

# Developing Selenium tests with JUnit 5



Selenium Conf  
VIRTUAL 2020

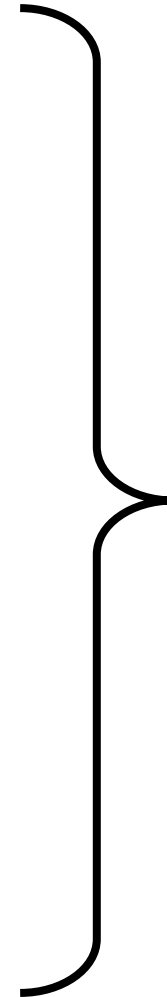
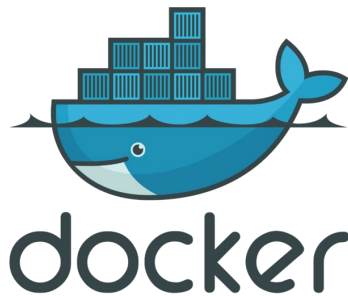
Boni García

✉ [boni.garcia@uc3m.es](mailto:boni.garcia@uc3m.es) 🏠 <http://bonigarcia.github.io/>

🐦 [@boni\\_gg](https://twitter.com/boni_gg) 🐙 <https://github.com/bonigarcia>

# 1. Introduction

JUnit 



<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>

Fork me on GitHub

Requirements to run these examples:

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



# Table of Contents

## 1. Introduction

## 2. JUnit 5

- Introduction
- Architecture
- Support
- Setup
- Basic tests
- Assertions
- Parameterized tests
- Features
- Extension model

## 3. Selenium-Jupiter

## 4. Final remarks and future work

## 2. JUnit 5 - Introduction

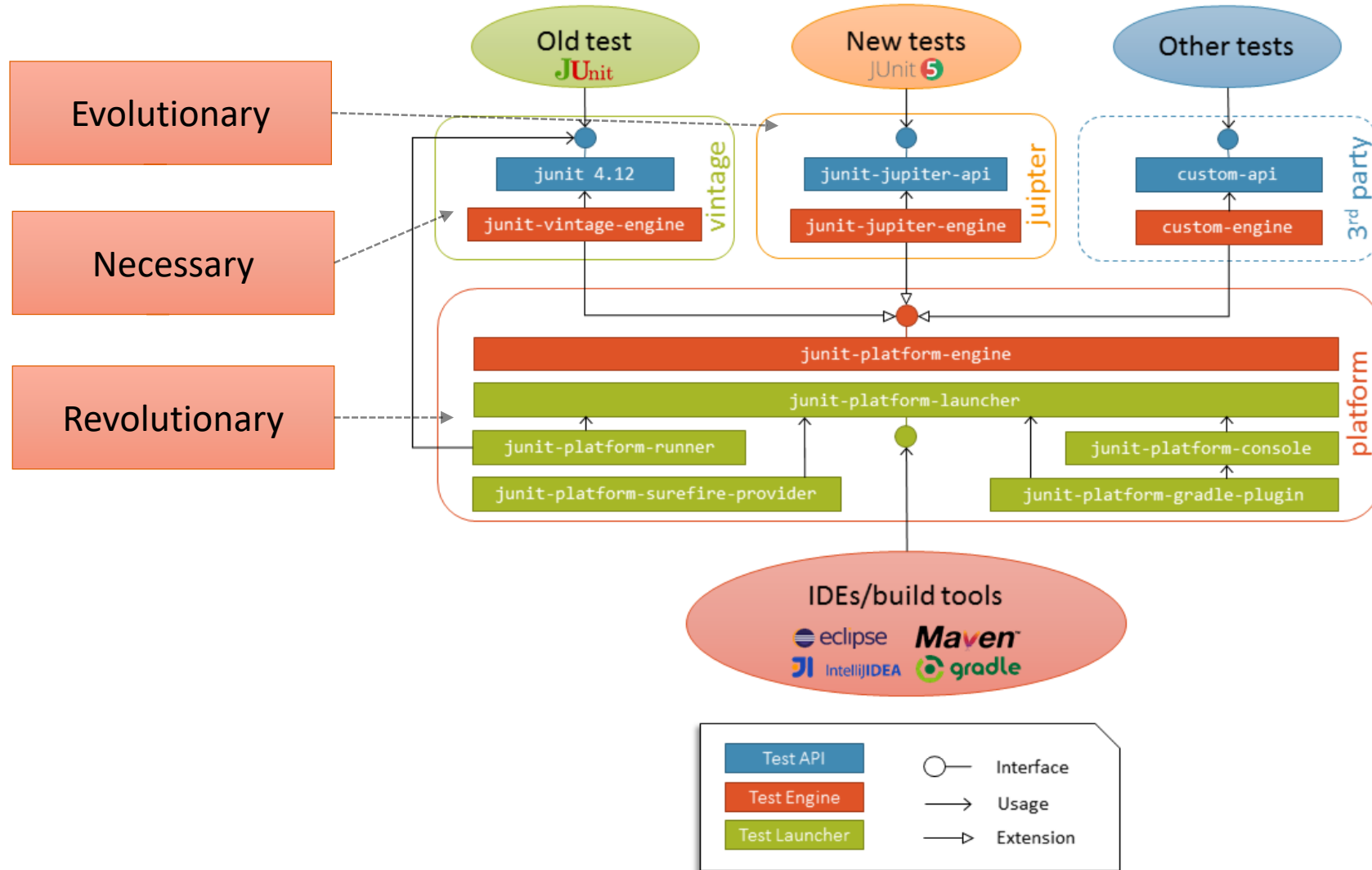
- **JUnit** is the most popular testing framework for Java and can be used to implement different types of tests (unit, integration, end-to-end, ...)
- **JUnit 5** (first GA released on September 2017) provides a brand-new programming an extension model called **Jupiter**



