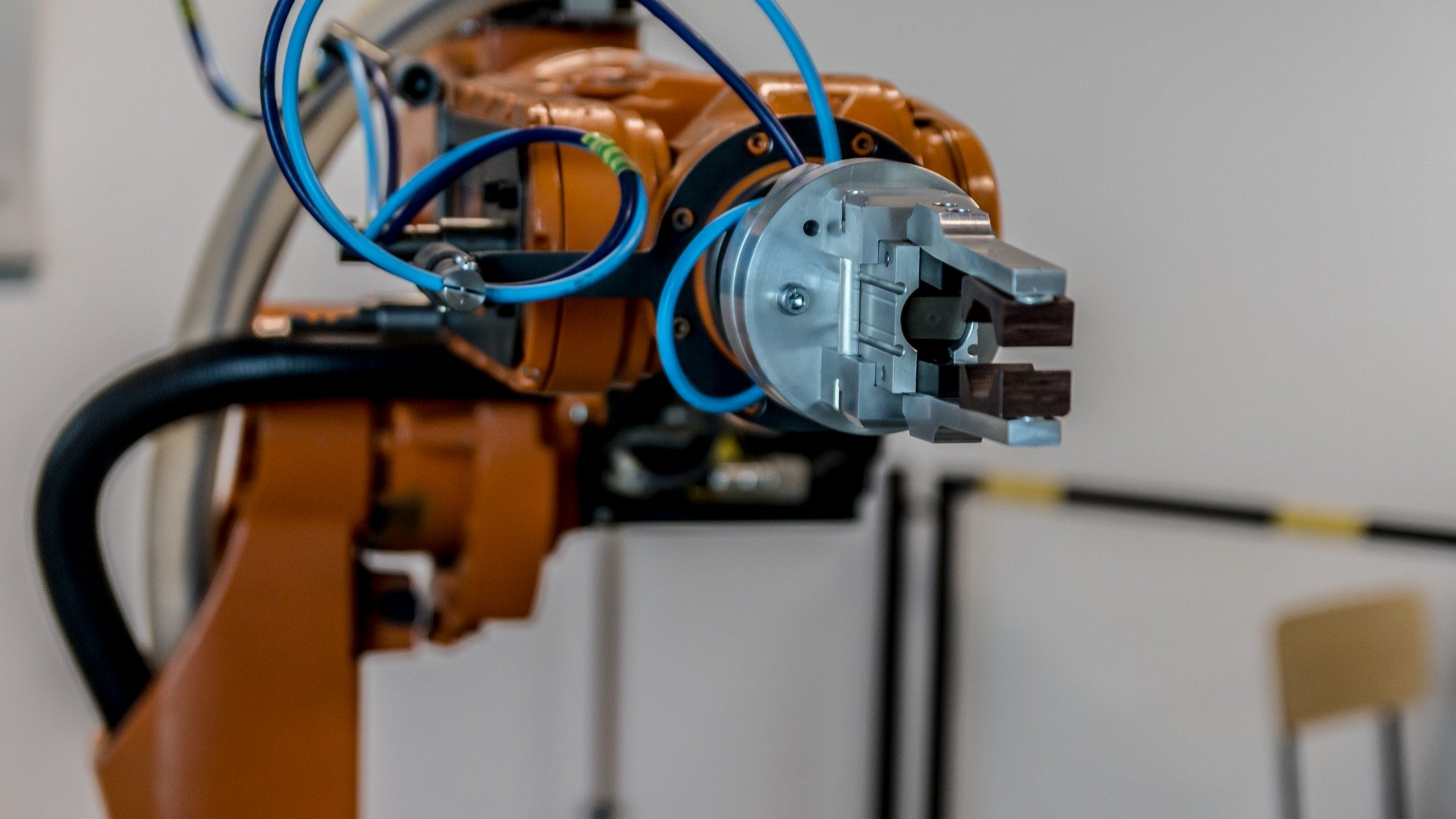

#WISSENTEILEN
@_openKnowledge
TESTING
WITH
CONTAINERS
Stephan Müller, Christian Wansart | open knowledge GmbH

Demo application & exercises

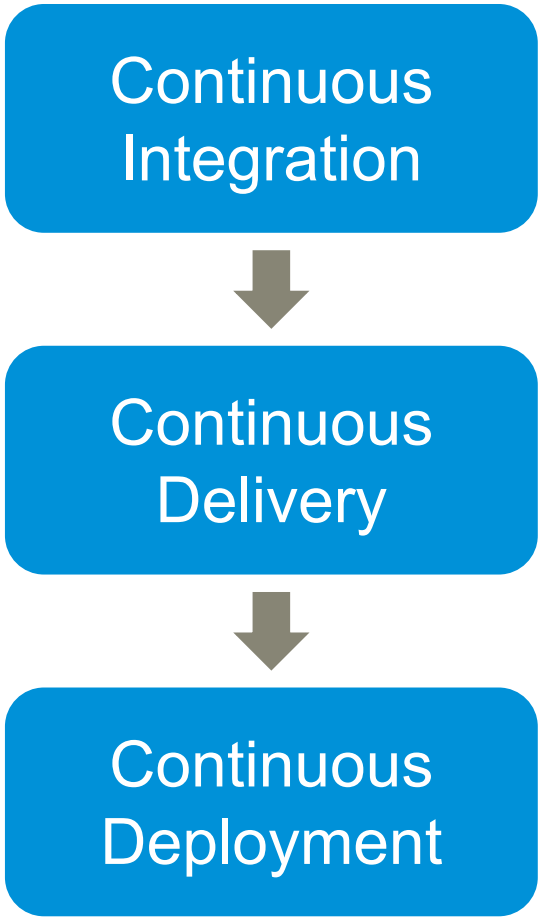
https://github.com/stephan-mueller/testing-with-containers

