

Selenium-Jupiter: The next level of web testing

Agile & Automation Days 2019
29/10/2019

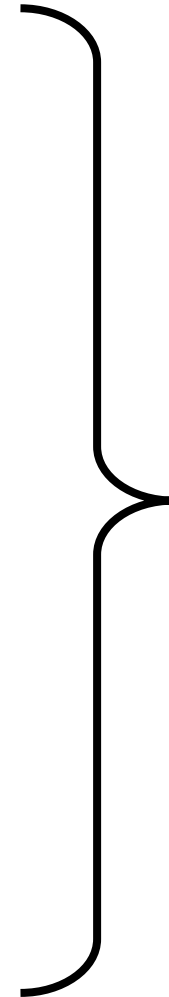
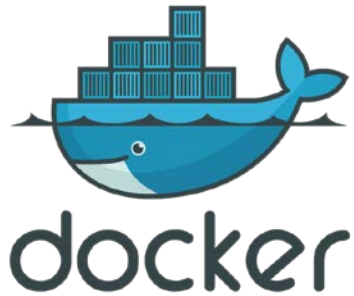
Boni García

 boni.garcia@urjc.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
 - Ordered tests
 - Other features
 - Extension model
3. Selenium
4. Selenium-Jupiter
5. Final remarks and future work

2. JUnit 5 - Introduction

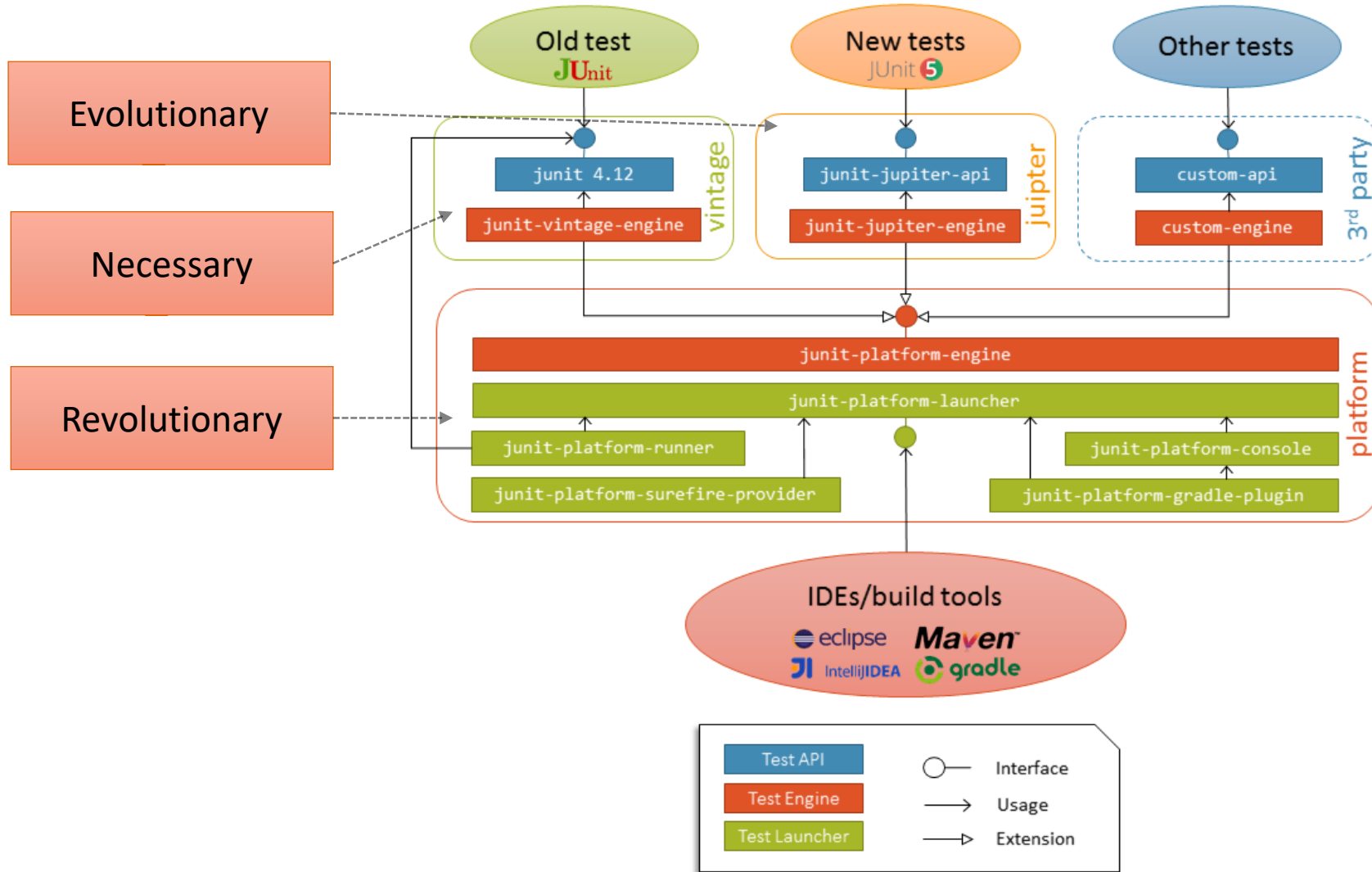
- **JUnit** is the most popular testing framework for Java and can be used to implement different types of tests (unit, integration, end-to-end, ...)
- **JUnit 5** (first GA released on September 2017) provides a brand-new programming an extension model called **Jupiter**



JUnit 

<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>5.5.2</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 = '5.5.2'
}

