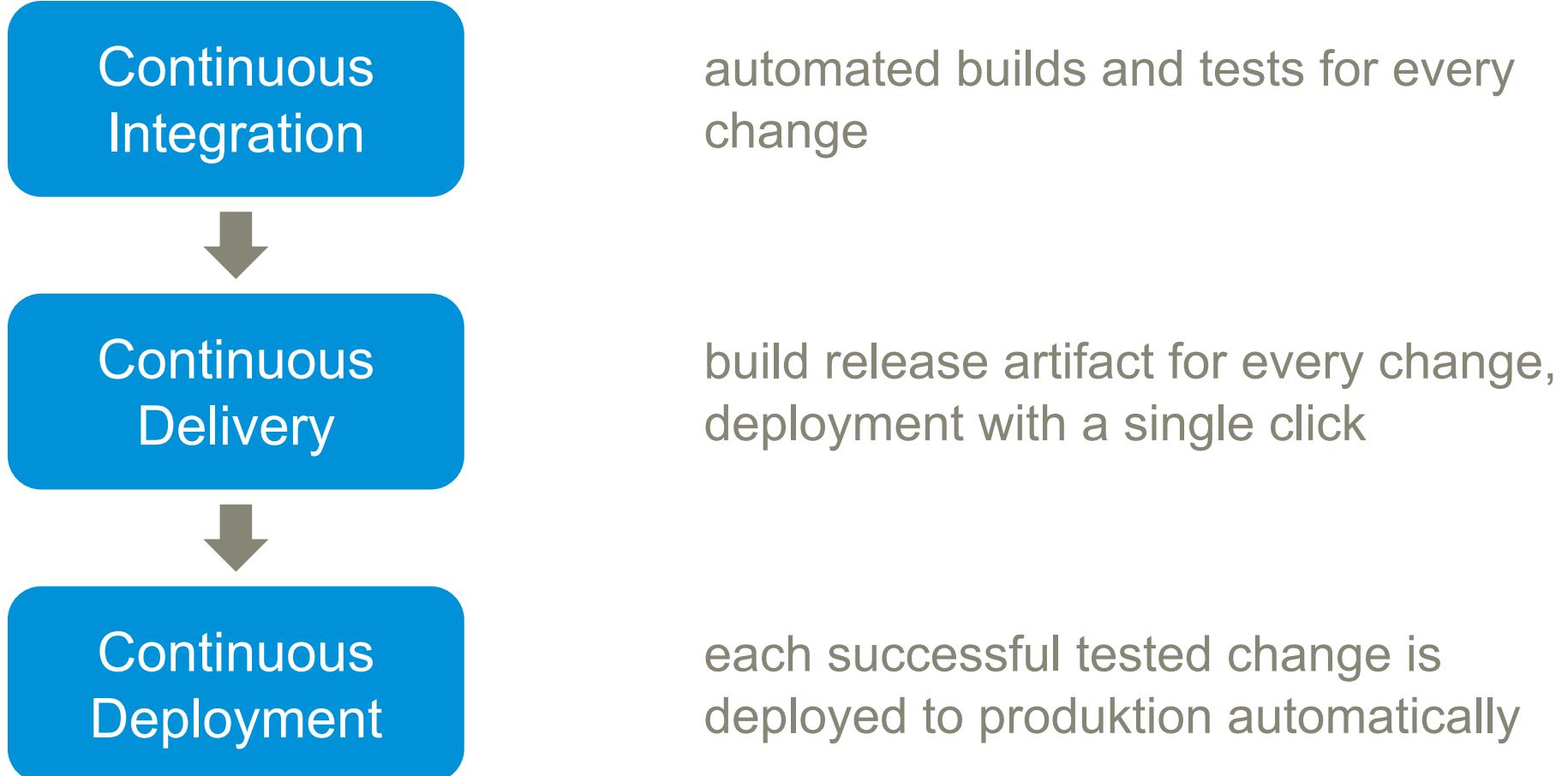
<https://github.com/stephan-mueller/testing-with-containers>

Amazon deploys  
to production every 11,6s\*

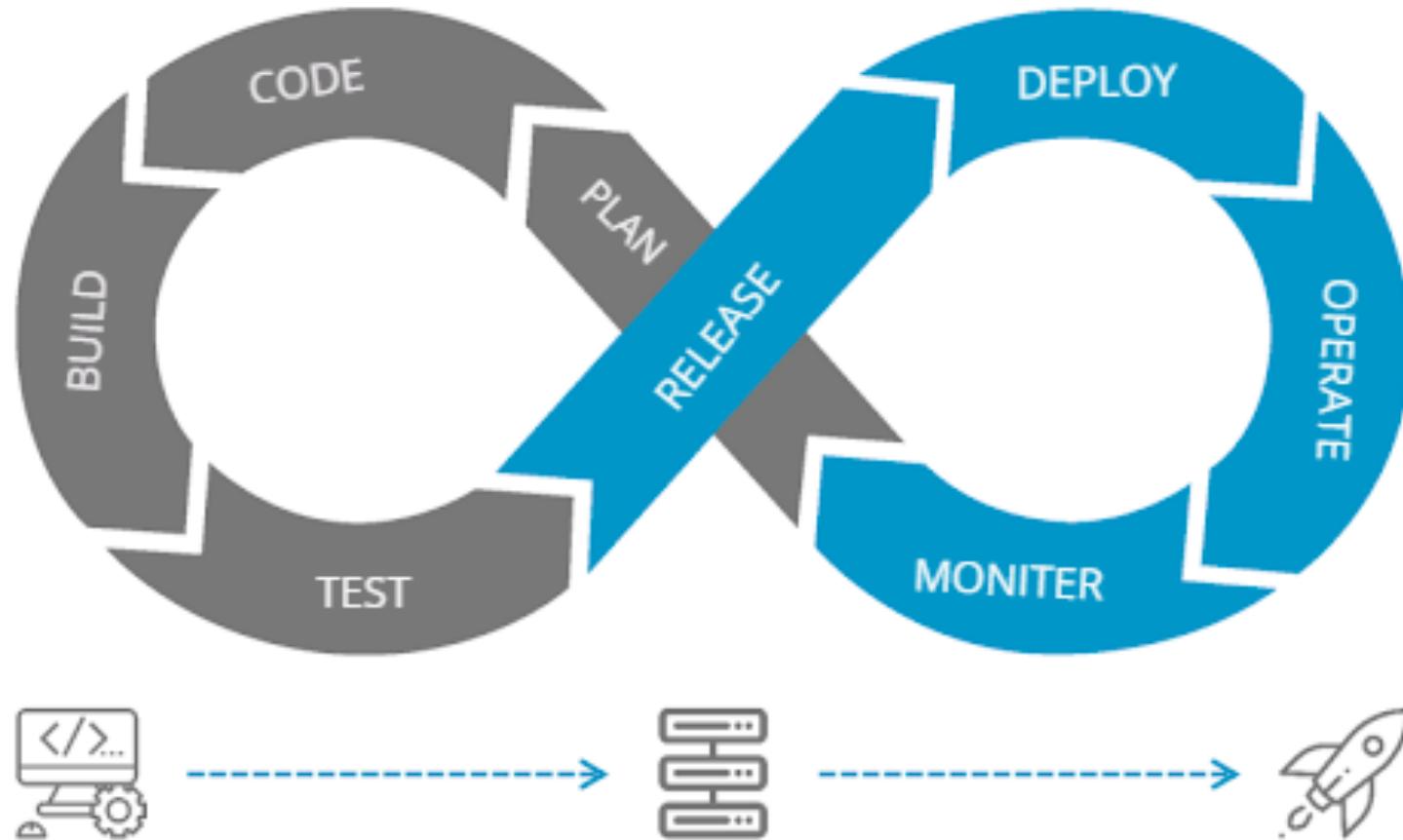
\* In 2011. 2018 Amazon made 23.000 deployments a day



