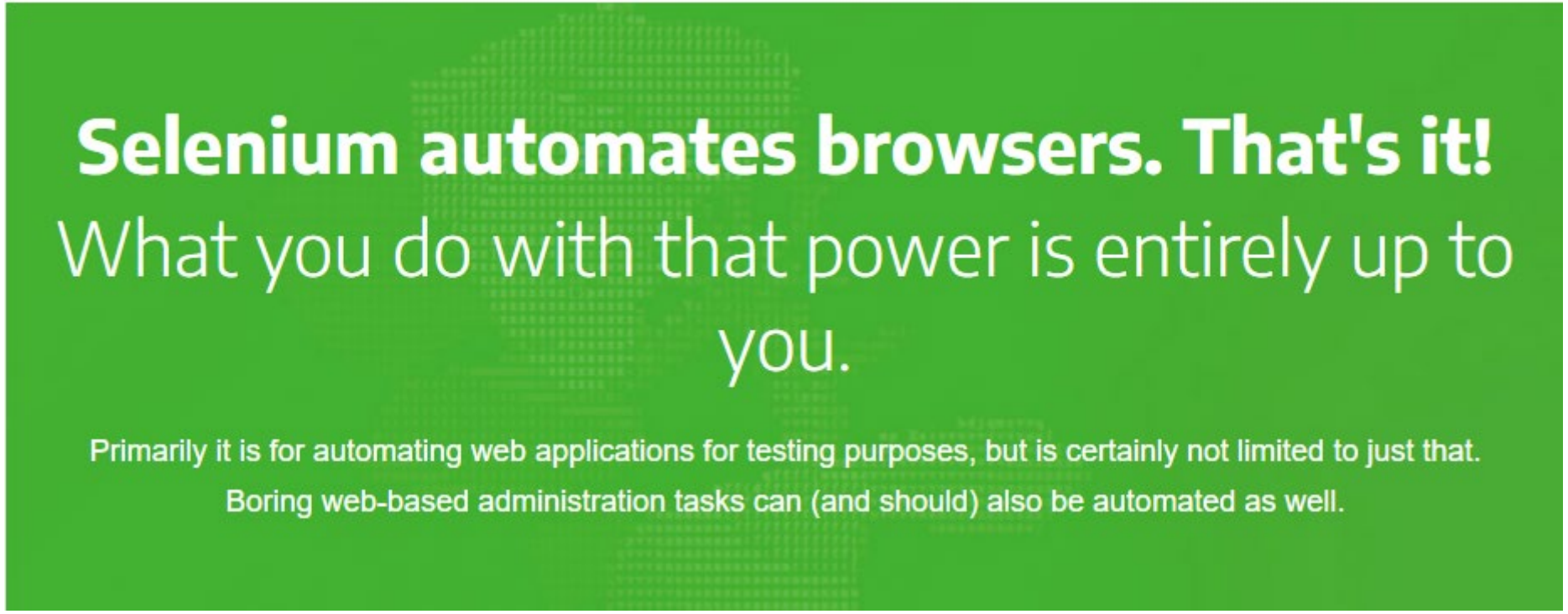
Selenium + JUnit 5: The Perfect Match

Boni García

 boni.garcia@uc3m.es  http://bonigarcia.github.io/

 @boni_gg  https://github.com/bonigarcia

1. Introduction



https://www.selenium.dev/

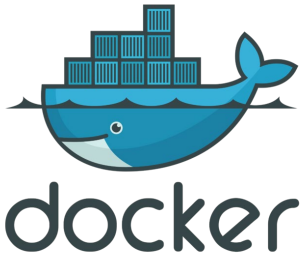
1. Introduction

- When it is used for testing, the calls to the Selenium API is typically placed in a test case (using a testing framework), e.g.:

1. Introduction

- This talk is focused on **Java** as language binding for Selenium and **JUnit 5** as testing framework

+



Selenium-Jupiter

https://bonigarcia.github.io/selenium-jupiter/

1. Introduction

- Source code: https://github.com/bonigarcia/selenium-jupiter
- Documentation: https://bonigarcia.github.io/selenium-jupiter
- Examples: https://github.com/bonigarcia/selenium-jupiter-examples

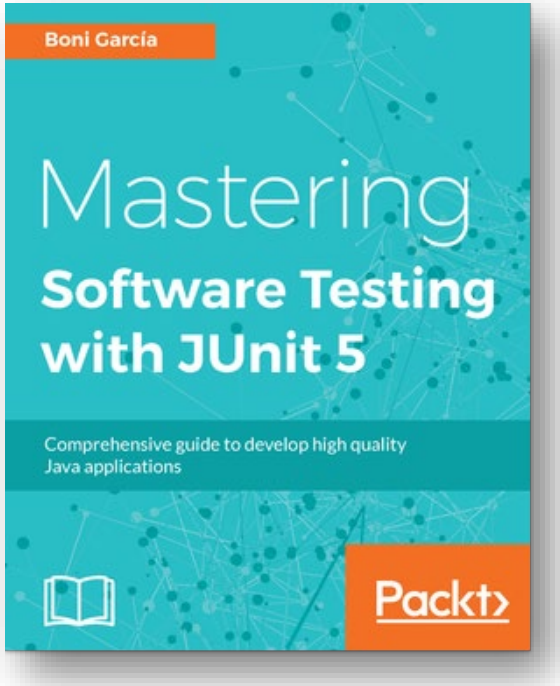
Requirements to run these examples:

- Java
- An IDE or Maven/Gradle
- Docker Engine (only required for running Docker examples)
- Linux (only required for running Android in Docker)