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}")
}
```



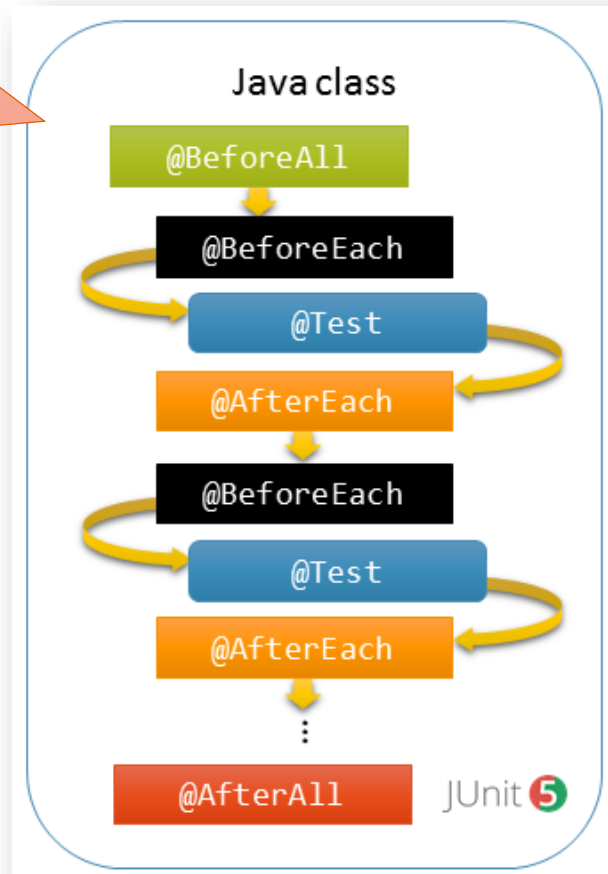
To be precise, we need the API in compile time for tests and the engine in execution time