Amazon deploys to production every 11,6s*

* In 2011. 2018 Amazon made 23.000 deployments a day



Modern Software Development

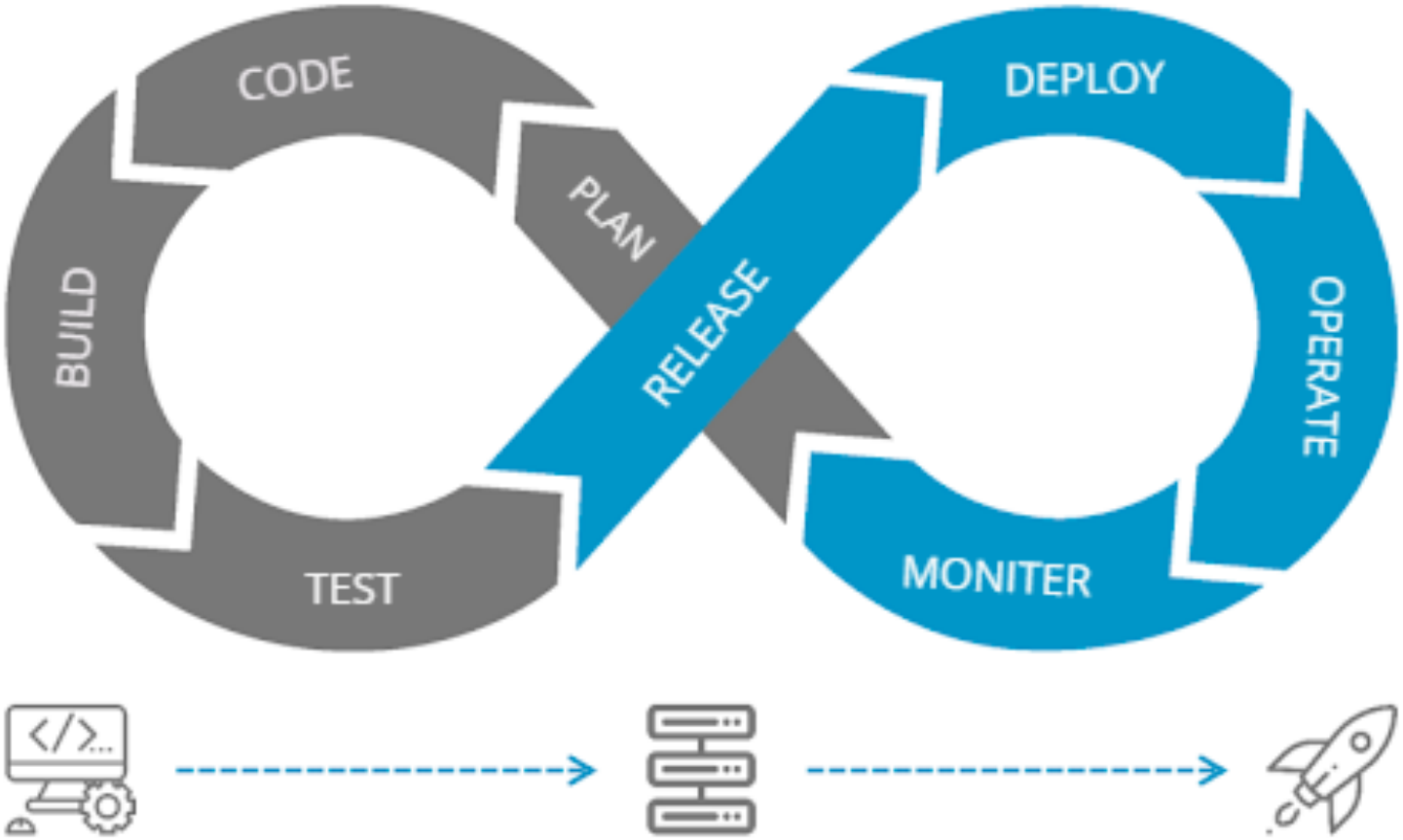


automated builds and tests for every change

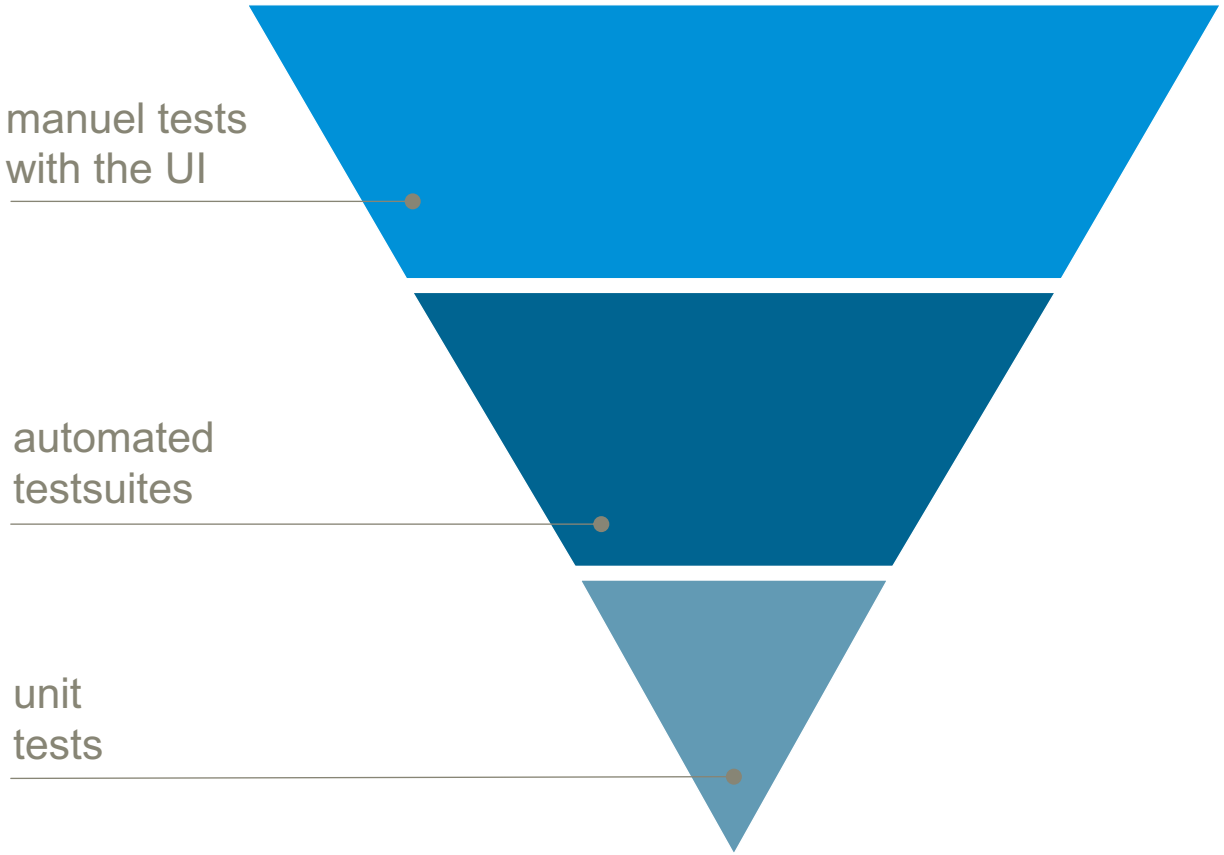