Table of Contents

2. JUnit 5 - Introduction

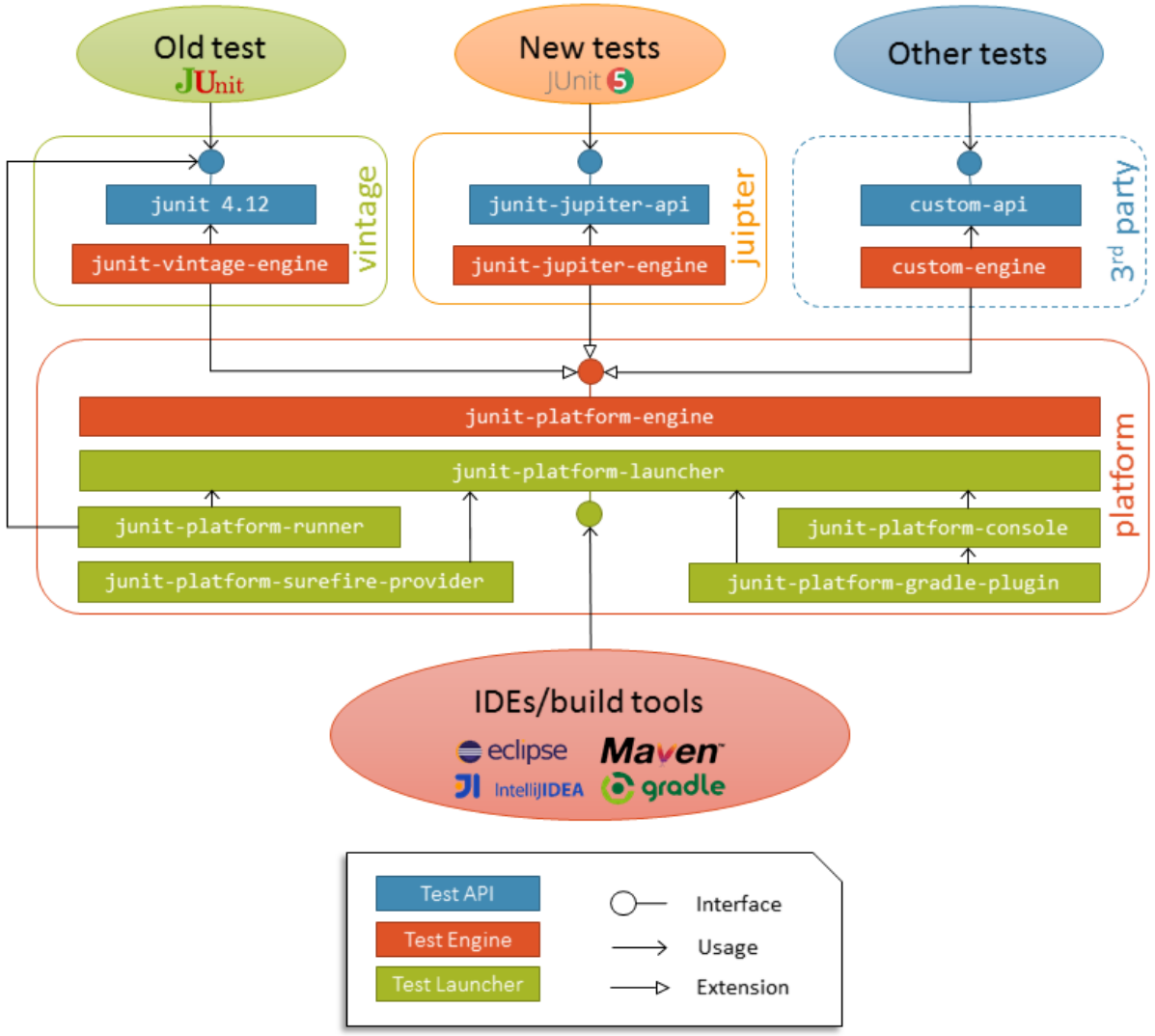
- **JUnit** is the most popular testing framework for Java and can be used to implement different types of tests (unit, integration, end-to-end, …)
- **JUnit 5** (first GA released on September 2017) provides a brand-new programming an extension model called **Jupiter**



JUnit 5

https://junit.org/junit5/docs/current/user-guide/

2. JUnit 5 - Architecture



JUnit 5

2. JUnit 5 - Support

- JUnit 5 test can be executed in different ways:

1. Using a **build tools**:

2. Using an **IDE**:

3. Using the **console launcher** (standalone JAR provided by the JUnit 5 team):

```
java -jar junit-platform-console-standalone-version.jar <Options>
```

2. JUnit 5 - Setup

- To execute JUnit 5 with **Maven** we need to configure `pom.xml`:

```xml
<properties>
    <junit5.version>5.7.0</junit5.version>
    <maven-surefire-plugin.version>2.22.2</maven-surefire-plugin.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>${junit5.version}</version>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>${maven-surefire-plugin.version}</version>
        </plugin>
    </plugins>
</build>
```

To be precise, we need the API in compile time for tests and the engine in execution time

```xml
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>${junit5.version}</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>${junit5.version}</version>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

2. JUnit 5 - Setup

- To execute JUnit 5 with **Gradle** we need to configure `build.gradle`:

```
repositories {
    mavenCentral()
}

ext {
    junit5  = '5.7.0'
}

apply plugin: 'java'
apply plugin: 'eclipse'
apply plugin: 'idea'

test {
    useJUnitPlatform()

    testLogging {
        events "passed", "skipped", "failed"
    }
}

compileTestJava {
    sourceCompatibility = 1.8
    targetCompatibility = 1.8
    options.compilerArgs += '-parameters'
}
```

