

Toolbox for Selenium Tests in Java: WebDriverManager and Selenium-Jupiter

SeleniumConf Tokyo
19/04/2019

Boni García



boni.garcia@urjc.es



<http://bonigarcia.github.io/>



[@boni_gg](https://twitter.com/boni_gg)



<https://github.com/bonigarcia>

Boni García

- PhD in Information and Communications Technology from Technical University of Madrid (UPM) in Spain. Dissertation focused on software testing with Selenium
- Assistant Professor at King Juan Carlos University (URJC) in Spain
- Open source software enthusiast. Creator and maintainer of a number of projects: WebDriverManager, Selenium-Jupiter, DualSub
- Author of more than 30 research papers in different journals, magazines, international conferences, and the book Mastering Software Testing with JUnit 5



Table of contents

WebDriverManager



Selenium-Jupiter

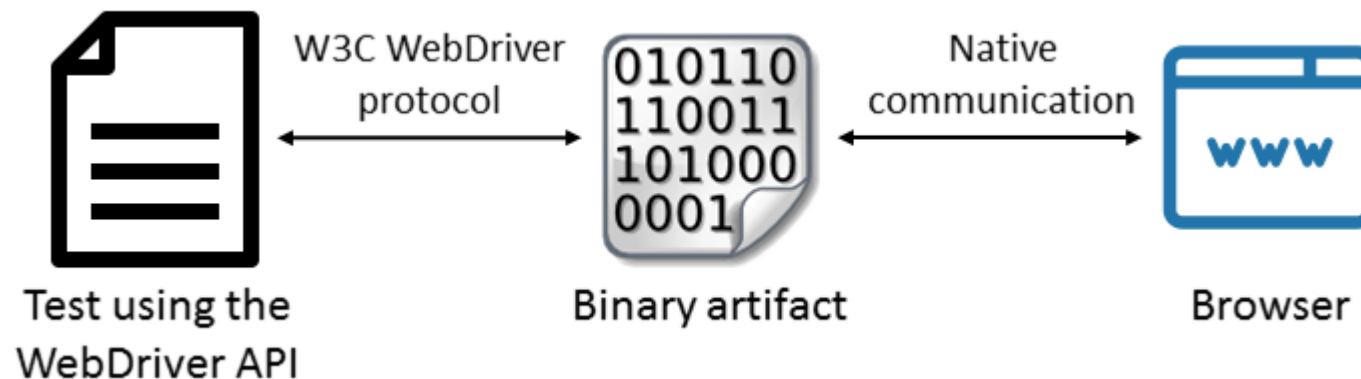


Toolbox for Selenium in Java (but not only Java)

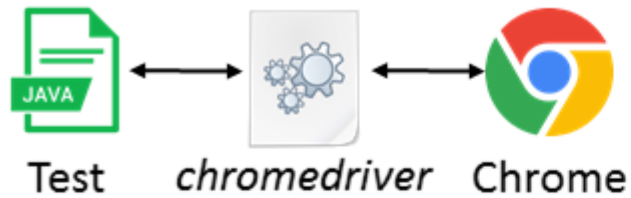


WebDriverManager - Motivation

- **Selenium WebDriver** allows to control different types of browsers (such as Chrome, Firefox, Opera, Edge, and so on) programmatically using different programming languages (Java, JavaScript, Python, C#, ...)
- Browser is driven using native mechanism, so we need a binary file (*driver*) in between the test using the WebDriver API and the actual browser



WebDriverManager - Motivation



```
System.setProperty("webdriver.chrome.driver", "/path/to/chromedriver");  
System.setProperty("webdriver.opera.driver", "/path/to/operadriver");  
System.setProperty("webdriver.ie.driver", "C:/path/to/IEDriverServer.exe");  
System.setProperty("webdriver.edge.driver", "C:/path/to/MicrosoftWebDriver.exe");  
System.setProperty("phantomjs.binary.path", "/path/to/phantomjs");  
System.setProperty("webdriver.gecko.driver", "/path/to/geckodriver");
```



WebDriverManager - Motivation

- Driver management is painful:
 - Driver (chromedriver, geckodriver, etc.) must be downloaded manually for the proper platform running the test (i.e. Windows, Linux, Mac)
 - Proper driver version must match the browser version
 - Browser are constantly updated (and drivers too)
 - Tests are not portable (different operative systems, path to driver)