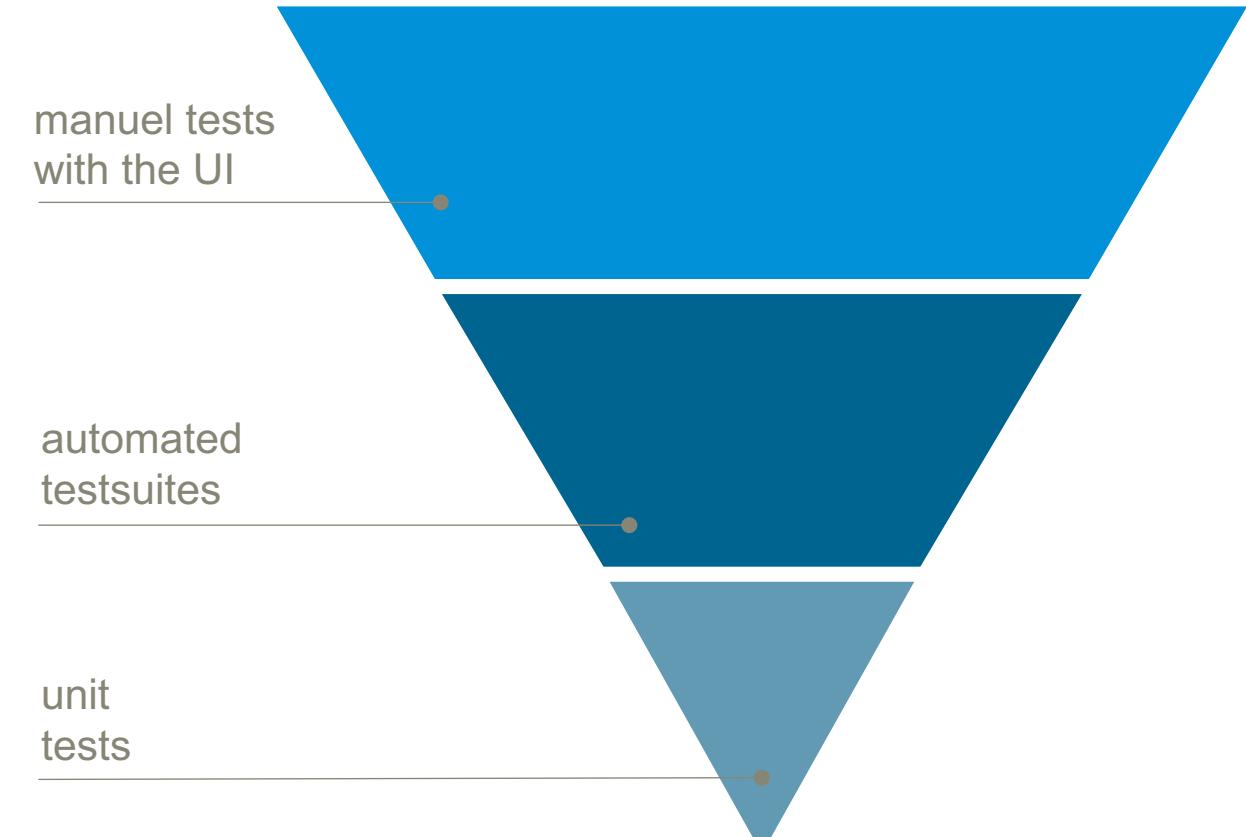
# Modern Software Development



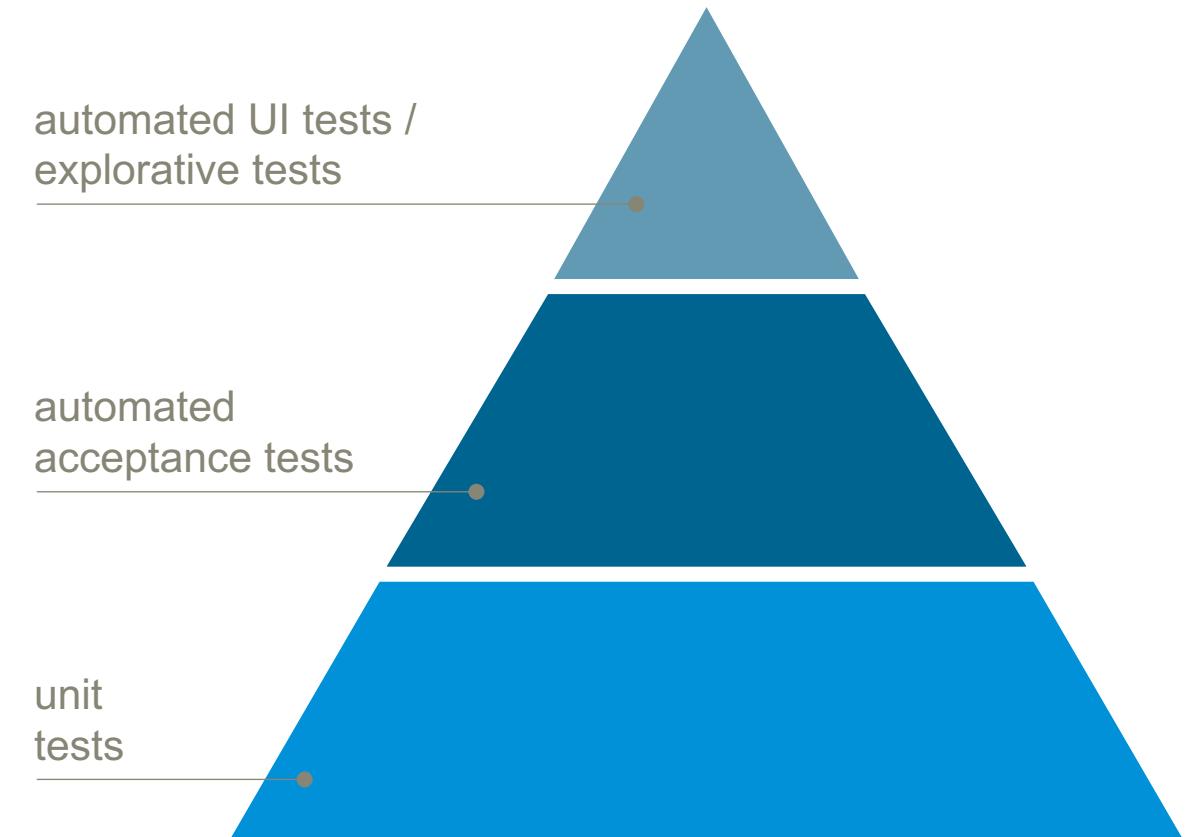
# Modern Software Development



# Modern Software Development



**Traditional** (find bugs)



**Agile** (prevent bugs)

# Rules for excellent tests

- automated \*
- atomic
- independent \*
- regressionable
- useful test data
- no conditions or repetitions
- Single-Responsibility-Principle

\* not always possible

# Why should tests be atomic?

- Each test has only two possible results (**green/red**)
- There are no semi-successful tests
- Failure of one test case means failure of the whole suite

# Why should tests be independent (isolated)?

- test sequence must be arbitrary \*
- test execution must always lead to the same result
- Don't share any state between two test cases (exception: Unchangeable status)
- test setup is performed for each test case \*
- isolation of classes or environment can be achieved with mocks

\* not always possible

# Why should tests be regressionable?

- Each test should always give the same result
- Challenge: databases and 3rd party systems must be prepared for each test case

# Why should tests not contain conditions?

- all input parameters are well known
  - test behavior is always predictable
  - distinct result
- 
- Hint: Split test cases

# Why should tests not contain repetitions?

- Variant 1: few repetitions 
- Variant 2: hundreds of repetitions 
- Variant 3: unknown number of repetitions 
- The result of a test has to be always predictable → simplify the test case!

# Why should the single-responsibility principle apply for tests?

- A test case is only responsible for one scenario
- test behaviour instead of methods
- method with different behaviour > provide multiple test cases
- one behavior which requires multiple method calls > one single test case
- Multiple assertions per test are allowed if they test the same behavior

# How do we test an application?

# Testlevel

## Unit Tests

fast, stable, easy to write  