```
dependencies {
    testImplementation("org.junit.jupiter:junit-jupiter-engine:${junit5}")
}
```

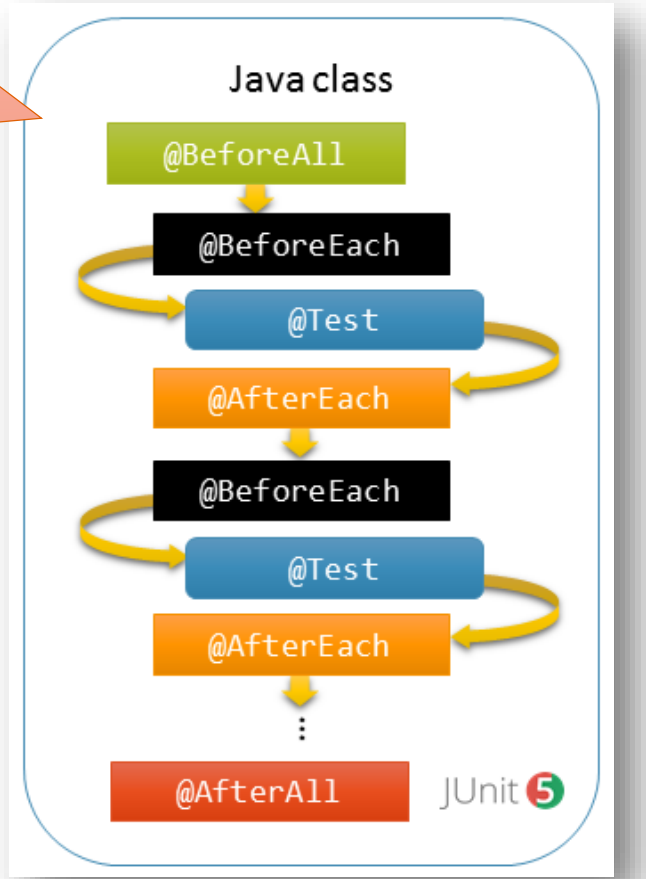
To be precise, we need the API in compile time for tests and the engine in execution time

```
dependencies {
    testImplementation("org.junit.jupiter:junit-jupiter-api:${junit5}")
    testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:${junit5}")
}
```

2. JUnit 5 - Basic tests

- Basic tests in JUnit 5 are similar to JUnit 4:

The names of the annotations for test lifecycle have changed in JUnit 5

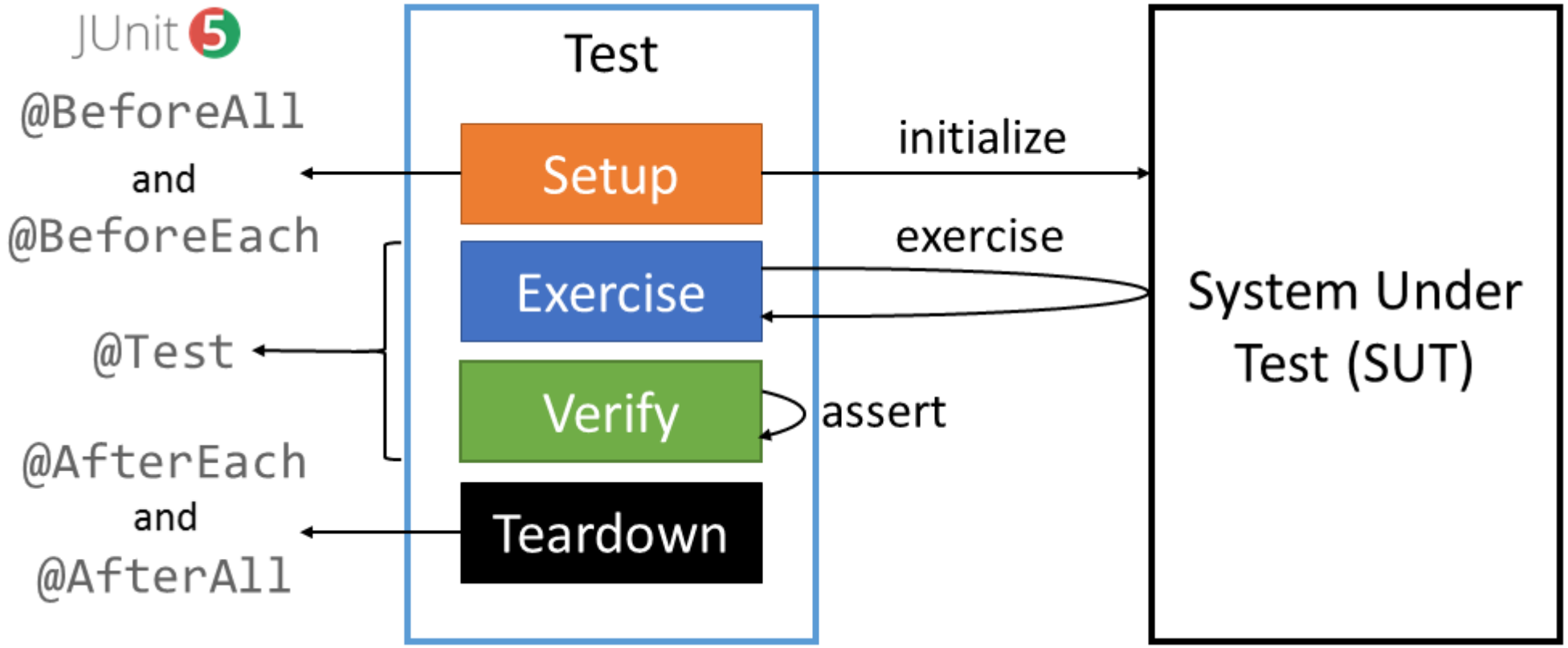