<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:



```
<properties>
  <junit5.version>5.6.2</junit5.version>
  <maven-surefire-plugin.version>2.22.0</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

```
<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** (4.8+) we need to configure `build.gradle`:

```
repositories {  
    mavenCentral()  
}  
  
ext {  
    junit5 = '5.6.2'  
}  
  
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 {  
    testCompile("org.junit.jupiter:junit-jupiter-engine:${junit5}")  
}
```



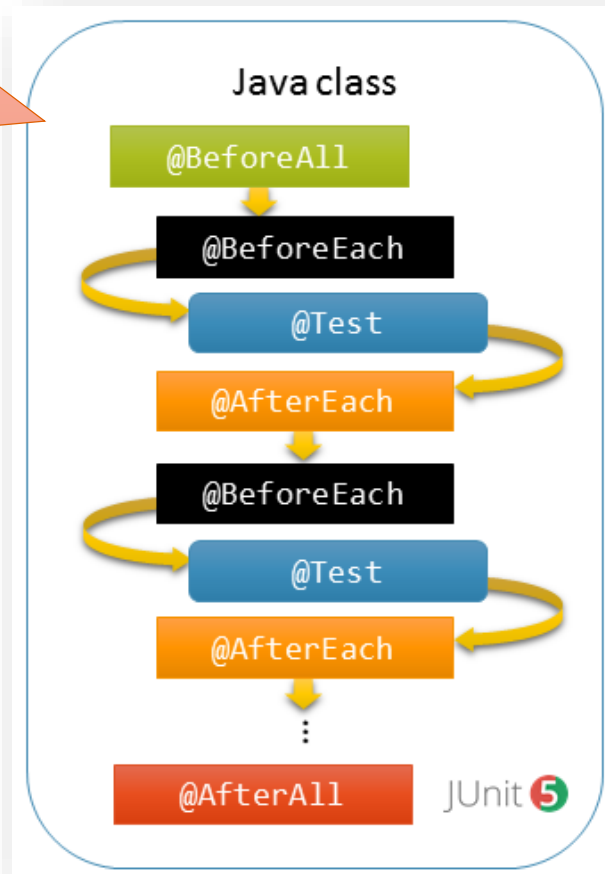
To be precise, we need the API in compile time for tests and the engine in execution time