```
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



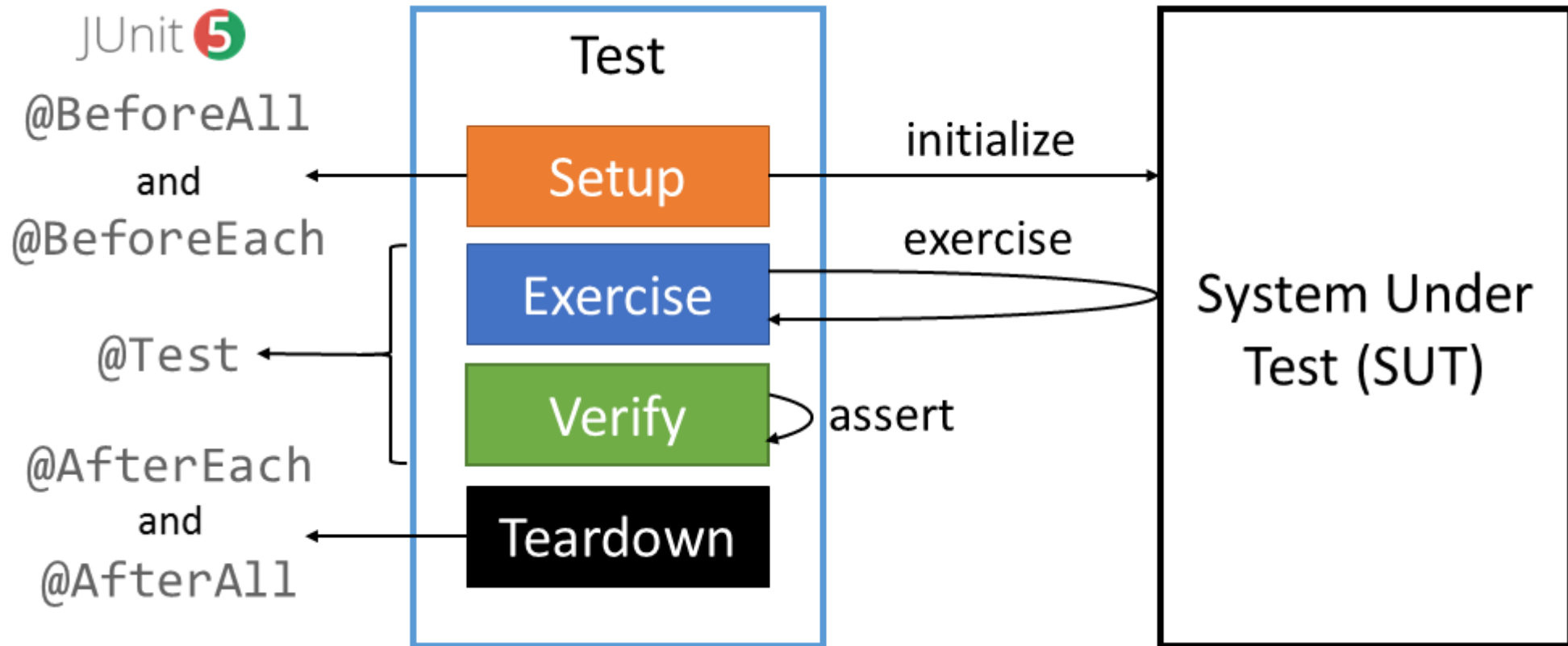
```
class BasicJUnit5Test {  
    @BeforeAll  
    static void setupAll() {  
        // setup all tests  
    }  
  
    @BeforeEach  
    void setup() {  
        // setup each test  
    }  
  
    @Test  
    void test() {  
        // exercise and verify SUT  
    }  
  
    @AfterEach  
    void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

JUnit 5

Methods are no longer required to be `public` anymore

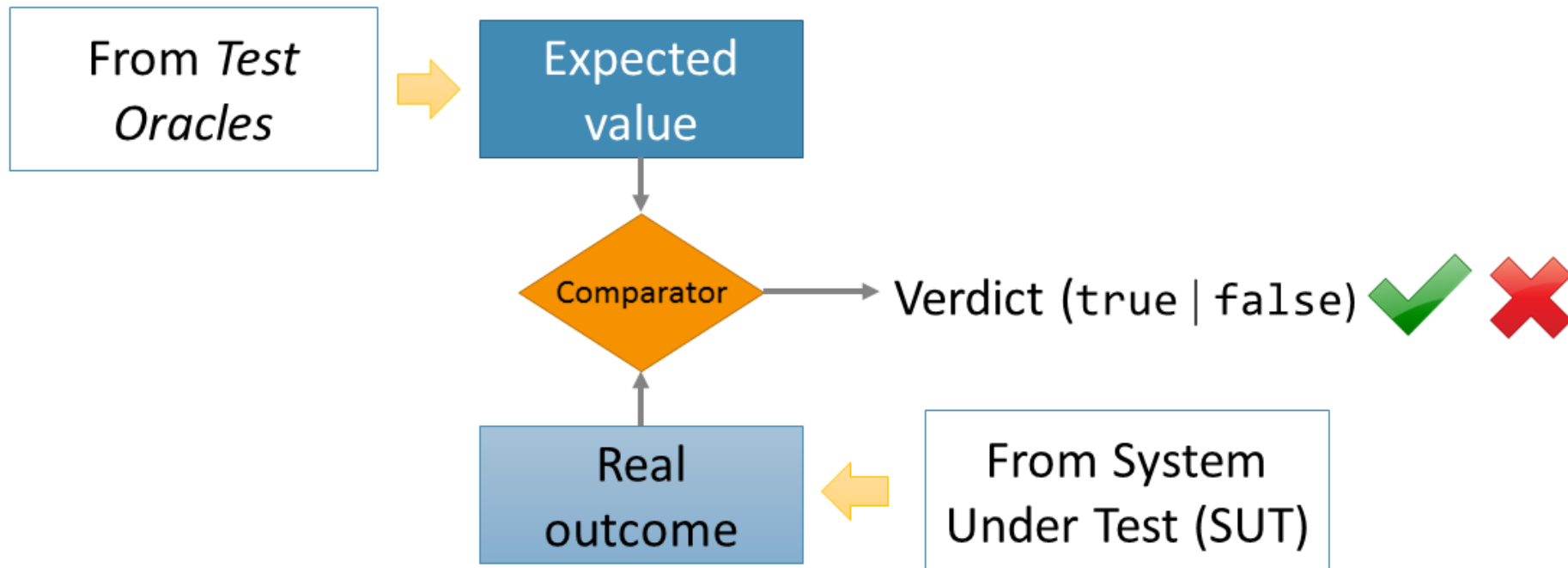
2. JUnit 5 - Basic tests

- 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 - Parameterized tests

- **Parameterized tests** execute the same test logic with different input data
- To implement these kind of tests, we should add the following dependency to our project:

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



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



2. JUnit 5 - Parameterized tests

- In order to implement a parameterized test we need to:
 1. Use the annotation `@ParameterizedTest` instead of `@Test`
 2. Chose an argument provider (data location):

Arguments provider	Description
<code>@ValueSource</code>	Array of String, int, long, or double
<code>@EnumSource</code>	Enumerated values (<code>java.lang.Enum</code>)
<code>@MethodSource</code>	Used to specify an static method providing an Stream of values
<code>@CsvSource</code>	CSV (comma-separated values) data
<code>@CsvFileSource</code>	CSV in a file located in the Project classpath
<code>@ArgumentsSource</code>	Class wich implements the interface <code>org.junit.jupiter.params.provider.ArgumentsProvider</code>

2. JUnit 5 - Ordered tests

- By default, tests in JUnit 5 are ordered using an algorithm that is deterministic but intentionally nonobvious
- This default order can be change using the annotation `@TestMethodOrder` and one of the following options:
 - `Alphanumeric`: sorts test methods alphanumerically based on their names and parameter lists