```
class BasicJUnit5Test {

    @BeforeAll
    static void setupAll() {
        // setup all tests
    }

    @BeforeEach
    void setup() {
        // setup each test
    }

    @Test
    void test() {
        // exercise and verify SUT
    }

    @AfterEach
    void teardown() {
        // teardown each test
    }

    @AfterAll
    static void teardownAll() {
        // teardown all tests
    }
}
```

Methods are no required to be `public` anymore

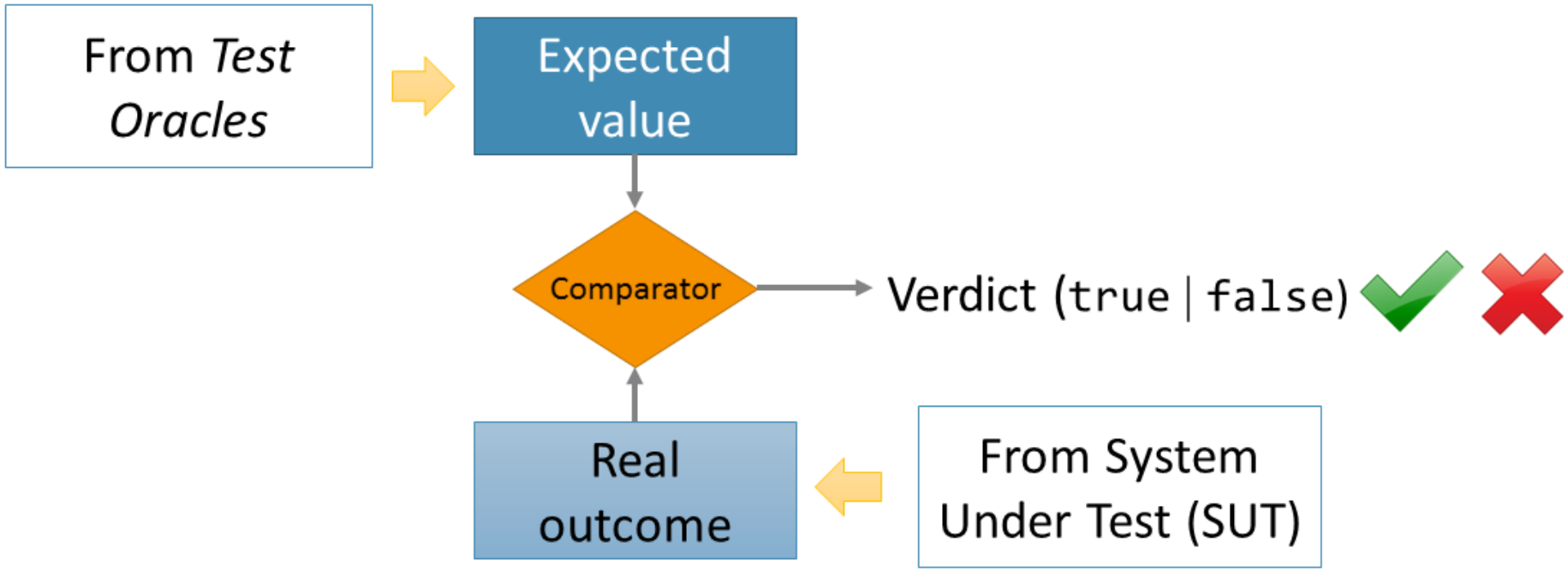
2. JUnit 5 - Basic tests

- We can represent the basic test lifecycle as follows:



2. JUnit 5 - Assertions

- An **assertion** is a predicate (boolean function) that should be evaluated to true to continue with the execution of the program or test



2. JUnit 5 - Assertions

- JUnit 5 provides a rich variety of assertions (static methods of the class `Assertions`):
  - `assertTrue`, `assertFalse`, `assertEquals`, `assertSame`, …
- In addition, there is a number of Java libraries providing fluent APIs for assertions, such as:
  - Hamcrest: http://hamcrest.org/
  - AssertJ: https://assertj.github.io/doc/
  - Truth: https://truth.dev/

In the examples repository, Truth is used