WebDriverManager - Motivation



WebDriverManager - Objective

- **WebDriverManager** is a Java library that allows to **automate** the management of the binary drivers (chromedriver, geckodriver, etc.) required by Selenium WebDriver

Fork me on GitHub

```
dependencies {  
    testCompile("io.github.bonigarcia:webdrivermanager:3.4.0")  
}
```



```
<dependency>  
    <groupId>io.github.bonigarcia</groupId>  
    <artifactId>webdrivermanager</artifactId>  
    <version>3.4.0</version>  
    <scope>test</scope>  
</dependency>
```

Maven™

```
WebDriverManager.chromedriver().setup();  
WebDriverManager.firefoxdriver().setup();  
WebDriverManager.operadriver().setup();  
WebDriverManager.edgedriver().setup();  
WebDriverManager.iedriver().setup();  
WebDriverManager.phantomjs().setup();
```



<https://github.com/bonigarcia/webdrivermanager>

WebDriverManager - Objective

```
public class ChromeTest {  
  
    private WebDriver driver;  
  
    @BeforeClass  
    public static void setupClass() {  
        WebDriverManager.chromedriver().setup();  
    }  
  
    @Before  
    public void setupTest() {  
        driver = new ChromeDriver();  
    }  
  
    @After  
    public void teardown() {  
        if (driver != null) {  
            driver.quit();  
        }  
    }  
  
    @Test  
    public void test() {  
        // Your test code here  
    }  
}
```

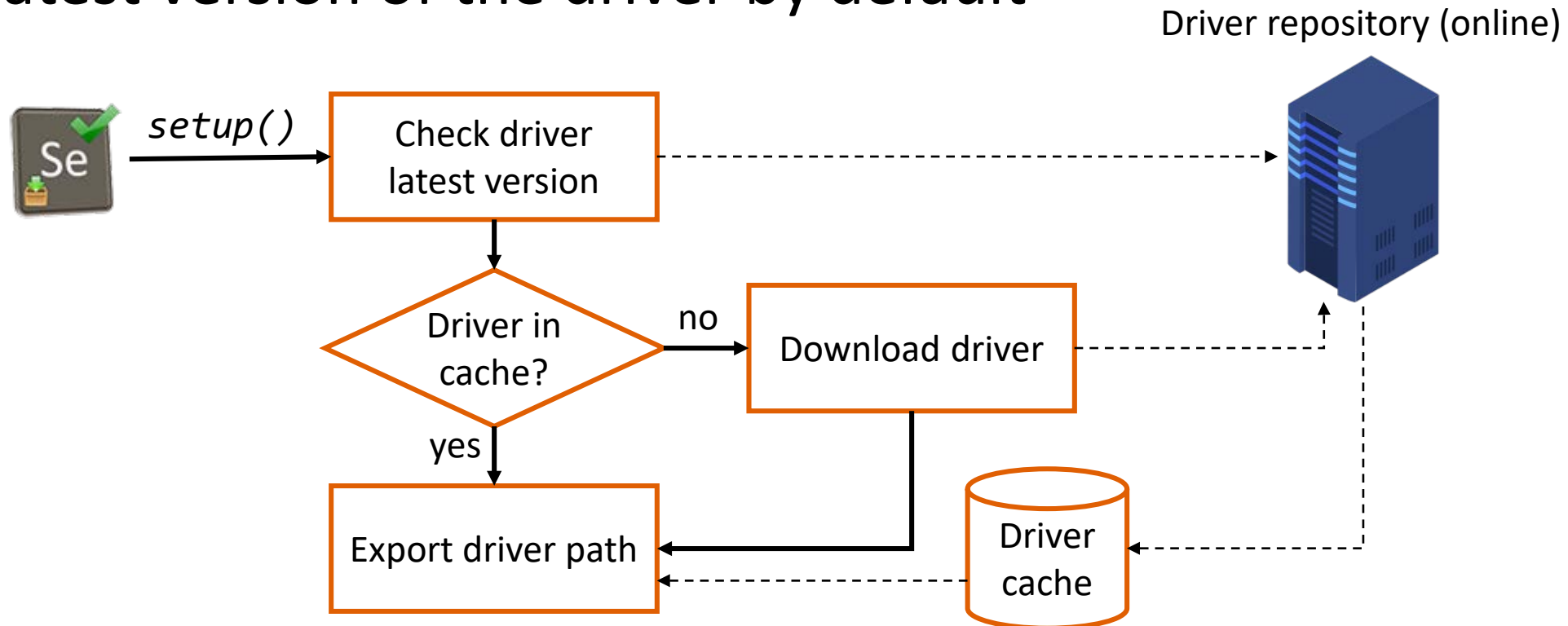


```
public class ChromeTest {  
  
    private WebDriver driver;  
  
    @BeforeClass  
    public static void setupClass() {  
        WebDriverManager.firefoxdriver().setup();  
    }  
  
    @Before  
    public void setupTest() {  
        driver = new FirefoxDriver();  
    }  
  
    @After  
    public void teardown() {  
        if (driver != null) {  
            driver.quit();  
        }  
    }  
  
    @Test  
    public void test() {  
        // Your test code here  
    }  
}
```