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



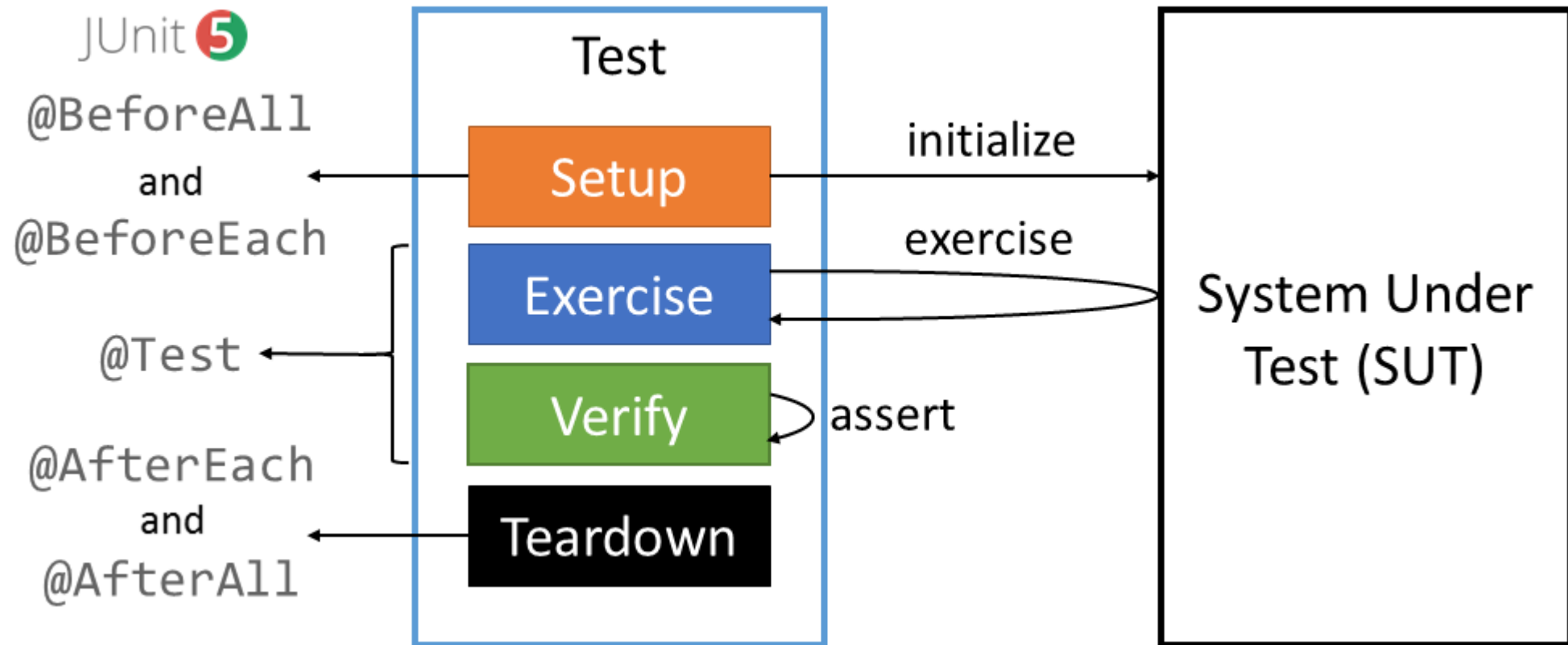
```
class BasicJUnit5Test {  
    @BeforeAll  
    static void setupAll() {  
        // setup all tests  
    }  
  
    @BeforeEach  
    void setup() {  
        // setup each test  
    }  
  
    @Test  
    void test() {  
        // exercise and verify SUT  
    }  
  
    @AfterEach  
    void teardown() {  
        // teardown each test  
    }  
  
    @AfterAll  
    static void teardownAll() {  
        // teardown all tests  
    }  
}
```