```
<dependency>
    <groupId>com.google.truth</groupId>
    <artifactId>truth</artifactId>
    <version>${truth.version}</version>
    <scope>test</scope>
</dependency>
```

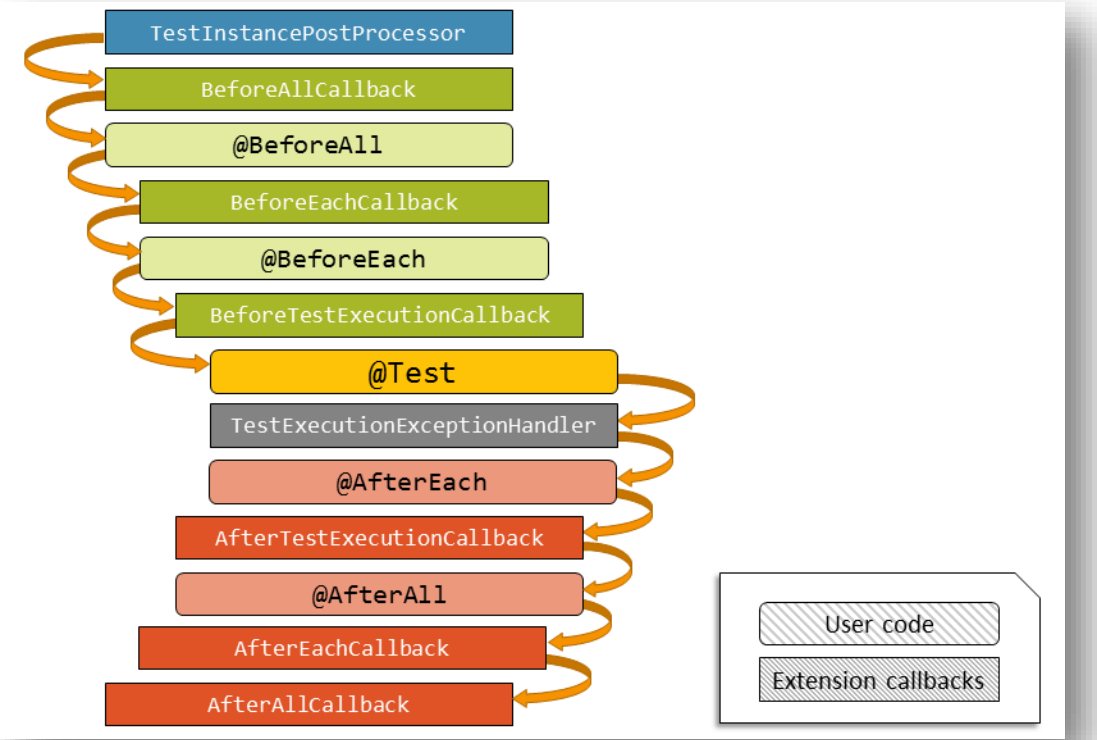
2. JUnit 5 - Other features

- JUnit 5 has many features, such as:
  - Parameterized tests
  - Parallel execution
  - Ordered tests
  - Display names
  - Assumptions
  - Conditional test execution
  - Tagging and filtering
  - Nested tests
  - Repeated tests
  - Dynamic tests
  - Timeouts
  - …

https://junit.org/junit5/docs/current/user-guide/

2. JUnit 5 - Extension model

- The **extension model** of Jupiter allows to add custom features to the programming model:
  - Dependency injection in test methods and constructors
  - Custom logic in the test lifecycle
  - Test templates

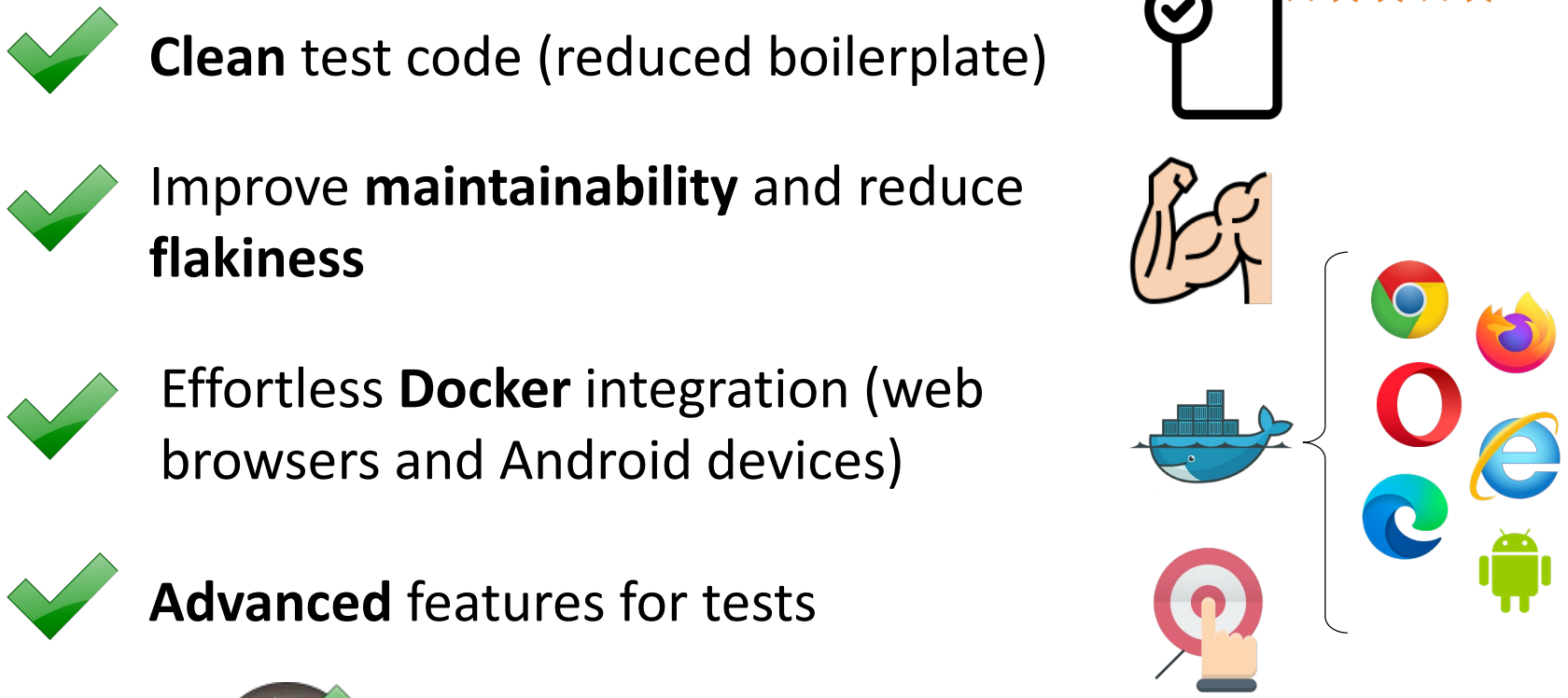


Very convenient for Selenium!

Table of Contents

3. Selenium-Jupiter - Motivation

- **Selenium-Jupiter** is a JUnit 5 extension aimed to ease the use of Selenium from Java tests



Clean test code (reduced boilerplate)

Improve **maintainability** and reduce **flakiness**

Effortless **Docker** integration (web browsers and Android devices)

Advanced features for tests

https://bonigarcia.github.io/selenium-jupiter/

3. Selenium-Jupiter - Setup

- **Selenium-Jupiter** can be included in a Java project as follows:

```
<dependency>
    <groupId>io.github.bonigarcia</groupId>
    <artifactId>selenium-jupiter</artifactId>
    <version>3.4.0</version>
    <scope>test</scope>
</dependency>
```

Maven

Using the latest version is always recommended