build release artifact for every change, deployment with a single click

each successful tested change is deployed to produktion automatically

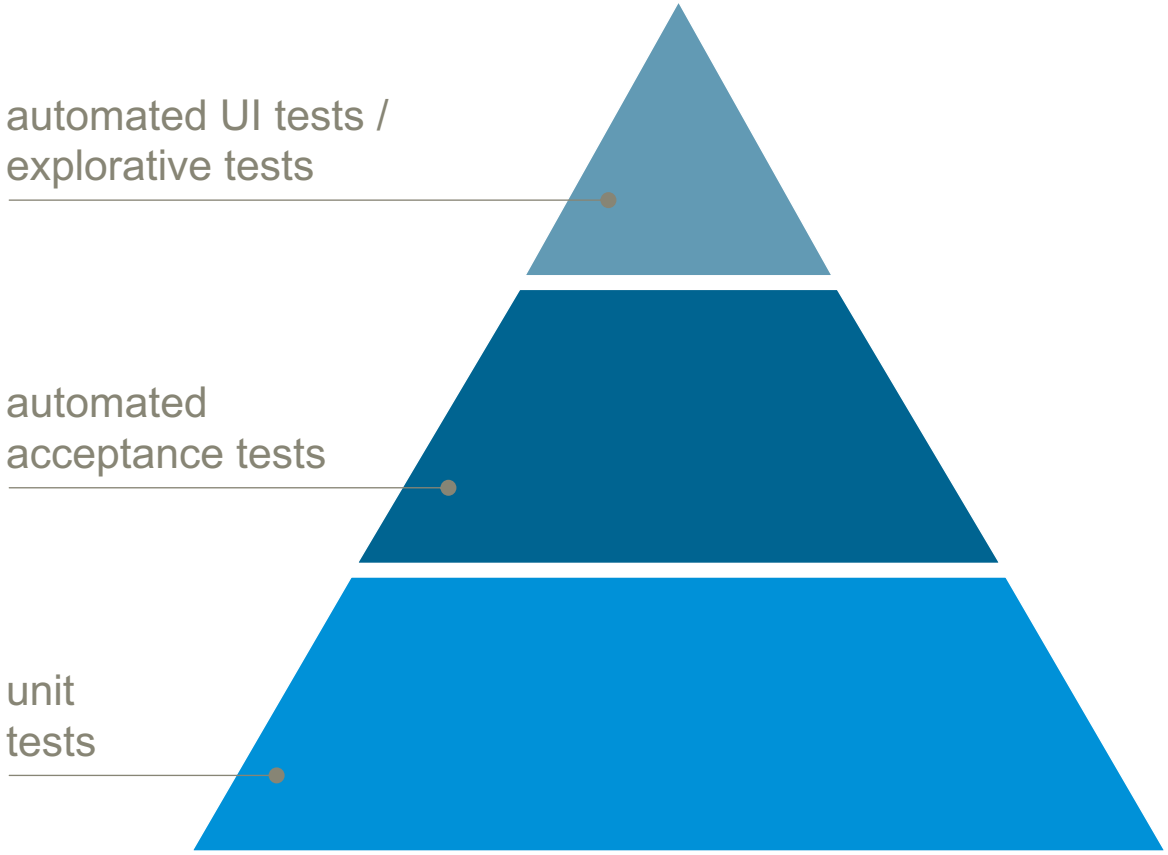
Modern Software Development



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 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