 - `OrderAnnotation`: sorts test methods based on values specified via the annotation `@Order`
 - `Random`: orders test methods pseudo-randomly (supports configuration of a custom seed)

2. JUnit 5 - Other features

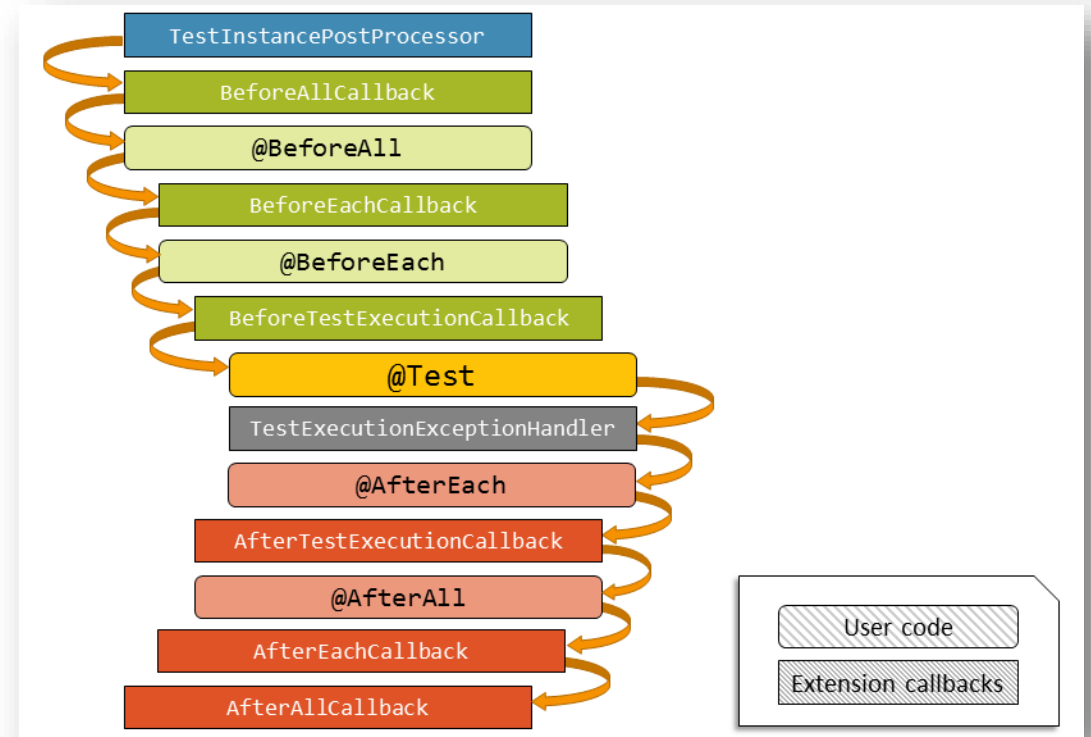
- JUnit 5 has many other features, such as:
 - Display names
 - Assumptions
 - Conditional test execution
 - Tagging and filtering
 - Nested tests
 - Repeated tests
 - Dynamic tests
 - Timeouts
 - Parallel execution
 - ...



<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**
 - WebDriver
 - Grid
4. Selenium-Jupiter
5. Final remarks and future work

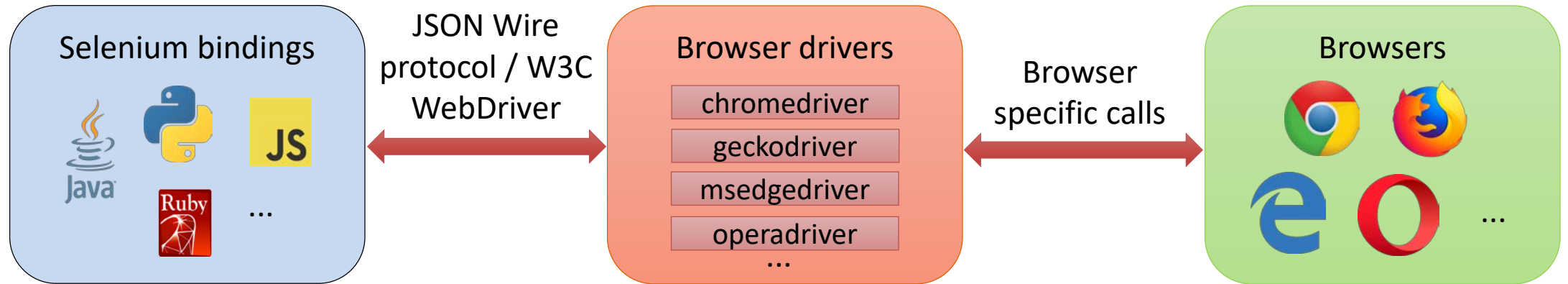
3. Selenium

- **Selenium** is a family of projects that enable the test automation with web browsers:
 - **WebDriver** allows to control different types of browsers (such as Chrome, Firefox, Opera, Edge, and so on) programmatically using different languages bindings (Java, JavaScript, Python, C#, ...)
 - **Grid** allows to drive web browsers in parallel hosted on remote machines
 - **IDE** is a browser extension (for Chrome or Firefox) which allows to record, edit and playback manual interactions with these browsers



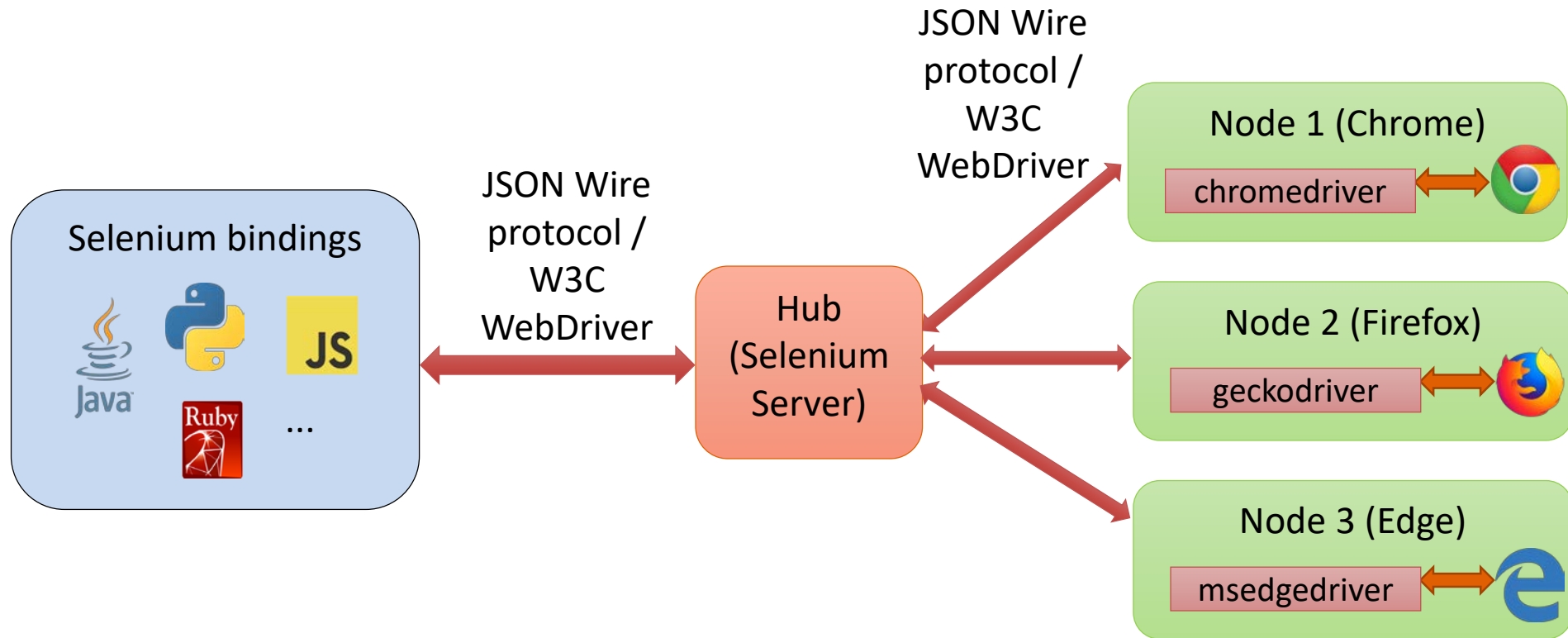
<https://seleniumhq.github.io/docs/>

3. Selenium - WebDriver



<https://seleniumhq.github.io/docs/site/en/webdriver/>

3. Selenium - Grid



<https://seleniumhq.github.io/docs/site/en/grid/>

Table of Contents

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

4. Selenium-Jupiter - Motivation

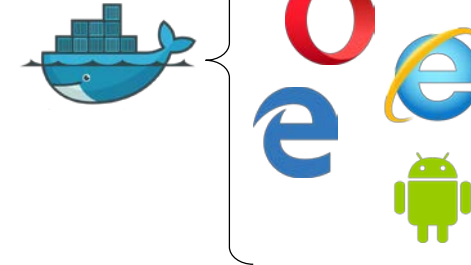
- **Selenium-Jupiter** is a JUnit 5 extension aimed to ease the use of Selenium from Java tests



Clean test code (reduced boilerplate)



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



Advanced features for tests



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

4. 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>3.3.2</version>  
  <scope>test</scope>  
</dependency>
```