```
dependencies {
    testImplementation("io.github.bonigarcia:selenium-jupiter:3.4.0")
}
```

3. Selenium-Jupiter - Local browsers

- JUnit 4 and Selenium  JUnit 5 and Selenium-Jupiter:

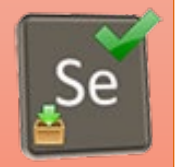
3. Selenium-Jupiter - Local browsers

- Selenium-Jupiter uses JUnit 5's **dependency injection**:

Valid types: ChromeDriver, FirefoxDriver, OperaDriver, SafariDriver, EdgeDriver, InternetExplorerDriver, HtmlUnitDriver, PhantomJSDriver, AppiumDriver, SelenideDriver

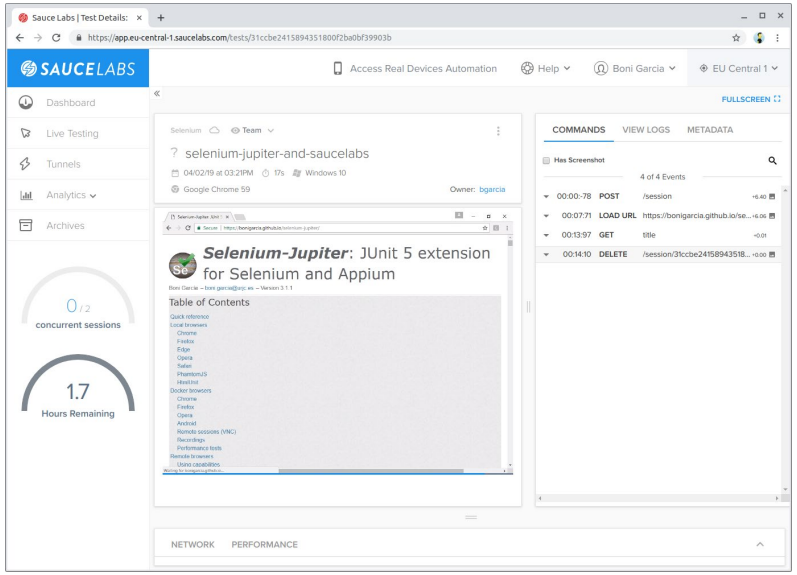
```
@ExtendWith(SeleniumJupiter.class)
class SeleniumJupiterTest {

    @Test
    void test(ChromeDriver chromeDriver) {
        // Use Chrome in this test
    }

}
```

Internally, Selenium-Jupiter uses WebDriverManager to resolve properly the required browser drivers (chromedriver in this example)

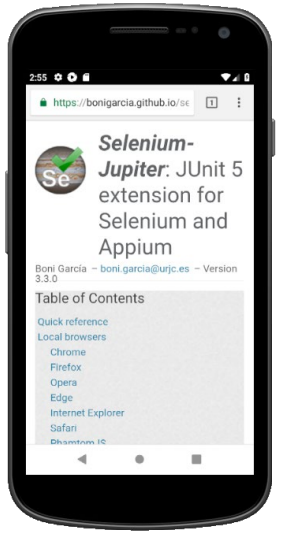


3. Selenium-Jupiter - Remote browsers

- Selenium-Jupiter provides the annotations `@DriverUrl` and `@DriverCapabilities` to control remote browsers and mobiles, e.g.:



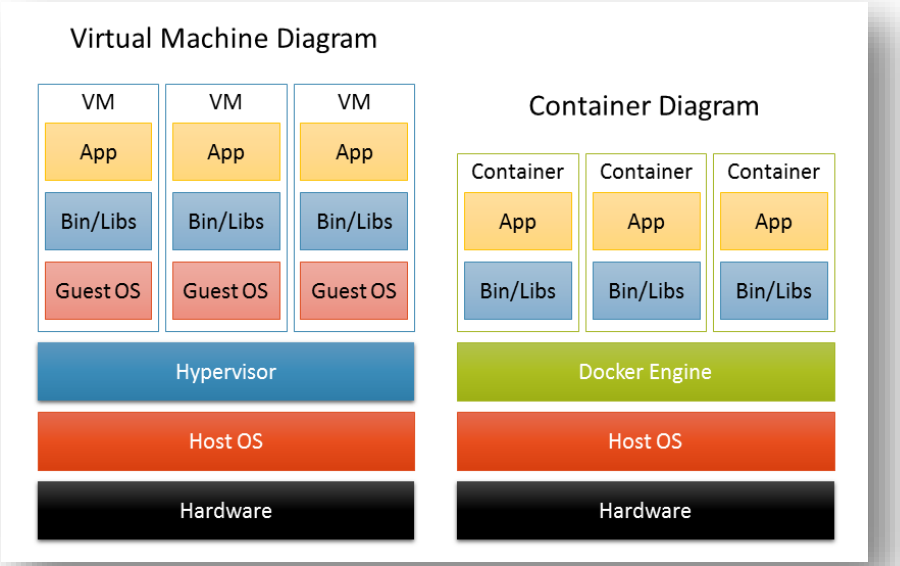
SAUCELABS
https://saucelabs.com/



appium
http://appium.io/

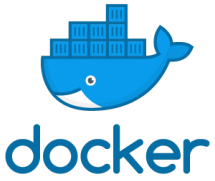
3. Selenium-Jupiter - Docker browsers

- **Docker** is a software technology which allows to pack and run any application as a lightweight and portable container
- The Docker platform has two main components: the Docker Engine, to create and execute containers; and the Docker Hub (https://hub.docker.com/), a cloud service for distributing containers

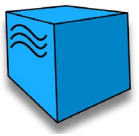


https://www.docker.com/

3. Selenium-Jupiter - Docker browsers



- Selenium-Jupiter provides seamless integration with **Docker** using the annotation `@DockerBrowser`:
  - Chrome, Firefox, Opera, and Edge:
    - Docker images for stable versions are maintained by Aerokube
    - Beta and unstable (Chrome and Firefox) are maintained by ElasTest
  - Internet Explorer:
    - Due to Windows license, this images is not hosted in Docker Hub
    - It can be built following a tutorial provided by Aerokube
  - Android devices:
    - Docker images for Android (docker-android project) by Budi Utomo



3. Selenium-Jupiter - Docker browsers

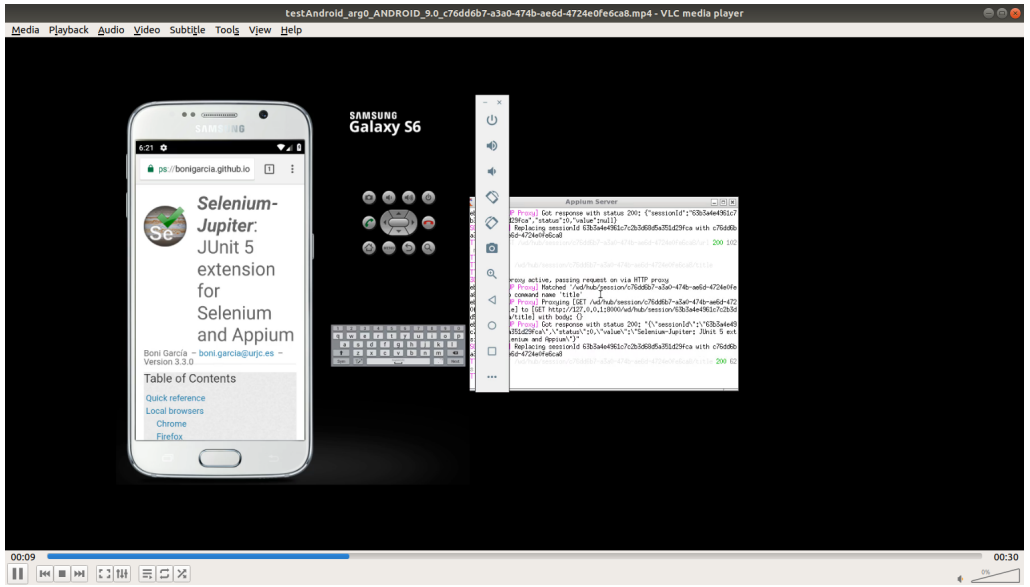
```
@ExtendWith(SeleniumJupiter.class)
class DockerBasicTest {