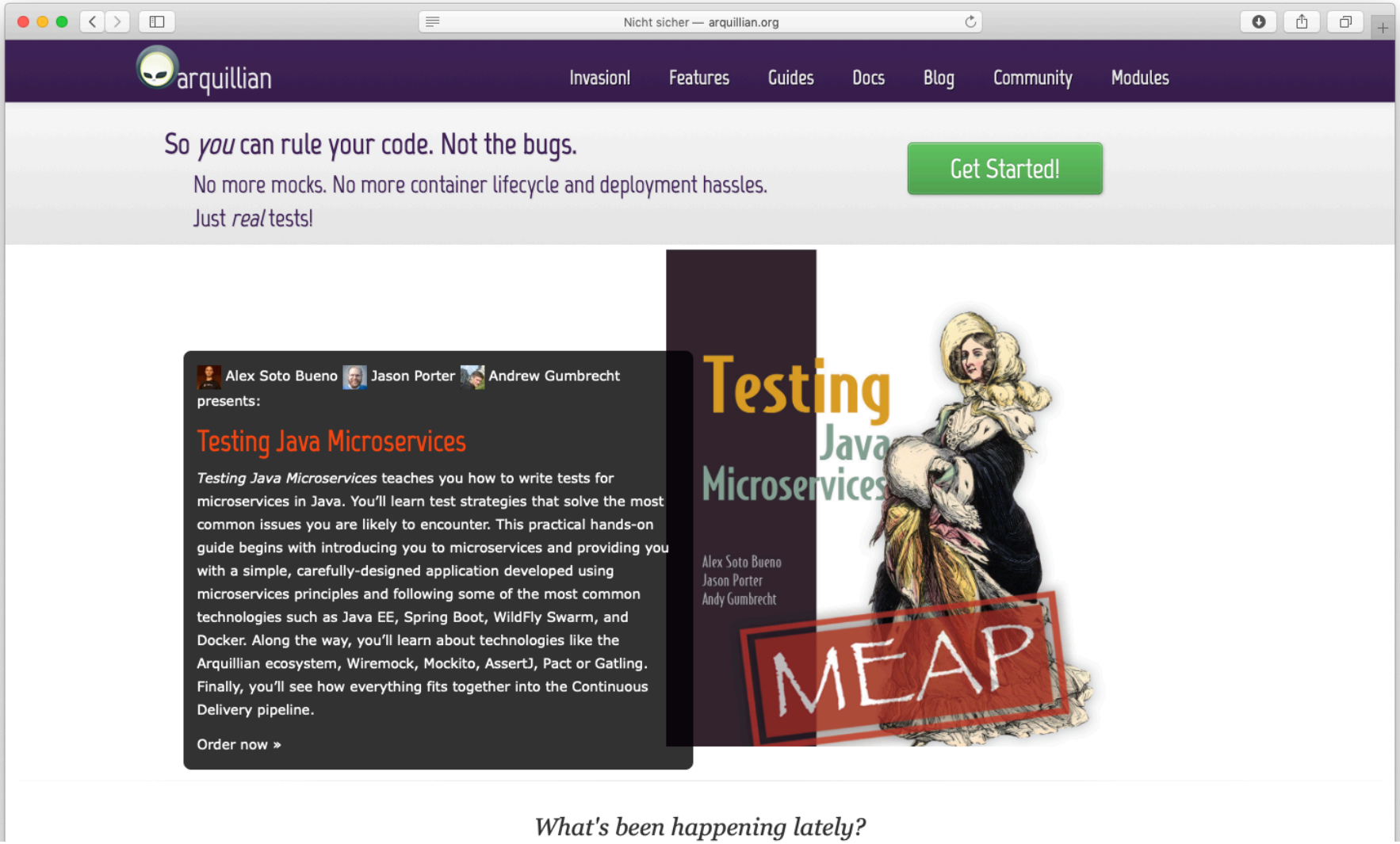
  `great test-framework, but only for Spring applications ;(`