Using the latest version is always recommended!

```
dependencies {  
  testCompile("io.github.bonigarcia:selenium-jupiter:3.3.2")  
}
```



4. Selenium-Jupiter - Local browsers

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

JUnit



JUnit **5**



4. Selenium-Jupiter - Local browsers

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

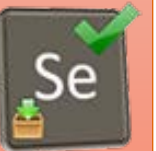
Valid types: ChromeDriver,
FirefoxDriver, OperaDriver,
SafariDriver, EdgeDriver,
InternetExplorerDriver,
HtmlUnitDriver, PhantomJS
Driver, AppiumDriver, SelenideDriver

```
@ExtendWith(SeleniumExtension.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)



4. Selenium-Jupiter - Local browsers

- Seamless integration with **Selenide** (fluent API for Selenium in Java):



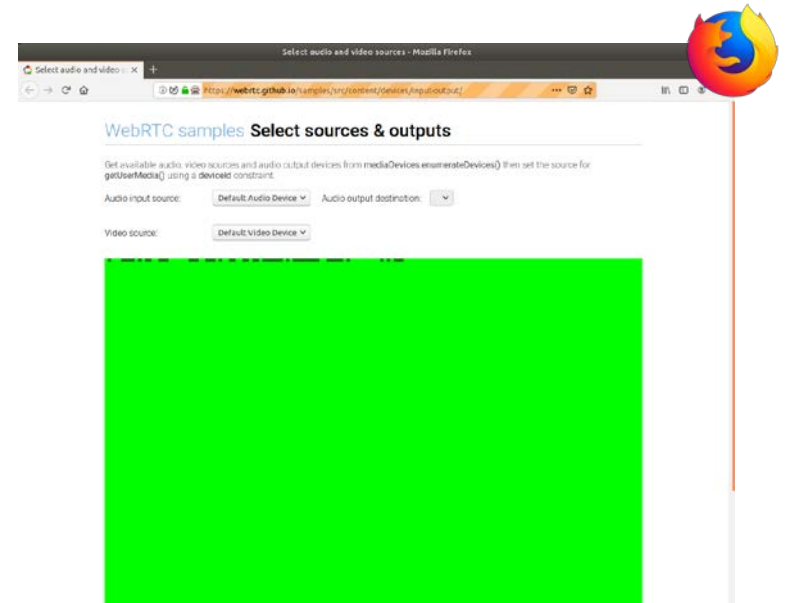
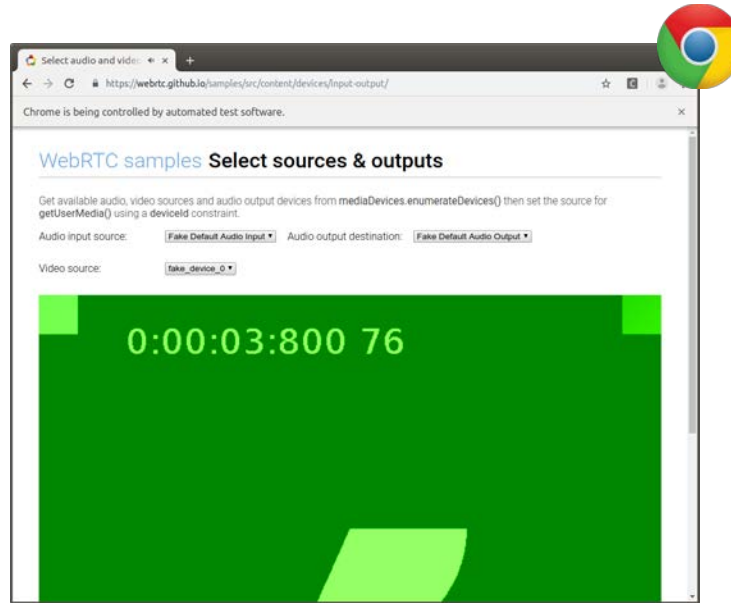
<https://selenide.org/>