JUnit 5

Methods are no required to be `public` anymore

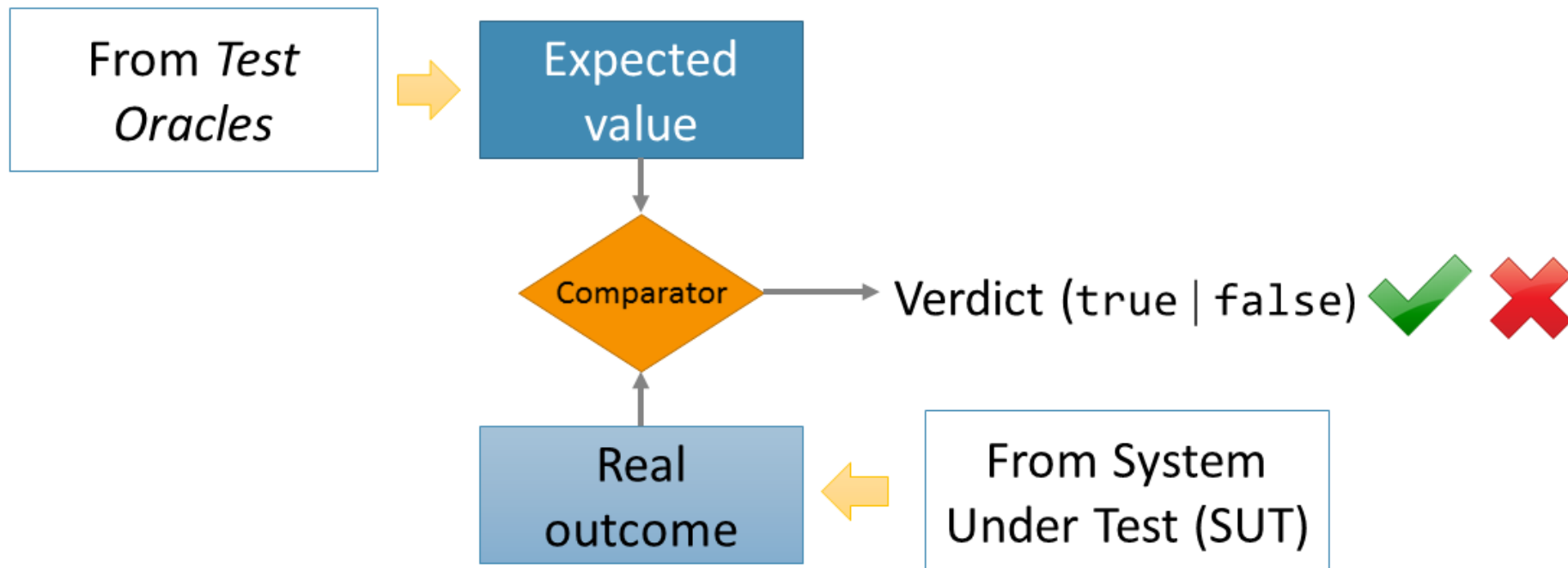
## 2. JUnit 5 - Basic tests

- 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

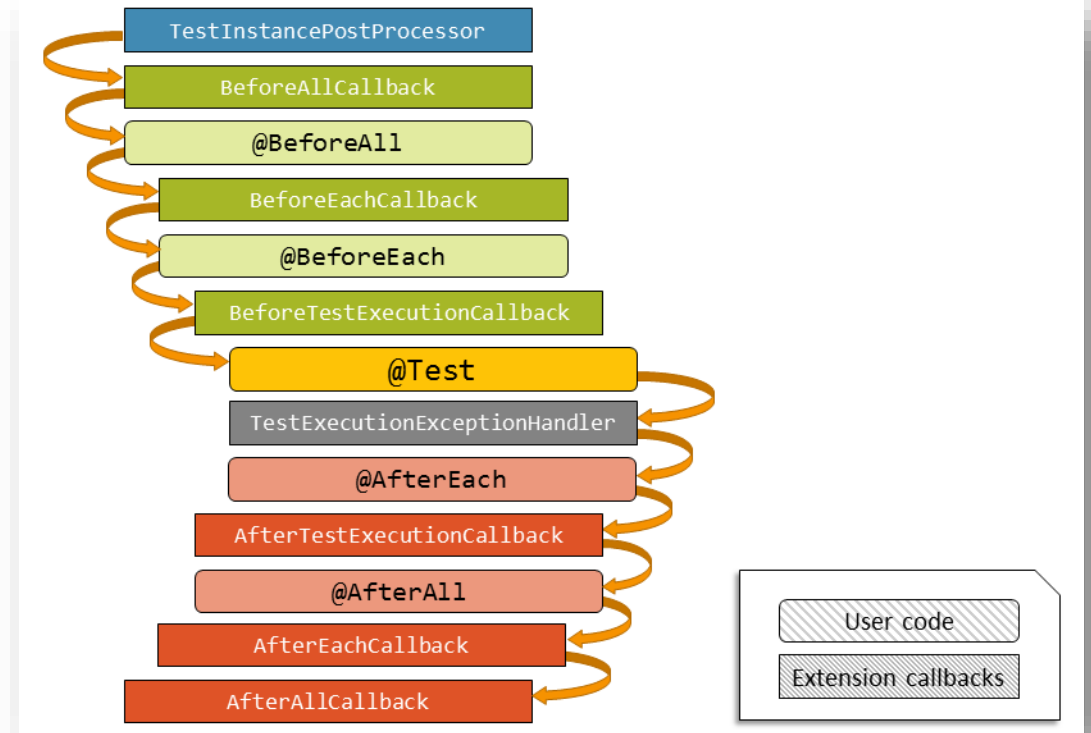
- JUnit 5 has many features, such as:
  - Parameterized tests
  - Parallel execution
  - Ordered tests
  - Display names
  - Assumptions
  - Conditional test execution
  - Tagging and filtering
  - Nested tests
  - Repeated tests
  - Dynamic tests
  - Timeouts
  - ...