- Arquillian

  `The alien!`

- Artifact in target runtime environment

  `that's close to production ;)`



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

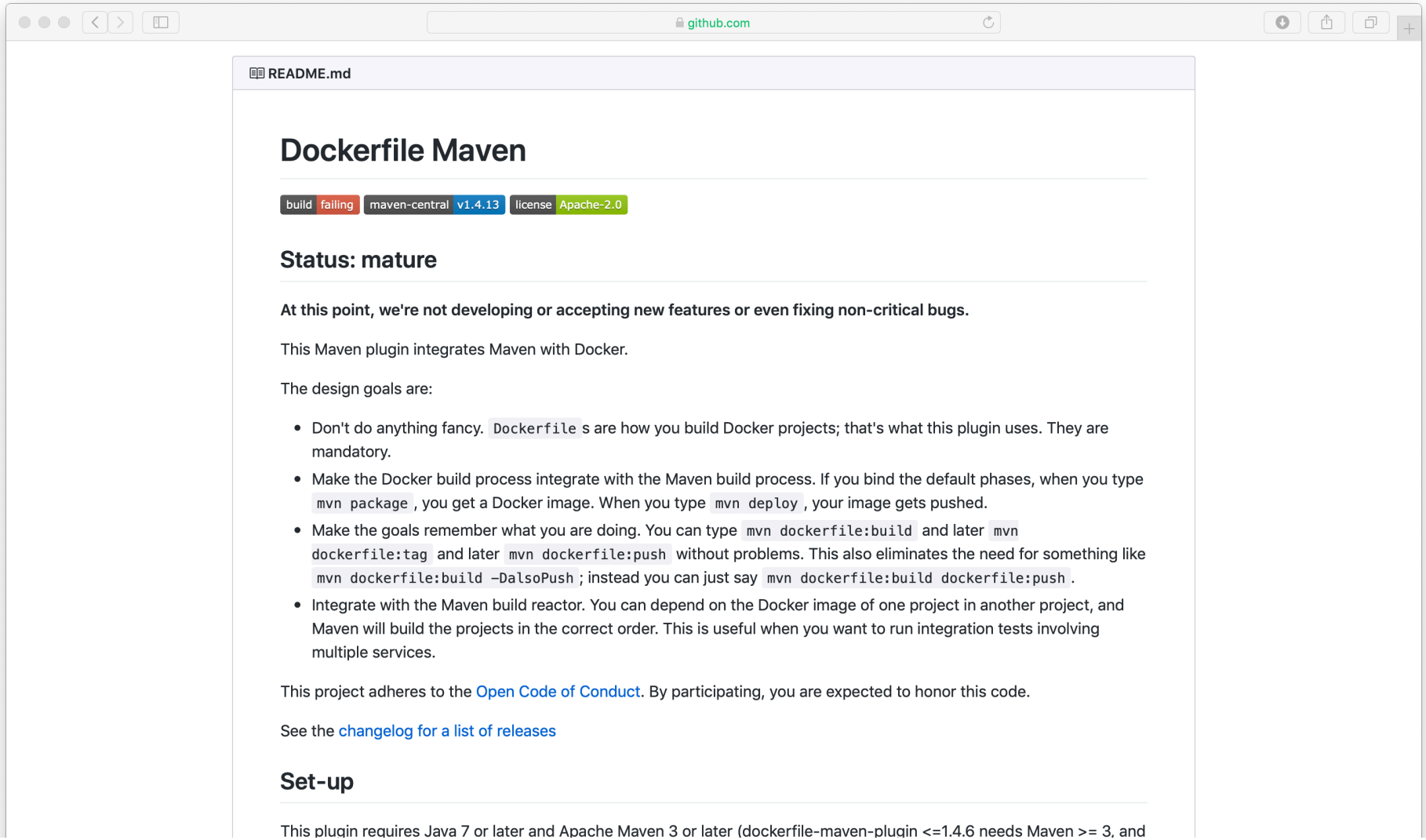
  ```
  docker build .
  ```

- during maven build by using

  ```
  Spotify dockerfile-maven-plugin
  fabric8 docker-maven-plugin
  ```

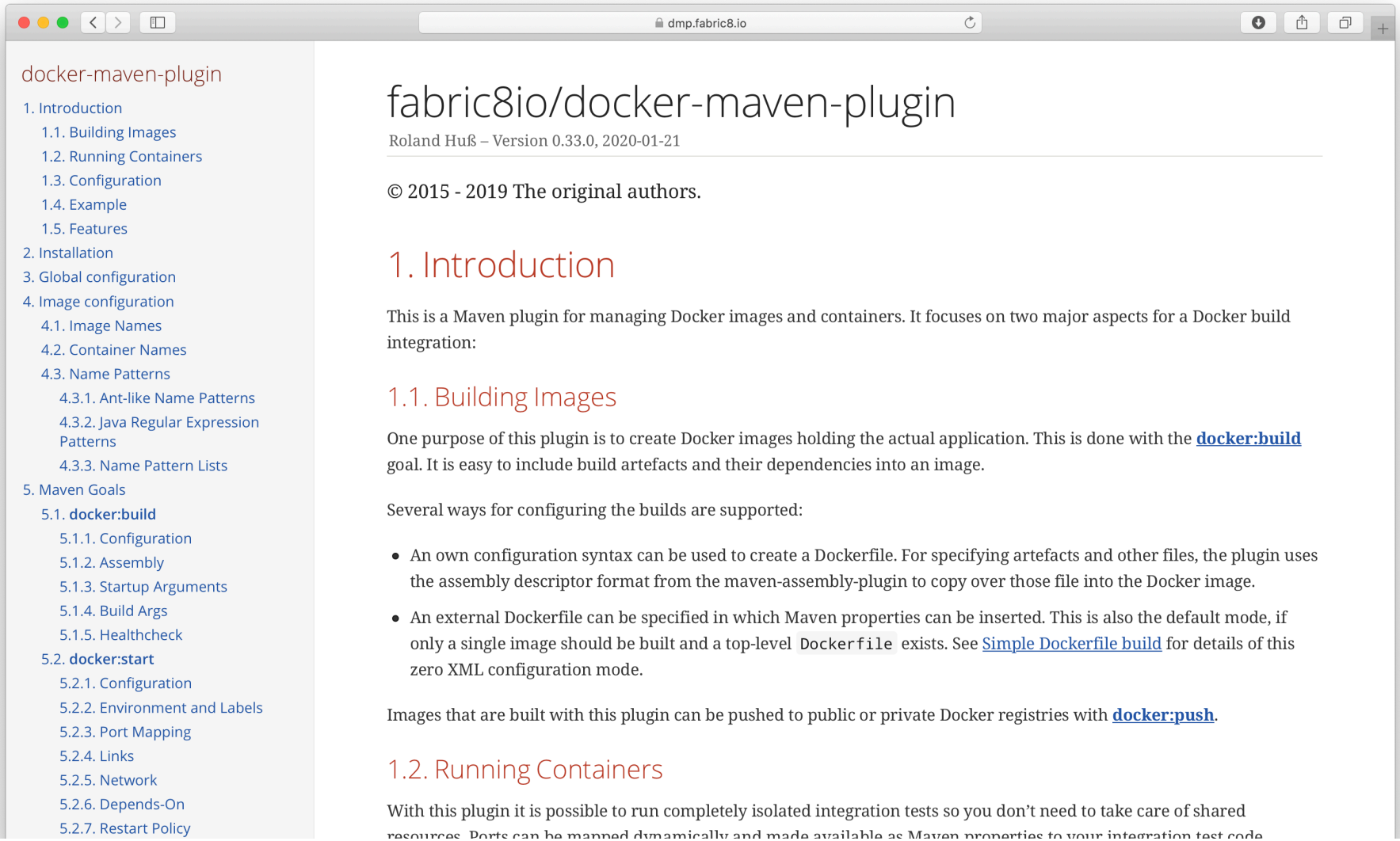
- on-the-fly during Testcontainers test execution