refactoring is simple

## Integration Tests

test running application incl. databases,  
frameworks and interfaces  
Uses mocks for 3rd party systems

## System Tests

test the whole system by simulating a  
user or a process  
but: slow, unstable, high maintenance

# Testlevel

## Unit Tests

fast, stable, easy to write  
refactoring is simple

## Integration Tests

test running application incl. databases,  
frameworks and interfaces  
Uses mocks for 3rd party systems

## System Tests

test the whole system by simulating a  
user or a process  
but: slow, unstable, high maintenance

# Which options do I have for integration tests?

\* in the Java eco system

# Integration tests for Java applications

- Spring-test  
**great test-framework, but only for Spring applications ;(**
- Arquillian  
**The alien!**
- Artifact in target runtime environment  
**that's close to production ;)**

Nicht sicher — arquillian.org

arquillian

Invasion! Features Guides Docs Blog Community Modules

So you can rule your code. Not the bugs.

No more mocks. No more container lifecycle and deployment hassles.  
Just *real* tests!

Get Started!

Alex Soto Bueno Jason Porter Andrew Gumbrecht  
presents:

**Testing Java Microservices**

*Testing Java Microservices* teaches you how to write tests for microservices in Java. You'll learn test strategies that solve the most common issues you are likely to encounter. This practical hands-on guide begins with introducing you to microservices and providing you with a simple, carefully-designed application developed using microservices principles and following some of the most common technologies such as Java EE, Spring Boot, WildFly Swarm, and Docker. Along the way, you'll learn about technologies like the Arquillian ecosystem, Wiremock, Mockito, AssertJ, Pact or Gatling. Finally, you'll see how everything fits together into the Continuous Delivery pipeline.

Order now »

What's been happening lately?