    @Test
    void testFirefoxBeta(
            @DockerBrowser(type = FIREFOX, version = "beta") RemoteWebDriver driver) {
        driver.get("https://bonigarcia.github.io/selenium-jupiter/");
        assertThat(driver.getTitle(),
                containsString("JUnit 5 extension for Selenium"));
    }

}
```

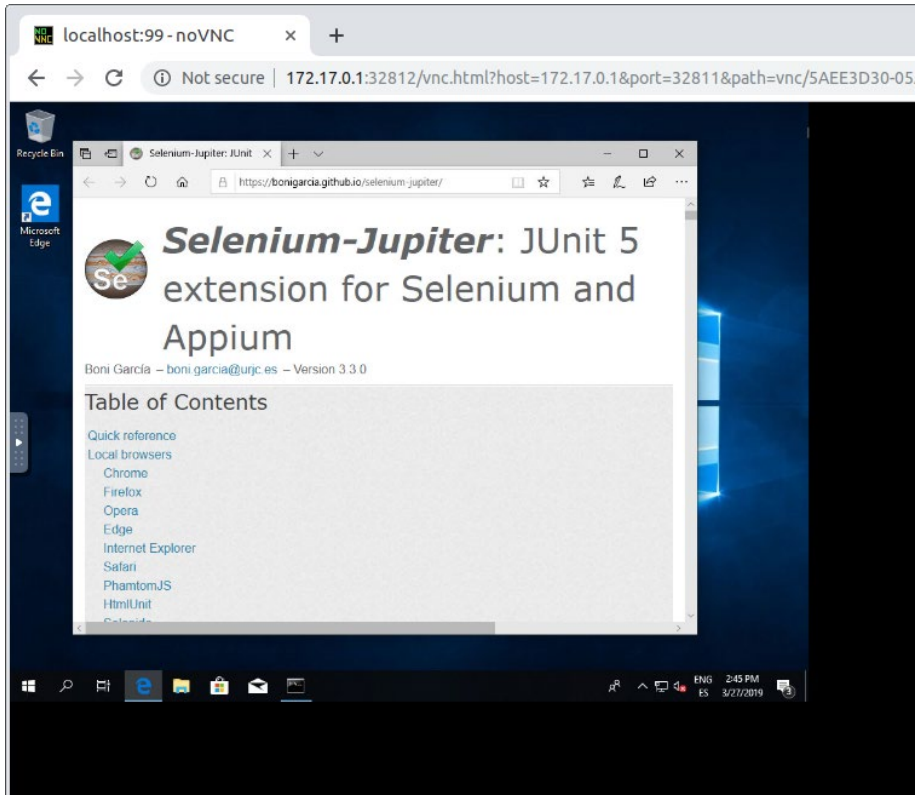
Supported browser types are: *CHROME*, *FIREFOX*, *OPERA*, *EDGE* , *IEXPLORER* and *ANDROID*

If *version* is not specified, the latest container version in Docker Hub is pulled. This parameter allows fixed versions and also the special values: *latest*, *latest-**, *beta*, and *unstable*

3. Selenium-Jupiter - Docker browsers

- The use of Docker enables a rich number of features:
  - Remote session access with **VNC**
  - Session **recordings**
  - **Performance** tests





3. Selenium-Jupiter - Docker browsers

- The possible **Android** setup options are the following:

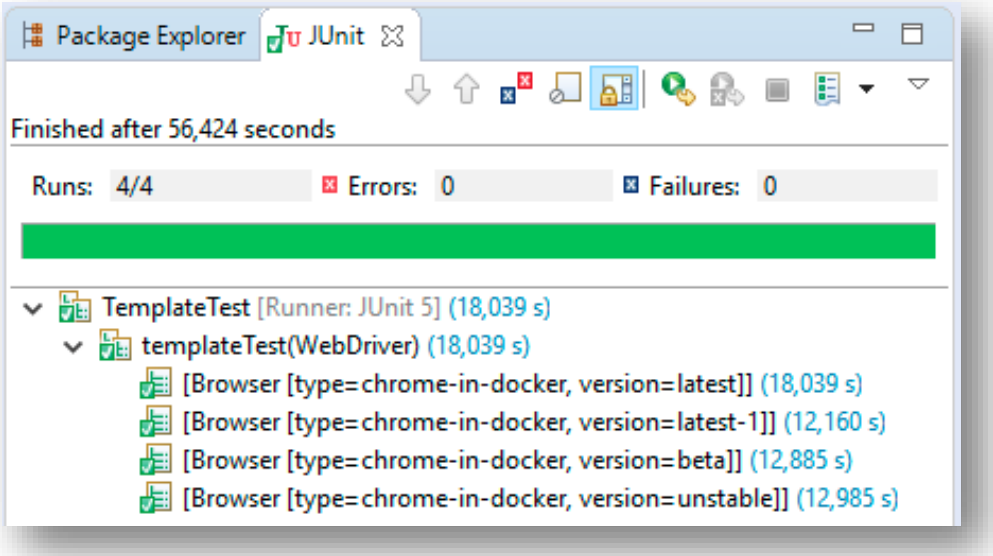
Type	Device name
Phone	Samsung Galaxy S6
Phone	Nexus 4
Phone	Nexus 5
Phone	Nexus One
Phone	Nexus S
Tablet	Nexus 7

Android version	API level	Browser name
5.0.1	21	browser
5.1.1	22	browser
6.0	23	chrome
7.0	24	chrome
7.1.1	25	chrome
8.0	26	chrome
8.1	27	chrome
9.0	28	chrome

3. Selenium-Jupiter - Test templates

- Selenium-Jupiter use the JUnit 5's support for **test templates**:

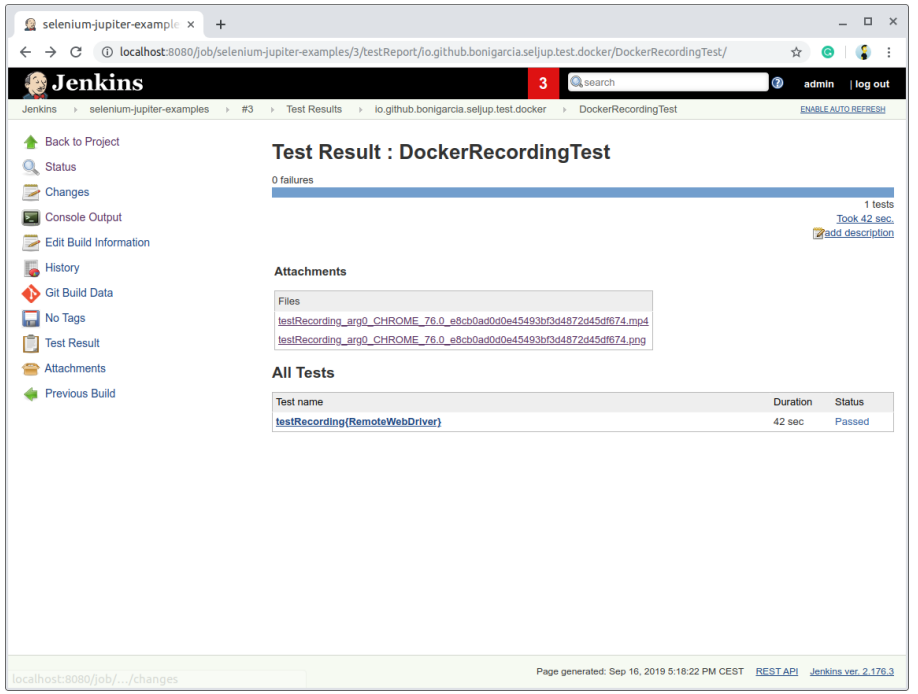
```
@ExtendWith(SeleniumJupiter.class)
public class TemplateTest {

    @TestTemplate
    void templateTest(WebDriver driver) {
       // test
    }

}
```



3. Selenium-Jupiter - Integration with Jenkins

- Seamless integration with Jenkins through the JUnit attachment plugin