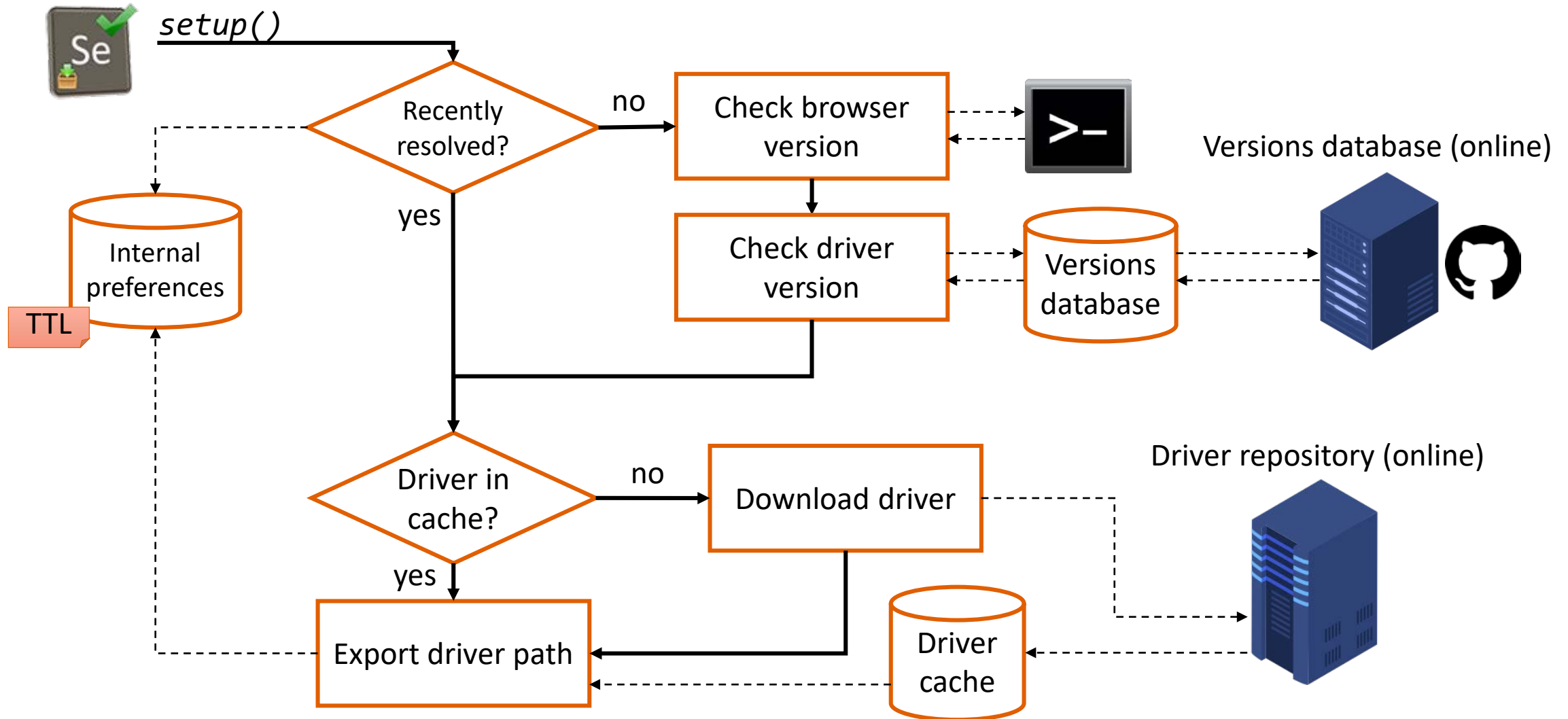
WebDriverManager - Design

- WebDriverManager was first released on 21st March 2015
- In its earlier versions, WebDriverManager downloaded the latest version of the driver by default