<http://arquillian.org>

# Arquillian

- supports JUnit-based tests
- provides two execution modes: "In-Container" or "As-Client"
- test execution in / against real application servers (e.g. Wildfly, Websphere)
- can create (partial) test-archives (JAR, WAR, EAR) at runtime
- both curse and blessing ☺  
**sometimes hard to setup and configure**  
**partial test-archives may suggest a false sense of correctness and reliability**

# Artifact in target runtime environment

- close to production environment
- integration to build process required

**product-related maven-plugin (e.g. liberty-maven-plugin)**

**exec-maven-plugin**

- setup of environment is still exhausting ☹

# Integration tests for Java applications

- Spring-test  
great test-framework, but only for Spring applications ;(
- Arquillian  
The alien!
- Artifact in target runtime environment  
that's close to production ;)
- Artifact & infrastructure in Docker containers  
sounds even better

# Artifact & infrastructure in Docker containers

- get rid of local setup hell – no more installation or configuration
- integration to build process required

**Spotify dockerfile-maven-plugin**

**fabric8 docker-maven-plugin**

- Testcontainers

**JUnit-based tests which starts containers for the test**

# Creating Docker images for tests

Three ways to create a Docker image for a test

- manual

`docker build .`

- during maven build by using

`Spotify dockerfile-maven-plugin`

`fabric8 docker-maven-plugin`

- on-the-fly during Testcontainers test execution

`new GenericContainer(new ImageFromDockerfile(...))`

The screenshot shows a GitHub README.md page for the Dockerfile Maven plugin. The page has a header with navigation icons, a title 'Dockerfile Maven', and a status bar showing 'build failing', 'maven-central v1.4.13', and 'license Apache-2.0'. Below the title is a section titled 'Status: mature' with a note: 'At this point, we're not developing or accepting new features or even fixing non-critical bugs.' It describes the plugin as integrating Maven with Docker and lists design goals:

- Don't do anything fancy. Dockerfiles are how you build Docker projects; that's what this plugin uses. They are mandatory.
- Make the Docker build process integrate with the Maven build process. If you bind the default phases, when you type `mvn package`, you get a Docker image. When you type `mvn deploy`, your image gets pushed.
- Make the goals remember what you are doing. You can type `mvn dockerfile:build` and later `mvn dockerfile:tag` and later `mvn dockerfile:push` without problems. This also eliminates the need for something like `mvn dockerfile:build -DalsoPush`; instead you can just say `mvn dockerfile:build dockerfile:push`.
- Integrate with the Maven build reactor. You can depend on the Docker image of one project in another project, and Maven will build the projects in the correct order. This is useful when you want to run integration tests involving multiple services.

The page also mentions adherence to the Open Code of Conduct and provides a changelog link. A 'Set-up' section notes requirements: Java 7 or later and Apache Maven 3 or later (dockerfile-maven-plugin <=1.4.6 needs Maven >= 3, and

<https://github.com/spotify/dockerfile-maven>

```
// Spotify dockerfile-maven-plugin
<plugin>
  <groupId>com.spotify</groupId>
  <artifactId>dockerfile-maven-plugin</artifactId>
  <version>1.4.13</version>
  <configuration>
    <repository>testing-with-containers/hello-world</repository>
    <tag>${project.version}</tag>
  </configuration>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>build</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

1

2

3

4

The screenshot shows a Mac OS X desktop environment with a web browser window open. The address bar displays 'dmp.fabric8.io'. The page content is the documentation for the 'docker-maven-plugin'. On the left, there is a navigation sidebar with the following structure:

- docker-maven-plugin
  - 1. Introduction
    - 1.1. Building Images
    - 1.2. Running Containers
    - 1.3. Configuration
    - 1.4. Example
    - 1.5. Features
  - 2. Installation
  - 3. Global configuration
  - 4. Image configuration
    - 4.1. Image Names
    - 4.2. Container Names
    - 4.3. Name Patterns
      - 4.3.1. Ant-like Name Patterns
      - 4.3.2. Java Regular Expression Patterns
      - 4.3.3. Name Pattern Lists
  - 5. Maven Goals
    - 5.1. docker:build
      - 5.1.1. Configuration
      - 5.1.2. Assembly
      - 5.1.3. Startup Arguments
      - 5.1.4. Build Args
      - 5.1.5. Healthcheck
    - 5.2. docker:start
      - 5.2.1. Configuration
      - 5.2.2. Environment and Labels
      - 5.2.3. Port Mapping
      - 5.2.4. Links
      - 5.2.5. Network
      - 5.2.6. Depends-On
      - 5.2.7. Restart Policy

# fabric8io/docker-maven-plugin

Roland Huß – Version 0.33.0, 2020-01-21

© 2015 - 2019 The original authors.

## 1. Introduction

This is a Maven plugin for managing Docker images and containers. It focuses on two major aspects for a Docker build integration:

### 1.1. Building Images

One purpose of this plugin is to create Docker images holding the actual application. This is done with the [docker:build](#) goal. It is easy to include build artefacts and their dependencies into an image.

Several ways for configuring the builds are supported:

- An own configuration syntax can be used to create a Dockerfile. For specifying artefacts and other files, the plugin uses the assembly descriptor format from the maven-assembly-plugin to copy over those file into the Docker image.
- An external Dockerfile can be specified in which Maven properties can be inserted. This is also the default mode, if only a single image should be built and a top-level `Dockerfile` exists. See [Simple Dockerfile build](#) for details of this zero XML configuration mode.

Images that are built with this plugin can be pushed to public or private Docker registries with [docker:push](#).

### 1.2. Running Containers

With this plugin it is possible to run completely isolated integration tests so you don't need to take care of shared resources. Ports can be mapped dynamically and made available as Maven properties to your integration test code.

<https://dmp.fabric8.io>