```
@ExtendWith(SeleniumExtension.class)
class SelenideDefaultTest {

    @Test
    void testWithSelenideAndChrome(SelenideDriver driver) {
        driver.open(
            "https://bonigarcia.github.io/selenium-jupiter/");
        SelenideElement about = driver.$(LinkText("About"));
        about.shouldBe(visible);
        about.click();
    }
}
```

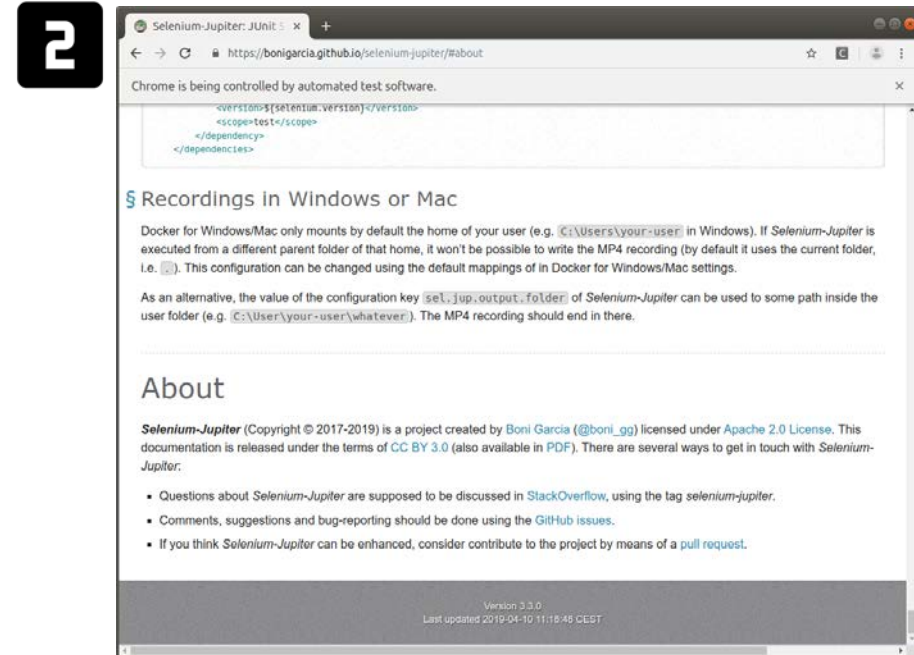
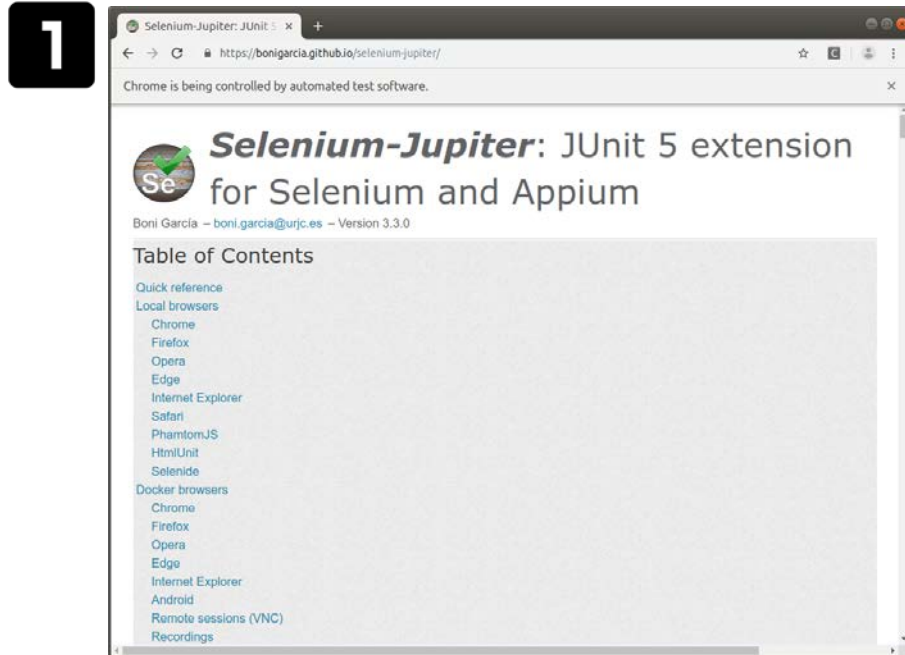

4. Selenium-Jupiter - Local browsers

- Use case: **WebRTC** applications (real-time communications using web browsers)
 - We need to specify **options** for browsers:



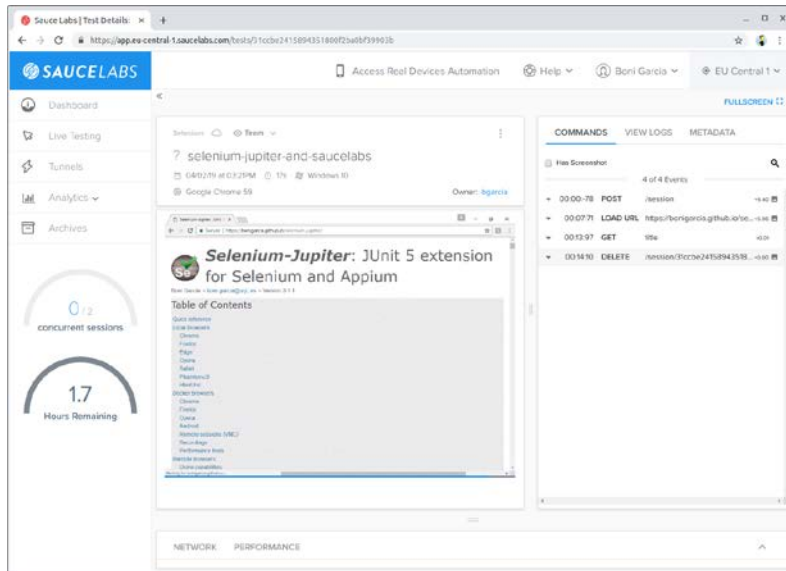
4. Selenium-Jupiter - Local browsers

- Use case: reuse same browser by different tests
 - Convenient for ordered tests (JUnit 5 feature)

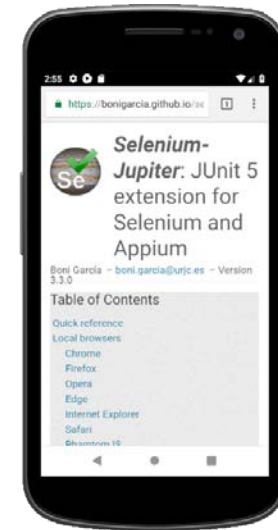


4. Selenium-Jupiter - Remote browsers