<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

1. Introduction
2. JUnit 5
3. Selenium-Jupiter
  - Motivation
  - Setup
  - Local browsers
  - Remote browsers
  - Docker browsers
  - Test templates
  - Integration with Jenkins
  - Beyond Java
4. Final remarks and future work

# 3. Selenium-Jupiter - Motivation

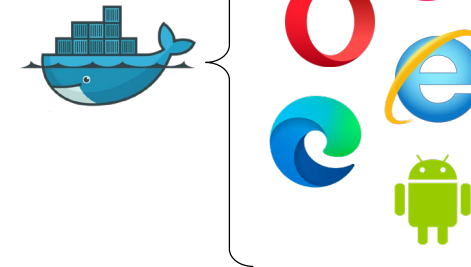
- **Selenium-Jupiter** is a JUnit 5 extension aimed to ease the use of Selenium from Java tests



**Clean** test code (reduced boilerplate)



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>3.3.5</version>
  <scope>test</scope>
</dependency>
```



Using the latest version is always recommended!

```
dependencies {
  testCompile("io.github.bonigarcia:selenium-jupiter:3.3.5")
}
```





### 3. Selenium-Jupiter - Local browsers

• JUnit 4 and Selenium

VS

JUnit 5 and Selenium-Jupiter:

JUnit



JUnit 5



# 3. Selenium-Jupiter - Local browsers

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

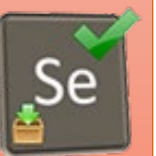
Valid types: ChromeDriver, FirefoxDriver, OperaDriver, SafariDriver, EdgeDriver, InternetExplorerDriver, HtmlUnitDriver, PhantomJSdriver, AppiumDriver, SelenideDriver

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

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

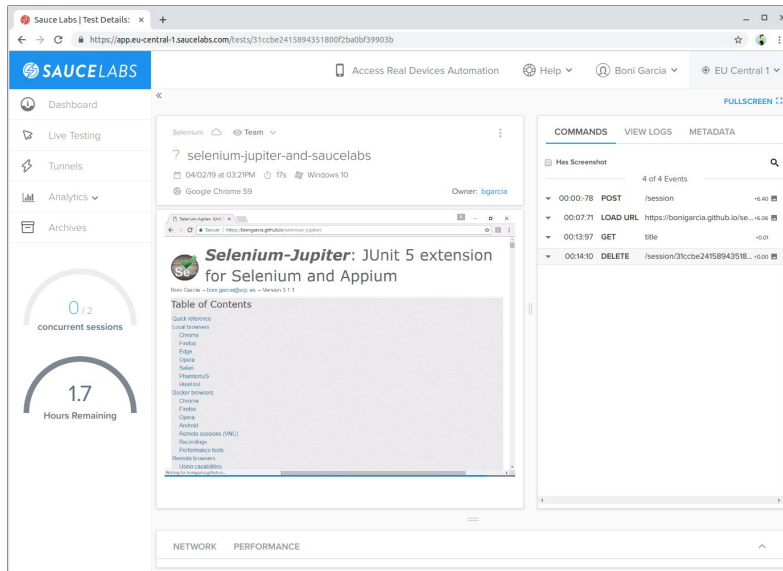


Internally, Selenium-Jupiter uses [WebDriverManager](#) to resolve properly the required browser drivers (chromedriver in this example)

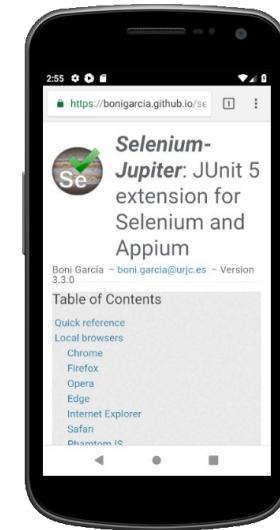


# 3. Selenium-Jupiter - Remote browsers

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



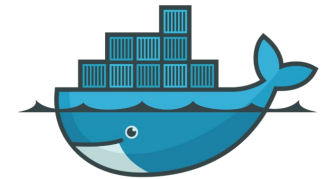