```
// fabric8 docker-maven-plugin
<plugin>
  <groupId>io.fabric8</groupId>
  <artifactId>docker-maven-plugin</artifactId>
  <version>0.33.0</version>
  <executions>
    <execution>
      <id>postman-tests</id>
      <phase>post-integration-test</phase>
      <goals>
        <goal>build</goal>
        <goal>start</goal>
      </goals>
      <configuration>...</configuration>
    </execution>
    ...
  </executions>
</plugin>
```

1

2

3

The screenshot shows a web browser window with the URL [testcontainers.org](http://testcontainers.org) in the address bar. The page has a blue header with the "Testcontainers" logo, a search bar, and a GitHub icon for the repository "testcontainers-java" (3.2k Stars - 582 Forks). The main content area features a large image of a 3D cube with the text "TESTCONTAINERS" below it. To the left is a sidebar with navigation links: Home, Quickstart, Features, Modules, Test framework integration, System Requirements, Getting help, and Contributing. To the right is a "Table of contents" sidebar with links to About, Prerequisites, Maven dependencies, Who is using Testcontainers?, License, Attributions, and Copyright.

# Testcontainers



## About

Testcontainers is a Java library that supports JUnit tests, providing lightweight, throwaway instances of common databases, Selenium web browsers, or anything else that can run in a Docker container.