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



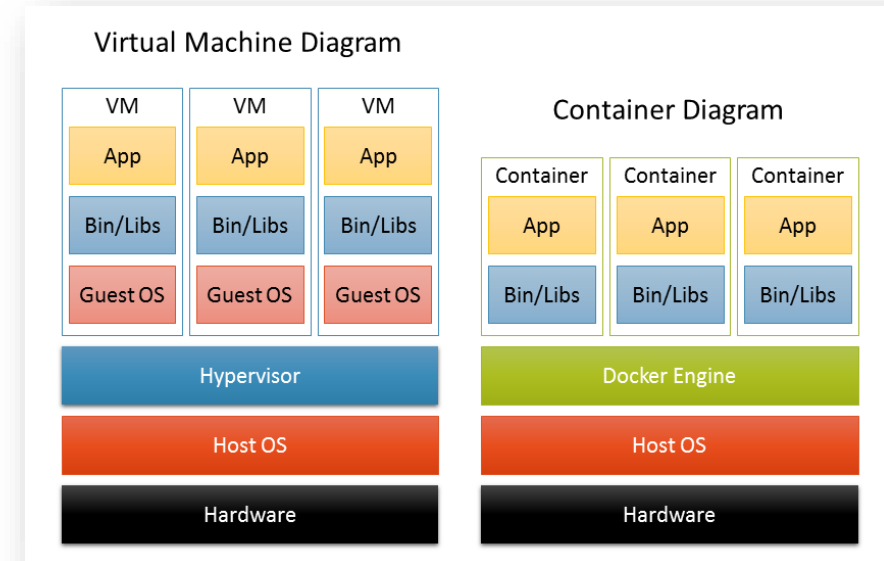
 SAUCE LABS
<https://saucelabs.com/>



 appium
<http://appium.io/>




4. 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/>

4. 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 ElasTest
 - 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



4. Selenium-Jupiter - Docker browsers

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

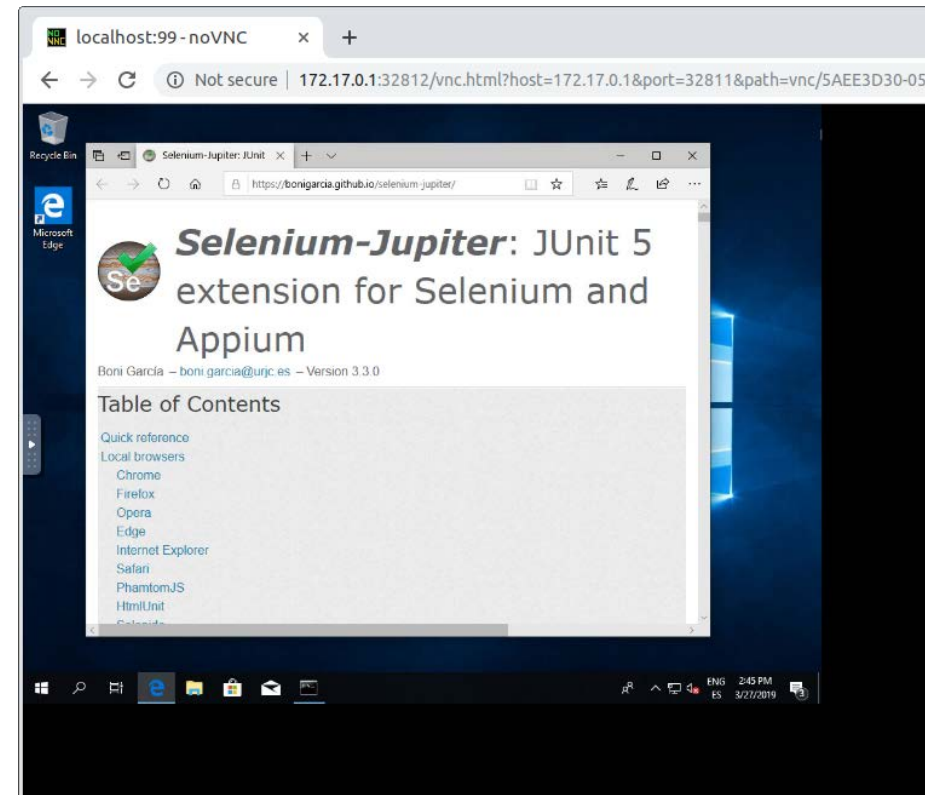
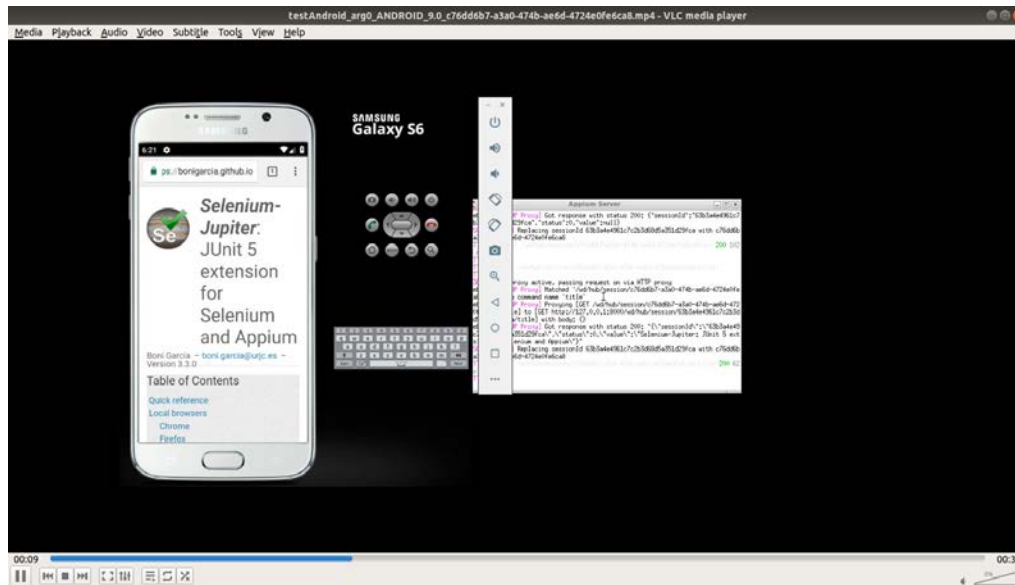
    @Test
    void testFirefoxBeta(
        @DockerBrowser(type = FIREFOX, version = "beta") RemoteWebDriver driver) {
        driver.get("https://bonigarcia.github.io/selenium-jupiter/");
        assertThat(driver.getTitle(),
            containsString("JUnit 5 extension for Selenium"));
    }
}
```