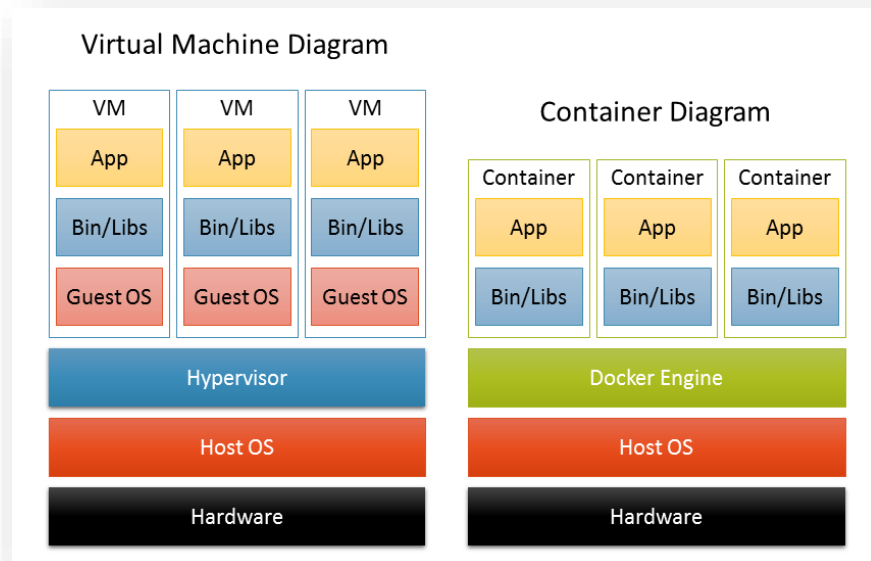
 **SAUCE LABS**  
<https://saucelabs.com/>



 **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, and Opera: 
    - Docker images for stable versions are maintained by Aerokube
    - Beta and unstable (Chrome and Firefox) are maintained by ElastiTest
  - Edge and Internet Explorer: 
    - Due to license, these Docker images are 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 {

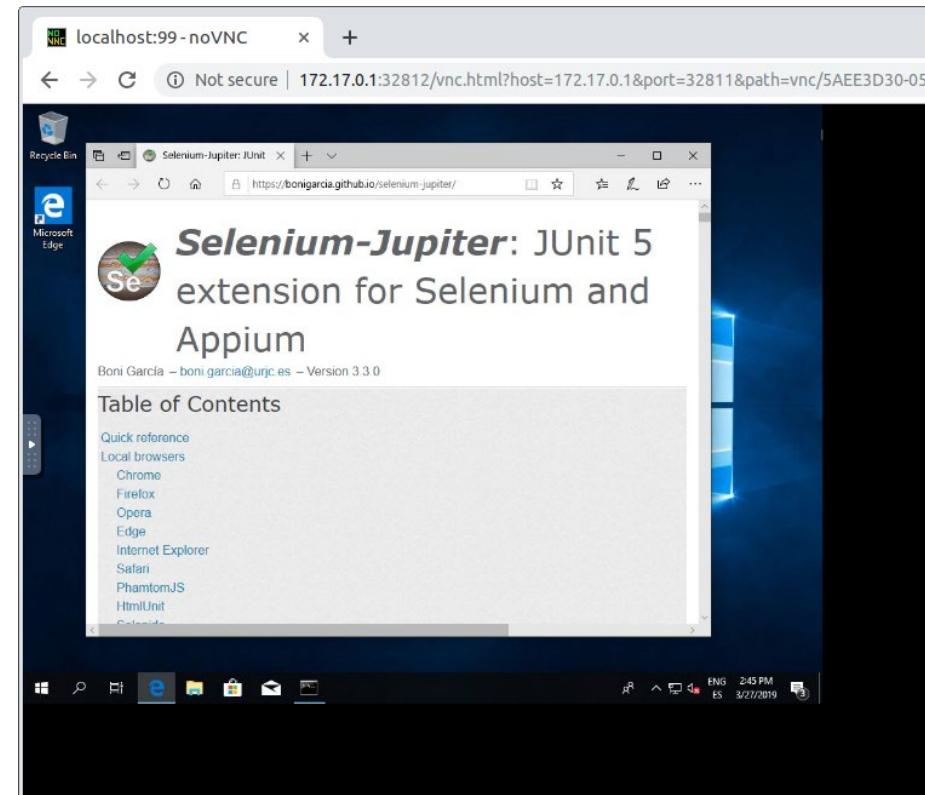
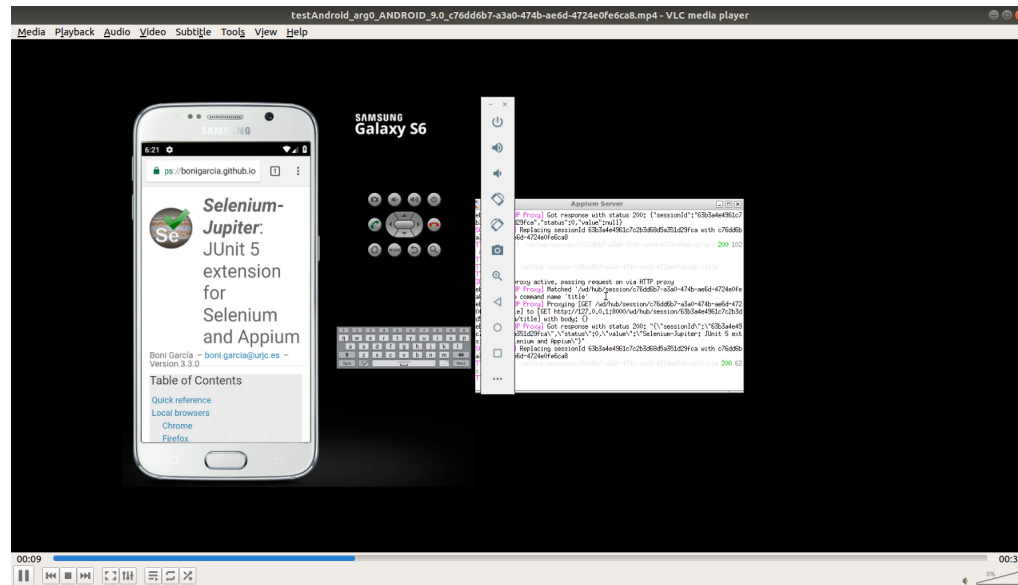
    @Test
    void testFirefoxBeta(
        @DockerBrowser(type = FIREFOX, version = "beta") RemoteWebDriver driver) {
        driver.get("https://bonigarcia.github.io/selenium-jupiter/");
        assertThat(driver.getTitle(),
            containsString("JUnit 5 extension for Selenium"));
    }
}
```