Testcontainers make the following kinds of tests easier:

- **Data access layer integration tests:** use a containerized instance of a MySQL, PostgreSQL or Oracle database to test your data access layer code for complete compatibility, but without requiring complex setup on developers' machines and safe in the knowledge that your tests will

<http://testcontainers.org/>

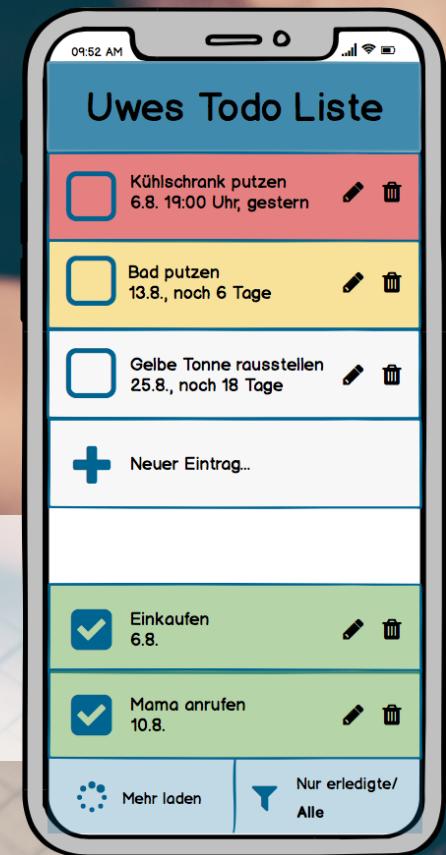
# Testcontainers

- supports JUnit-based tests
- provides throwaway instances of anything that can run in a Docker container
- makes the following kinds of tests easier
  - Data access layer integration tests**
  - Application integration tests**
  - UI/Acceptance tests**

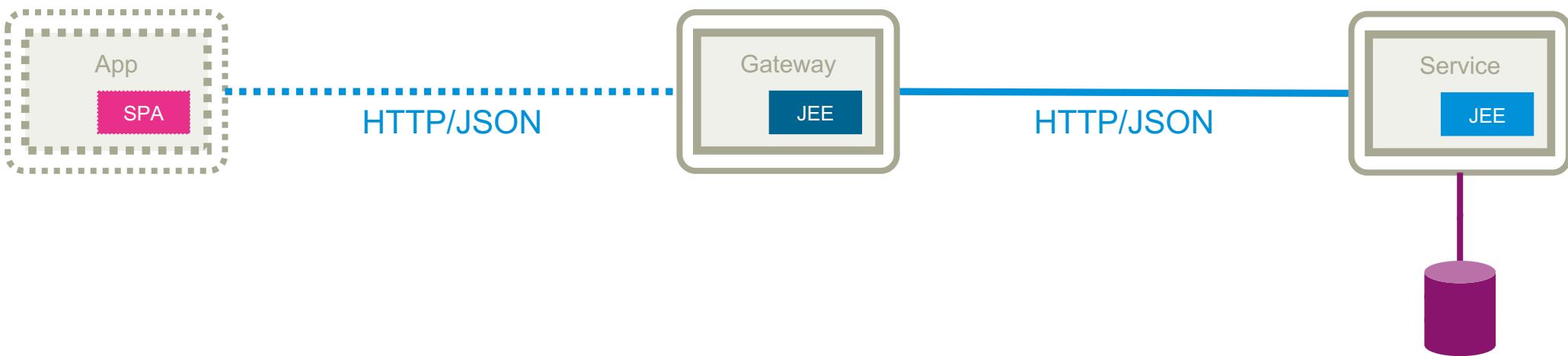
# Testcontainers

- supports JUnit 4/5 & manual container lifecycle control
- enables creation of docker images on-the-fly
- pass environment variables and files to container during startup
- exclusive network and communication between containers if necessary
- provides ready-to-use containers for database, messaging and other infrastructure

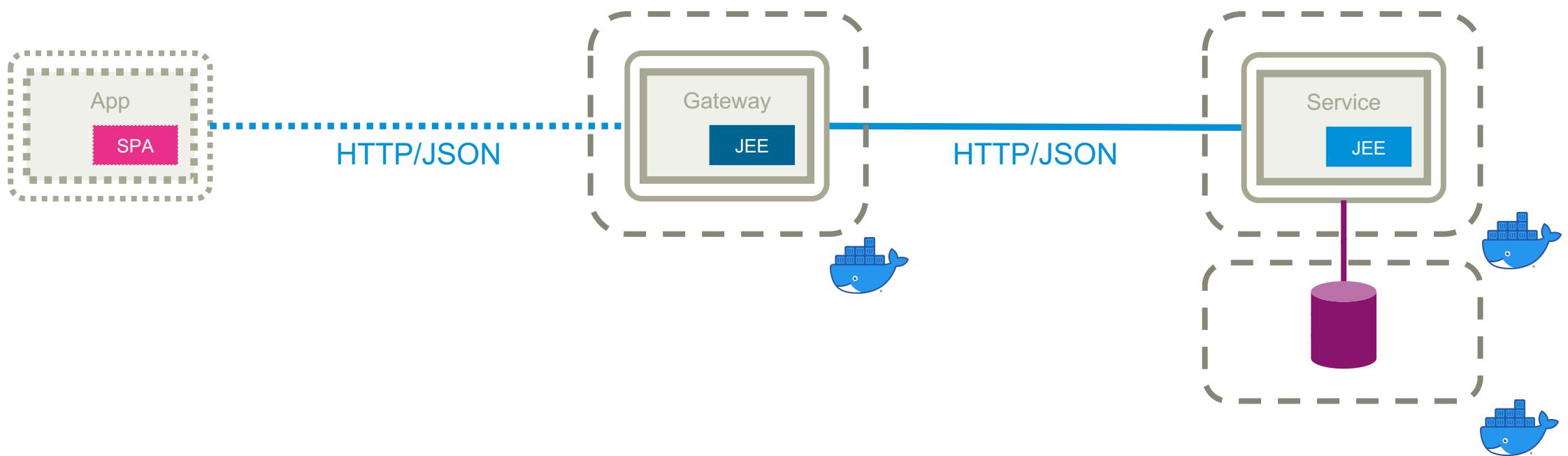
# Demo application: Todo-List



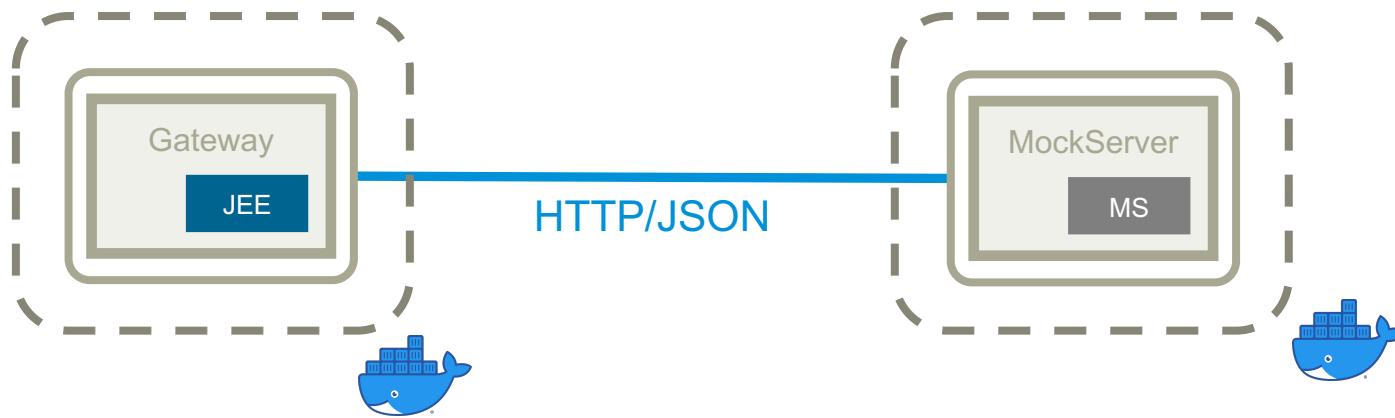
# Todo-List



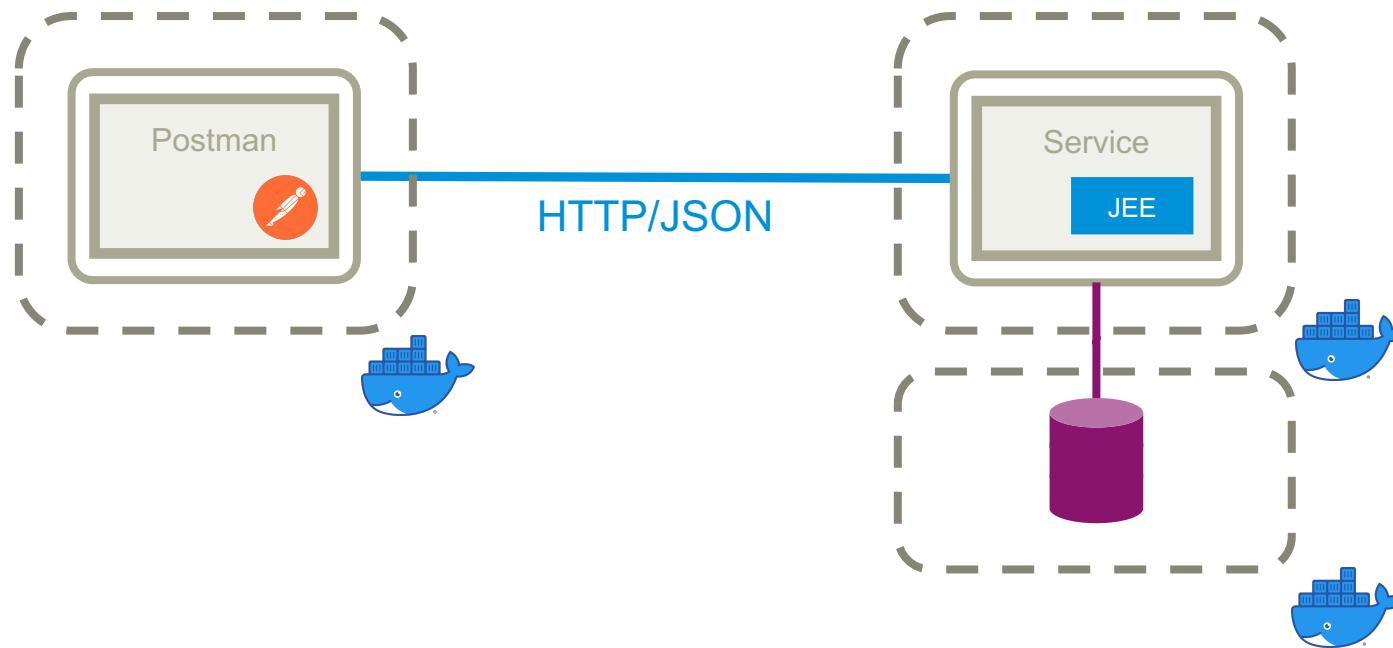
# Todo-List



# Todo-List (Gateway with MockServer)



# Todo-List (Postman + Service + Database)