WebDriverManager - Design

- Currently, WebDriverManager resolution algorithm is much richer



WebDriverManager - API

- WebDriverManager exposes a fluent API. For instance:

```
WebDriverManager.chromedriver().setup();
```

Default usage for managing chromedriver

```
WebDriverManager.chromedriver().version("2.46").setup();
```

Force a given version (2.46) for chromedriver

```
WebDriverManager.firefoxdriver().arch32().setup();
```

Force 32-bit architecture for geckodriver

```
WebDriverManager.operadriver().forceDownload().setup();
```

Force the download of operadriver

```
WebDriverManager.phantomjs().avoidPreferences().setup();
```

Avoid the use of preferences for PhantomJS driver

```
WebDriverManager.edgedriver().proxy("server:port").setup();
```

Set proxy setup when managing Edge driver

WebDriverManager - API

- More examples of the WebDriverManager API:

| Method | Description |
|---|---|
| <code>version(String)</code> | Set a concrete version for the driver to be downloaded |
| <code>targetPath(String)</code> | Change cache path (by default ~/.m2/repository/webdriver) |
| <code>architecture(Architecture)</code> | Force a given architecture: 32-bits or 64-bits |
| <code>operatingSystem(OperatingSystem)</code> | Force a given OS: WIN, LINUX, MAC |
| <code>proxy(String)</code> | Use a HTTP proxy for the Internet connection |
| <code>avoidPreferences()</code> | Avoid the use of Java preferences |
| <code>driverRepositoryUrl(URL)</code> | Set a custom repository URL |
| <code>timeout(int)</code> | Change connection timeout |
| <code>browserPath()</code> | Set path for a given browser |
| <code>ttl()</code> | Change time-to-live (by default 3600 seconds) |
| <code>forceDownload()</code> | Force to download driver even if it exists in cache |

<https://github.com/bonigarcia/webdrivermanager>

WebDriverManager - Configuration

- WebDriverManager is highly configurable with:

1. Environment variables. For example

```
export WDM_TARGETPATH=~/.selenium
```

```
export WDM_CHROMEDRIVERVERSION=2.46
```

2. Java properties. For example:

```
mvn test -Dwdm.targetPath=~/.selenium
```

```
gradle test -Dwdm.chromeDriverVersion=2.46
```

3. Configuration manager in Java. For example:

```
WebDriverManager.chromedriver().version("2.46").setup();
```

```
WebDriverManager.globalConfig().setTargetPath("~/.selenium");
```

WebDriverManager - Beyond Java

- WebDriverManager can be also used:
 1. As CLI (command line interface) tool:

```
> java -jar webdrivermanager-3.4.0-fat.jar chrome
[INFO] Using WebDriverManager to resolve chrome
[INFO] Reading https://chromedriver.storage.googleapis.com/ to seek chromedriver
[INFO] Latest version of chromedriver is 2.37
[INFO] Downloading https://chromedriver.storage.googleapis.com/2.37/chromedriver_win32.zip
to folder D:\projects\webdrivermanager
[INFO] Resulting binary D:\projects\webdrivermanager\target\chromedriver.exe
```

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

```
> java -jar webdrivermanager-3.4.0-fat.jar server
[INFO] WebDriverManager server listening on port 4041
```