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

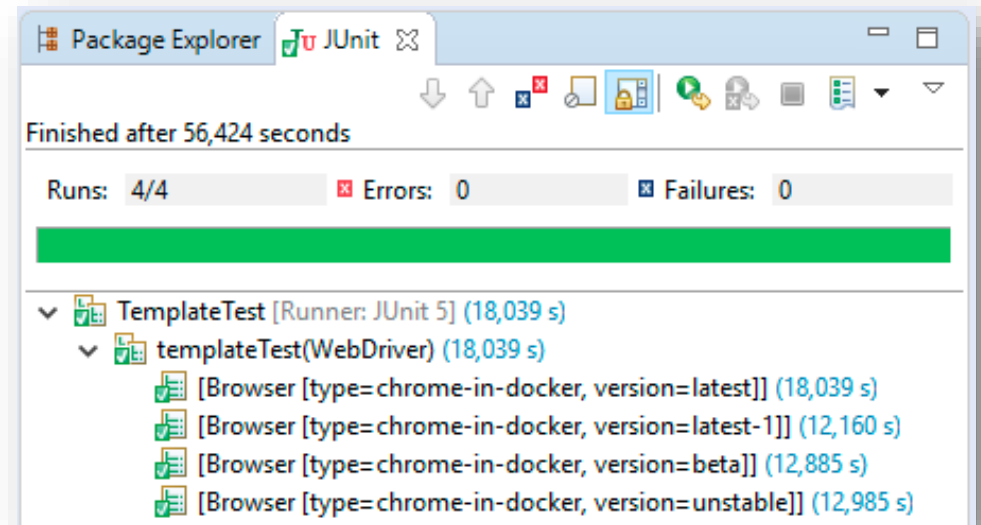


# 3. Selenium-Jupiter - Test templates

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

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

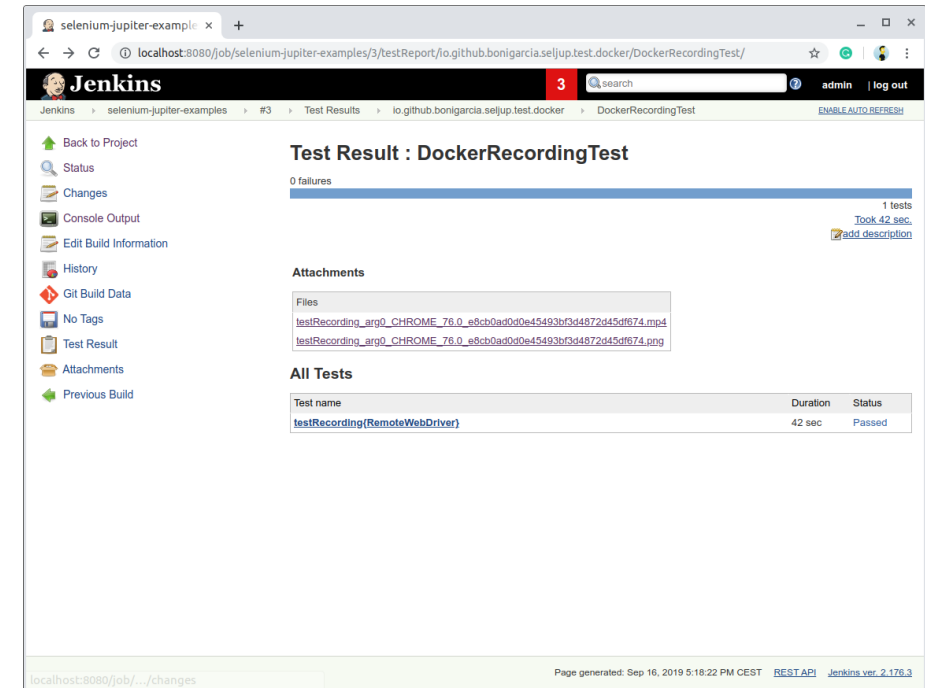
    @TestTemplate
    void templateTest(WebDriver driver) {
        // test
    }
}
```



# 3. Selenium-Jupiter - Integration with Jenkins

- Seamless integration with Jenkins through the [Jenkins 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-3.3.5-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-3.3.5-fat.jar server  
[INFO] Selenium-Jupiter server listening on http://localhost:4042/wd/hub
```

# Table of Contents

1. Introduction
2. JUnit 5
3. Selenium-Jupiter
4. Final remarks and future work

## 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 using Kubernetes
  - Enhance diagnostic capabilities (gathering mechanism for the browser console)



# Developing Selenium tests with JUnit 5

Thank you very much!

Boni García

 [boni.garcia@uc3m.es](mailto:boni.garcia@uc3m.es)  <http://bonigarcia.github.io/>

 [@boni\\_gg](https://twitter.com/boni_gg)  <https://github.com/bonigarcia>