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



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

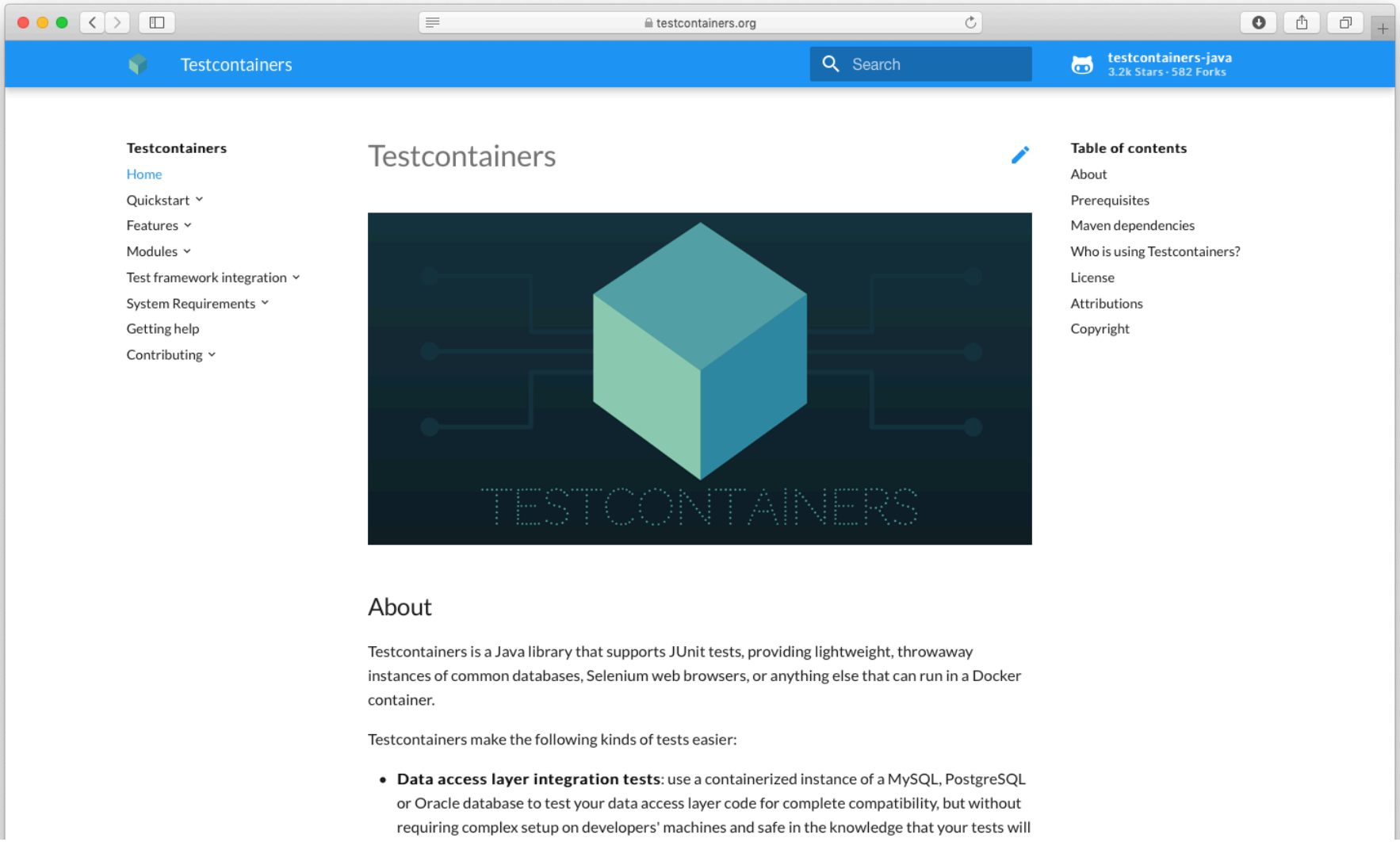
```
// Spotify dockerfile-maven-plugin
<plugin>
    <groupId>com.spotify</groupId>
    <artifactId>dockerfile-maven-plugin</artifactId>
    <version>1.4.13</version>
    <configuration>
1       <repository>testing-with-containers/hello-world</repository>
2       <tag>${project.version}</tag>
    </configuration>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