- It allows to attach output files in tests (e.g. PNG screenshots and MP4 recordings) in the Jenkins GUI
- For example:

```
$ mvn clean test -Dtest=DockerRecordingTest \
    -Dsel.jup.recording=true \
    -Dsel.jup.screenshot.at.the.end.of.tests=true \
    -Dsel.jup.screenshot.format=png \
    -Dsel.jup.output.folder=surefire-reports
```



Jenkins

3. Selenium-Jupiter - Beyond Java

- Selenium-Jupiter can be also used:

1. As **CLI** (Command Line Interface) tool:

Selenium-Jupiter allows to control Docker browsers through VNC (manual testing)

```
$ java -jar selenium-jupiter-3.4.0-fat.jar chrome unstable
[INFO] Using Selenium-Jupiter to execute chrome unstable in Docker
...
```

2. As a **server** (using a REST-like API):

Selenium-Jupiter becomes into a Selenium Server (Hub)

```
$ java -jar selenium-jupiter-3.4.0-fat.jar server
[INFO] Selenium-Jupiter server listening on http://localhost:4042/wd/hub
```

Table of Contents

4. Final remarks and future work

- Selenium-Jupiter has another features such as:
  - Generic driver (configurable type of browser)
  - Mapping volumes in Docker containers
  - Access to Docker client to manage custom containers
- Selenium-Jupiter is in constant development. Its roadmap includes:
  - Improve test template support (e.g. specifying options)
  - Improve scalability for performance tests (e.g. using a Docker cluster)
  - Enhance diagnostic capabilities (gathering mechanism for the browser console)

Selenium + JUnit 5: The Perfect Match

Thank you very much!

Boni García

 boni.garcia@uc3m.es  http://bonigarcia.github.io/

 @boni_gg  https://github.com/bonigarcia