```
// Testcontainer - Creating a container
@Testcontainers
public class HelloWorldResourceIT {

    @Container
    private static final GenericContainer<?> CONTAINER =
1      new GenericContainer("testing-with-containers/hello-world:0")
2      .withExposedPorts(9080);

    @Test
    public void sayHello() { ... }
}
```

[https://www.testcontainers.org/features/creating\\_container/](https://www.testcontainers.org/features/creating_container/)

```
// Testcontainer - JUnit5 Extension  
1  @Testcontainers  
public class HelloWorldResourceIT {  
  
2  @Container  
private static final GenericContainer<?> CONTAINER =  
3  new GenericContainer("testing-with-containers/hello-world:0")  
    .withExposedPorts(9080);  
  
@Test  
public void sayHello() { ... }  
}
```

[https://www.testcontainers.org/test\\_framework\\_integration/junit\\_5/](https://www.testcontainers.org/test_framework_integration/junit_5/)

```
// Testcontainer - JUnit4 ClassRule
public class HelloWorldResourceIT {

    1 @ClassRule
    2     public static GenericContainer<?> container =
        new GenericContainer("testing-with-containers/hello-world:0")
            .withExposedPorts(9080);

    @Test
    public void sayHello() { ... }

}
```

[https://www.testcontainers.org/test\\_framework\\_integration/junit\\_4/](https://www.testcontainers.org/test_framework_integration/junit_4/)

```
// Testcontainer - JUnit4 Rule
public class HelloWorldResourceIT {

    1 @Rule
    2 public GenericContainer<?> container =
        new GenericContainer("testing-with-containers/hello-world:0")
            .withExposedPorts(9080);

    @Test
    public void sayHello() { ... }

}
```

[https://www.testcontainers.org/test\\_framework\\_integration/junit\\_4/](https://www.testcontainers.org/test_framework_integration/junit_4/)

```
// Testcontainer - manual container lifecycle control
public class HelloWorldResourceIT {

    1  private static final GenericContainer<?> CONTAINER = ...

    @BeforeAll
    private void beforeAll() {
        2  CONTAINER.start();
    }

    @AfterAll
    private void afterAll() {
        3  CONTAINER.stop();
    }
}
```

[https://www.testcontainers.org/test\\_framework\\_integration/manual\\_lifecycle\\_control/](https://www.testcontainers.org/test_framework_integration/manual_lifecycle_control/)

```
// Testcontainer - Get container ip address & mapped ports
@Testcontainers
public class HelloWorldResourceIT {

    @Container
    1 private static final GenericContainer<?> CONTAINER = ...

    private static URI uri;

    @BeforeAll
    public static void setUpUri() {
        uri = UriBuilder.fromPath("hello-world")
            .scheme("http")
            2 .host(CONTAINER.getContainerIpAddress())
            3 .port(CONTAINER.getFirstMappedPort())
            .build();
    }
}
```

```
// Testcontainer - accessing container logs
@Testcontainers
public class HelloWorldResourceIT {

    1  private static final Logger LOG = LoggerFactory.getLogger(...);

    @Container
    private static final GenericContainer<?> CONTAINER =
        new GenericContainer("testing-with-containers/hello-world:0")
            .withExposedPorts(9080)
    2  .withLogConsumer(new Slf4jLogConsumer(LOG));

    ...
}
```

[https://www.testcontainers.org/features/container\\_logs/](https://www.testcontainers.org/features/container_logs/)

```
# Hello-World Dockerfile
FROM openjdk:8-jre

ADD target/hello-world.jar /opt/hello-world.jar

ENV JAVA_OPTS="-Djava.net.preferIPv4Stack=true -Djava.net.preferIPv4Addresses=true"

EXPOSE 9080
ENTRYPOINT exec java $JAVA_OPTS -jar /opt/hello-world.jar
```

```
// Testcontainer - Create docker image on-the-fly
@Testcontainers
public class HelloWorldResourceIT {

    @Container
    private static final GenericContainer<?> CONTAINER =
        new GenericContainer(
            new ImageFromDockerfile()
                .withFileFromFile("Dockerfile", new File("Dockerfile"))
                .withFileFromFile("target/hello-world.jar",
                    new File("target/hello-world.jar")))
                .withExposedPorts(9080);

    ...
}
```

[https://www.testcontainers.org/features/creating\\_images/](https://www.testcontainers.org/features/creating_images/)

1  
2  
3  
4  
5  
6  
7

```
// Testcontainer - Create docker image on-the-fly with builder
@Testcontainers
public class HelloWorldResourceIT {

    @Container
    private static final GenericContainer<?> CONTAINER =
        new GenericContainer(new ImageFromDockerfile()
            .withDockerfileFromBuilder(builder -> builder
                .from("openjdk:8-jre")
                .add("target/hello-world.jar", "/opt/hello-world.jar")
                .expose(9080)
                .entryPoint("exec java -jar /opt/hello-world.jar")
                .build())
            .withFileFromFile("target/hello-world.jar",
                new File("target/hello-world.jar")))

    ...
}
```

[https://www.testcontainers.org/features/creating\\_images/](https://www.testcontainers.org/features/creating_images/)

```
// Testcontainer - GenericContainer with fixed mapped port
public class TodoRepositoryIT {

    @ClassRule
    public static FixedHostPortGenericContainer<?> container =
        new FixedHostPortGenericContainer("postgres:12-alpine")
            .withExposedPorts(5432)
            .withFixedExposedPort(5432, 5432)
            .withEnv("POSTGRES_DB", "postgres")
            .withEnv("POSTGRES_USER", "postgres")
            .withEnv("POSTGRES_PASSWORD", "postgres")
            .withClasspathResourceMapping("docker/1-schema.sql",
                "/docker-entrypoint-initdb.d/1-schema.sql",
                BindMode.READ_ONLY)

    ...
}
```

```
// Testcontainer - Pass environment variables & file
public class TodoRepositoryIT {