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



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

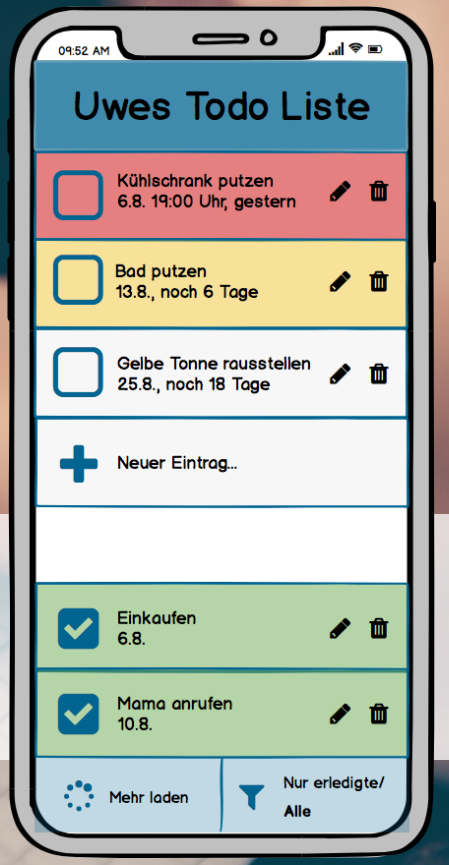
  Application integration tests

  UI/Acceptance tests

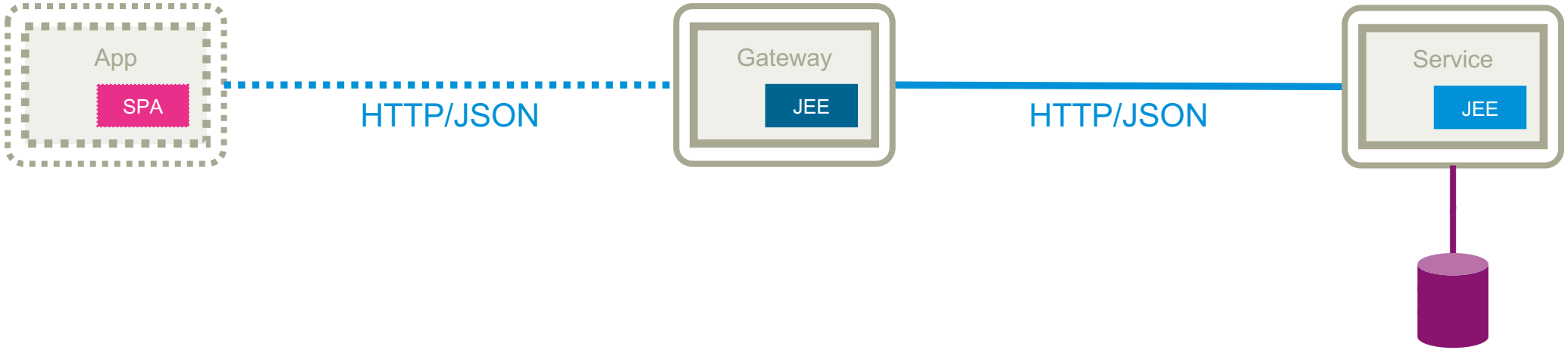
Testcontainers

- 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

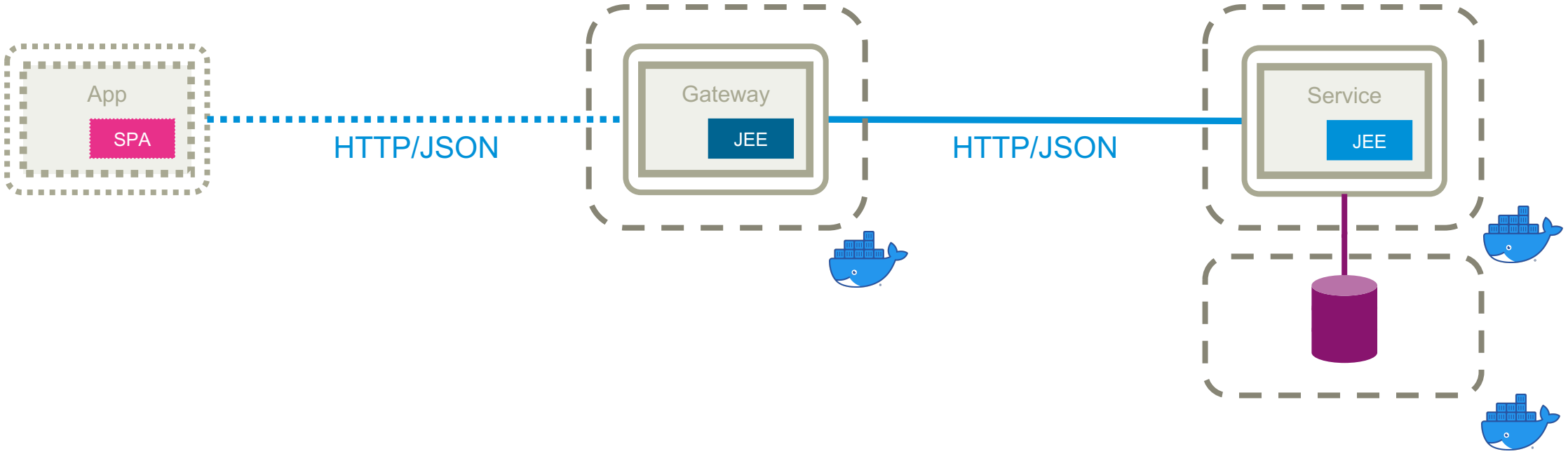
Demo application: Todo-List



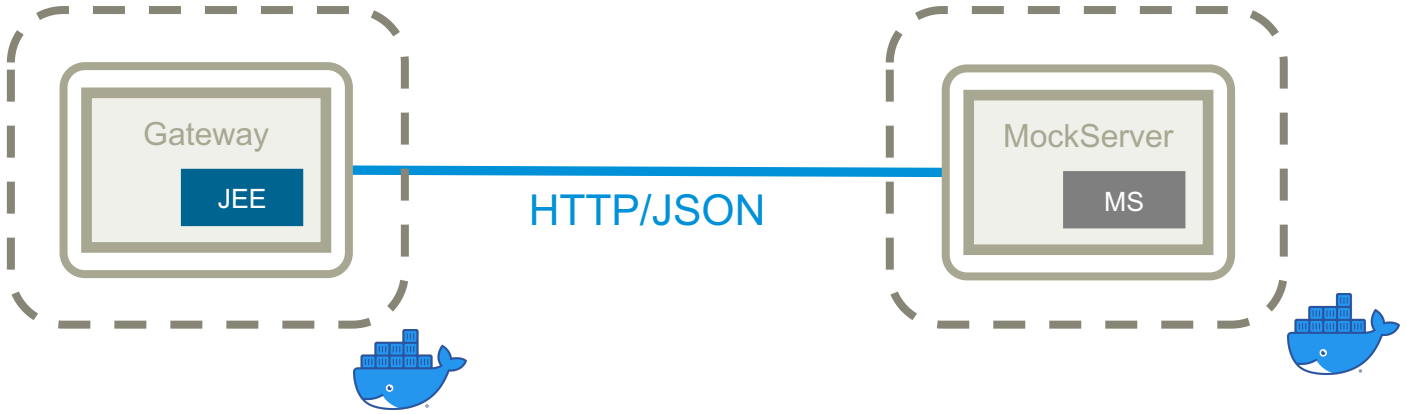
Todo-List



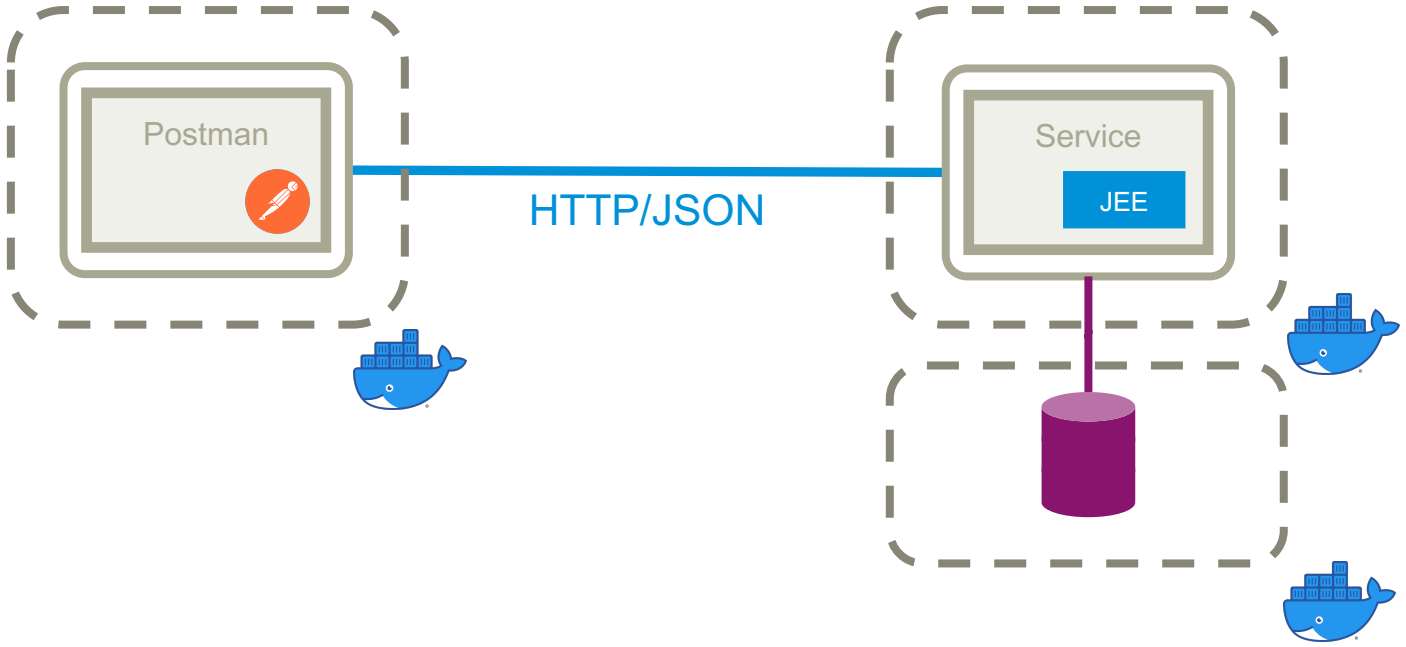
Todo-List



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/