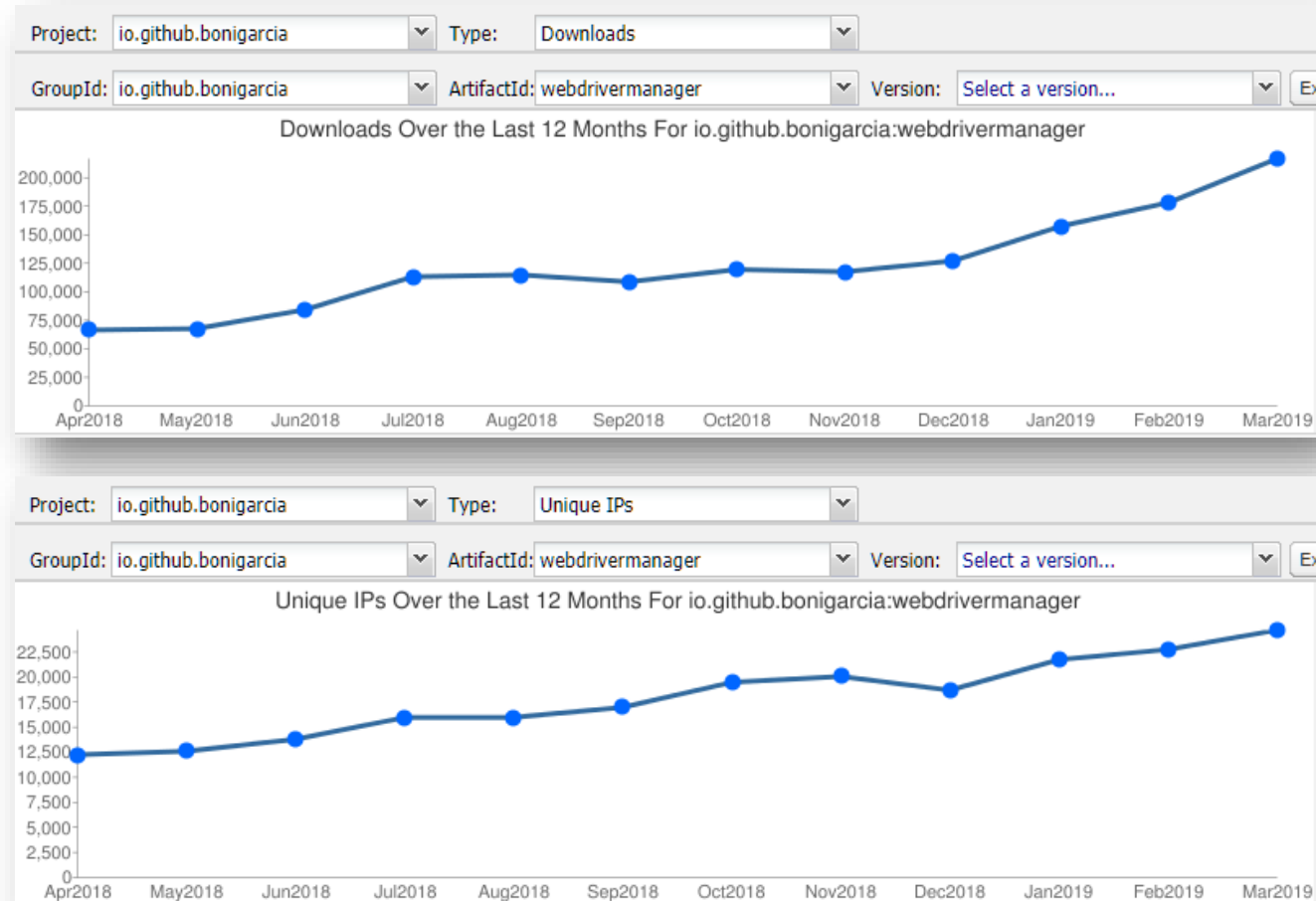
Examples of requests to WebDriverManager Server:

<http://localhost:4041/firefoxdriver>

<http://localhost:4041/chromedriver?chromeDriverVersion=2.40>

WebDriverManager - Conclusions

- WebDriverManager is a helper library for automating the management of Selenium drivers (chromedriver, etc.)



WebDriverManager - Conclusions

- WebDriverManager is used in different projects in the Selenium ecosystem. For instance:
 - io.appium » java-client: <https://github.com/appium/java-client>
 - com.codeborne » selenide: <https://github.com/selenide/selenide>
- WebDriverManager concept has been ported to other languages:
 - webdriver-manager (Node.js): <https://github.com/angular/webdriver-manager>
 - webdriver_manager (Python): https://github.com/jeffnyman/webdriver_manager
 - WebDriverManager.Net (.Net): <https://github.com/rosolko/WebDriverManager.Net>
 - Webdrivers Gem (Ruby): <https://github.com/titusfortner/webdrivers>
- WebDriverManager is in constant evolution. Its roadmap includes:
 - Support Edge based on Chromium
 - Using aspects (cross-cutting concerns) to resolve drivers automatically when instantiating WebDriver objects