    @ClassRule
    public static GenericContainer<?> container =
        new GenericContainer("postgres:12-alpine")
            .withExposedPorts(5432)
            .withEnv("POSTGRES_DB", "postgres")
            .withEnv("POSTGRES_USER", "postgres")
            .withEnv("POSTGRES_PASSWORD", "postgres")
            .withClasspathResourceMapping("docker/1-schema.sql",
                "/docker-entrypoint-initdb.d/1-schema.sql",
                BindMode.READ_ONLY)
}
```

1  
1  
1  
2

<https://www.testcontainers.org/features/files/>

```
// Testcontainer - Access database container (test class)
public class TodoRepositoryIT {

    @ClassRule
    public static GenericContainer<?> container =
        new GenericContainer("postgres:12-alpine")
            .withExposedPorts(5432)
            .withEnv("POSTGRES_DB", "postgres")
            .withEnv("POSTGRES_USER", "postgres")
            .withEnv("POSTGRES_PASSWORD", "postgres")
            .withClasspathResourceMapping("docker/1-schema.sql",
                "/docker-entrypoint-initdb.d/1-schema.sql",
                BindMode.READ_ONLY)

    ...
}
```

<https://www.testcontainers.org/modules/databases/>

```
// Testcontainer - Access database container (persistence.xml)
<persistence-unit name="postgres-db" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>de.openknowledge.projects.todolist.service.domain.Todo</class>

    <properties>
        1      <property name="javax.persistence.jdbc.driver"
                    value="org.testcontainers.jdbc.ContainerDatabaseDriver"/>
        2      <property name="javax.persistence.jdbc.url"
                    value="jdbc:postgresql://localhost:5432/postgres"/>
        3      <property name="javax.persistence.jdbc.user" value="postgres"/>
        4      <property name="javax.persistence.jdbc.password" value="postgres"/>
    </properties>
</persistence-unit>
```

<https://www.testcontainers.org/modules/databases/>

```
// Testcontainer - PostgresModule (test class)
public class TodoRepositoryIT {

    @ClassRule
1    public static PostgreSQLContainer<?> database =
        new PostgreSQLContainer<?>();

    ...
}
```

<https://www.testcontainers.org/modules/databases/postgres/>

```
// Testcontainer - PostgresModule (persistence.xml)
<persistence-unit name="postgres-test" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>de.openknowledge.projects.todolist.service.domain.Todo</class>

    <properties>
        1      <property name="javax.persistence.jdbc.driver"
                    value="org.testcontainers.jdbc.ContainerDatabaseDriver"/>
        2      <property name="javax.persistence.jdbc.url"
                    value="jdbc:tc:postgresql://localhost/test?
                        TC_INITSCRIPT=docker/1-schema.sql "/>
        3      <property name="javax.persistence.jdbc.user" value="test"/>
        4      <property name="javax.persistence.jdbc.password" value="test"/>
    </properties>
</persistence-unit>
```

<https://www.testcontainers.org/modules/databases/>

```
// Testcontainer - Networking and communication with Containers
public class TodoResourceIT {

    1  private static final Network NETWORK = Network.newNetwork();

    @ClassRule
    public static GenericContainer database = new GenericContainer(...)
        .withExposedPorts(5432)
        2  .withNetwork(NETWORK)
        3  .withNetworkAliases("database")
        ....;

    @ClassRule
    public static GenericContainer service = new GenericContainer(...)
        .withExposedPorts(9080)
        4  .withNetwork(NETWORK)
        5  .dependsOn(database)
        ....;
}
```

<https://www.testcontainers.org/features/networking/>

```
// Testcontainer - Waiting for containers to start or be ready
public class TodoResourceIT {

    private static final Network NETWORK = Network.newNetwork();

    @ClassRule
    public static GenericContainer database = new GenericContainer(...)
        .withExposedPorts(5432)
        .waitingFor(Wait.forLogMessage(".*server started.*", 1))
        ....;

    @ClassRule
    public static GenericContainer service = new GenericContainer(...)
        .withExposedPorts(9080)
        .waitingFor(Wait.forHttp("/todo-list-service/api/todos"))
        ....;
}
```

[https://www.testcontainers.org/features/startup\\_and\\_waits/](https://www.testcontainers.org/features/startup_and_waits/)

```
// Testcontainer - Docker Compose Module
public class TodoResourceIT {

    @ClassRule
    public static DockerComposeContainer environment =
        new DockerComposeContainer(new File("./docker-compose.yml"))
            .withExposedService("database", 5432)
            .withExposedService("service", 9080);
            .withLogConsumer("database", new Slf4jLogConsumer(LOG))
            .withLogConsumer("service", new Slf4jLogConsumer(LOG));

    ...
}
```

[https://www.testcontainers.org/modules/docker\\_compose/](https://www.testcontainers.org/modules/docker_compose/)

```
// Testcontainer - Docker Compose Module
public class TodoResourceIT {

    @ClassRule
    1 private static final DockerComposeContainer environment = ...

    private static URI uri;

    @BeforeClass
    public static void setUpUri() {
        uri = UriBuilder.fromPath("hello-world")
            .scheme("http")
            2 .host(environment.getServiceHost("service", 9080))
            3 .port(environment.getServicePort("service", 9080))
            .build();
    }
}
```

```
// Testcontainer - Mockserver Module
@Testcontainers
public class TodoGatewayResourceIT {

    private static final Network NETWORK = Network.newNetwork();

    @Container
1    private static final MockServerContainer MOCKSERVER =
        new MockServerContainer()
        .withNetwork(NETWORK)
2    .withNetworkAliases ("mockserver");

    @Container
    private static final GenericContainer<?> GATEWAY =
        GatewayContainer.newContainer()
            .withEnv ("SERVICE_HOST", "mockserver")
            .withEnv ("SERVICE_PORT", 1080)
            .withNetwork(NETWORK)
            ....;

}
```

<https://www.testcontainers.org/modules/mockserver/>

# QUESTIONS



# Contact



**OFFENKUNDIGGUT**

---

**STEPHAN MÜLLER**

SITE MANAGER – LOCATION ESSEN

[stephan.mueller@openknowledge.de](mailto:stephan.mueller@openknowledge.de)

+49 (0)441 4082 – 0

@\_openknowledge

# Bildnachweise



*All pictures inside this presentation originate from pixabay.com or were created by my own.*