```
// 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/

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

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

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

1
2

https://www.testcontainers.org/test_framework_integration/junit_4/

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

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

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

1
2

https://www.testcontainers.org/test_framework_integration/junit_4/

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

  private static final GenericContainer<?> CONTAINER = …

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

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

1
2
3

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 {

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

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

  …
}
```

1

2

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);

  …
}
```

1
2
3
4

https://www.testcontainers.org/features/creating_images/

```
// 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")))

  ...
}
```

1
2
3
4
5
6
7

https://www.testcontainers.org/features/creating_images/

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

  @ClassRule
1 public static FixedHostPortGenericContainer<?> container =
    new FixedHostPortGenericContainer("postgres:12-alpine")
2     .withExposedPorts(5432)
3     .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)
  …
}
```

1
2
3
4

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)
1     .waitingFor(Wait.forLogMessage(".*server started.*", 1))
      .…;

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

https://www.testcontainers.org/features/startup_and_waits/

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

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

  …
}
```

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()
3     **.withEnv("SERVICE_HOST", "mockserver")**
3     **.withEnv("SERVICE_PORT", 1080)**
      .withNetwork(NETWORK)
      .…;
}
```

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

QUESTIONS

Contact

OFFENKUNDIG**GUT**

STEPHAN MÜLLER
SITE MANAGER – LOCATION ESSEN

stephan.mueller@openknowledge.de
+49 (0)441 4082 – 0

@_openknowledge

Bildnachweise

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