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*

4. Selenium-Jupiter - Docker browsers

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



4. Selenium-Jupiter - Docker browsers

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

Type	Device name	Android version	API level	Browser name
Phone	Samsung Galaxy S6	5.0.1	21	browser
Phone	Nexus 4	5.1.1	22	browser
Phone	Nexus 5	6.0	23	chrome
Phone	Nexus One	7.0	24	chrome
Phone	Nexus S	7.1.1	25	chrome
Tablet	Nexus 7	8.0	26	chrome
		8.1	27	chrome
		9.0	28	chrome



4. Selenium-Jupiter - Test templates

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

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

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



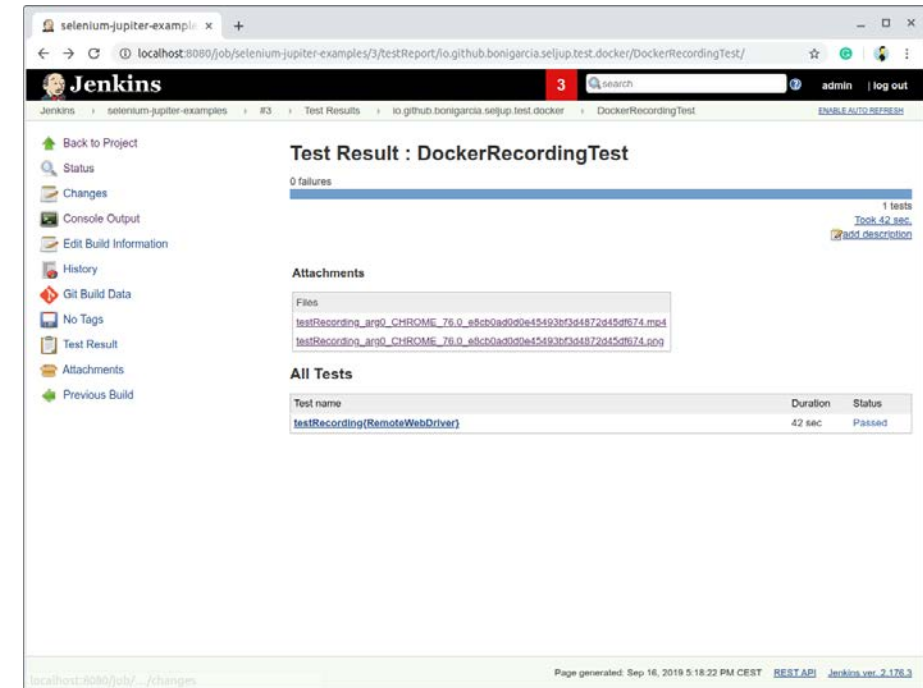
The screenshot shows the JUnit test runner interface. At the top, it says "Finished after 56,424 seconds". Below that, a summary bar indicates "Runs: 4/4", "Errors: 0", and "Failures: 0". A green progress bar is visible. The test results are listed in a tree view:

- TemplateTest [Runner: JUnit 5] (18,039 s)
 - templateTest(WebDriver) (18,039 s)
 - [Browser [type=chrome-in-docker, version=latest]] (18,039 s)
 - [Browser [type=chrome-in-docker, version=latest-1]] (12,160 s)
 - [Browser [type=chrome-in-docker, version=beta]] (12,885 s)
 - [Browser [type=chrome-in-docker, version=unstable]] (12,985 s)

4. 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
```



The screenshot shows the Jenkins web interface for a test run. The main heading is "Test Result : DockerRecordingTest". Below this, it indicates "0 failures" and "1 tests" with a duration of "Took 42 sec." and a link to "add description". Under the "Attachments" section, two files are listed: "testRecording_arg0_CHROME_76.0_e8cb0ad90e45493bf3d4872d45d9f74.mp4" and "testRecording_arg0_CHROME_76.0_e8cb0ad90e45493bf3d4872d45d9f74.png". The "All Tests" table shows a single test: "testRecording(RemoteWebDriver)" with a duration of "42 sec" and a status of "Passed".



Jenkins

4. 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-3.3.2-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-3.3.2-fat.jar server
[INFO] Selenium-Jupiter server listening on http://localhost:4042/wd/hub
```

Table of Contents

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

5. Final remarks and future work

- Selenium-Jupiter has another features such as:
 - Configurable screenshots at the end of test (as PNG image or Base64)
 - Integration with Genymotion (cloud provider for Android devices)
 - 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:
 - Implement a browser console (JavaScript log) gathering mechanism
 - Improve test template support (e.g. specifying options)
 - Improve scalability for performance tests (e.g. using Kubernetes)



Selenium-Jupiter: The next level of web testing

Thank you very much!

Boni García

 boni.garcia@urjc.es  <http://bonigarcia.github.io/>

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