Table of contents

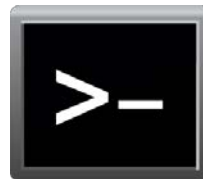
WebDriverManager



Selenium-Jupiter

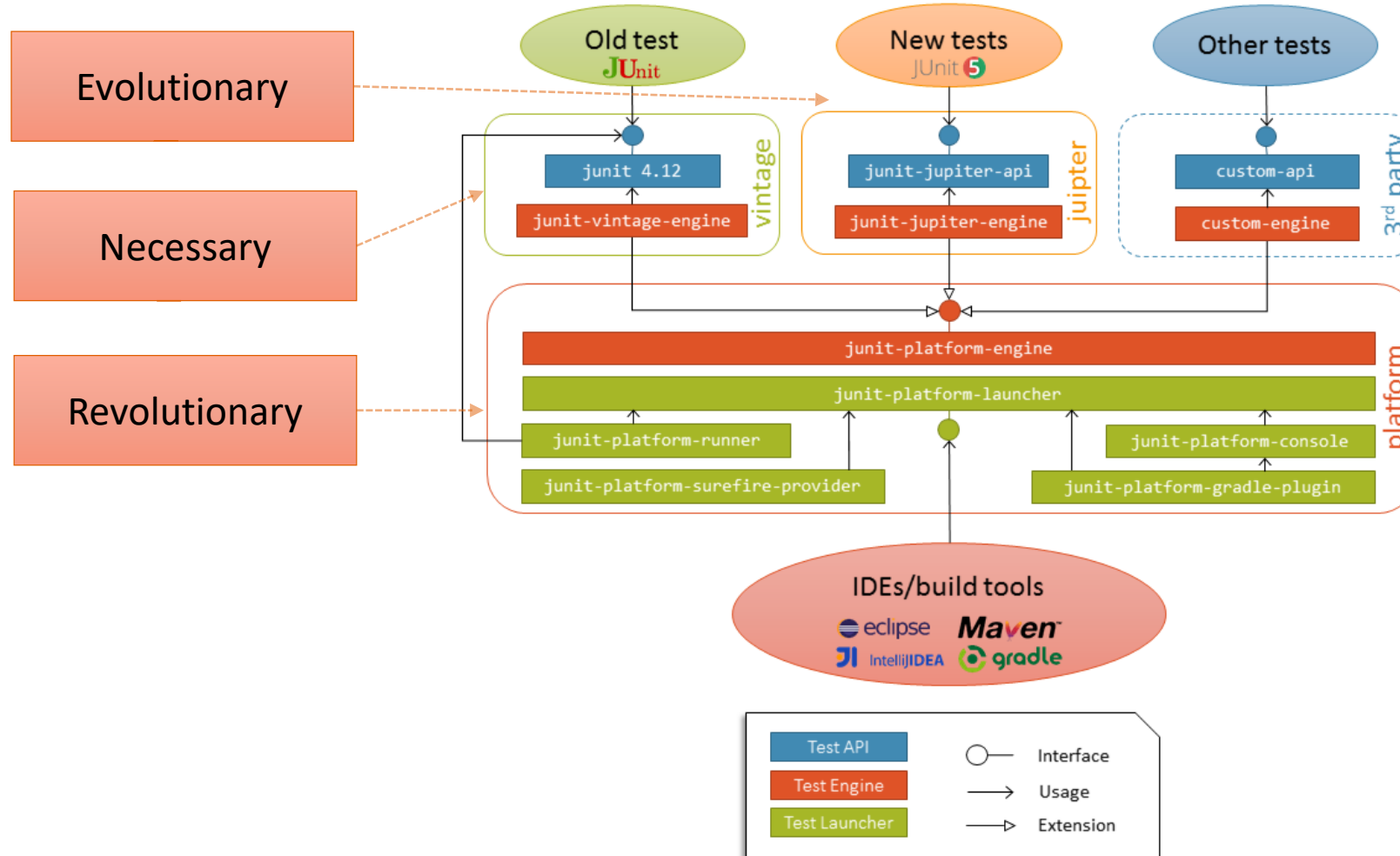


Toolbox for Selenium in Java (but not only Java)



Selenium-Jupiter - Motivation

- **JUnit 5** (stable) was first released on September 2017



JUnit 5

Selenium-Jupiter - Motivation

- **JUnit 5** provides a brand-new programming an extension model called **Jupiter**
- Basic test are similar than in JUnit 4 and provide a wide range of new features, such as:
 - Enhanced parameterized tests
 - Parallel execution
 - Test ordering
 - Kotlin support
 - ...

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

```
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
```

```
class BasicJUnit5Test {

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

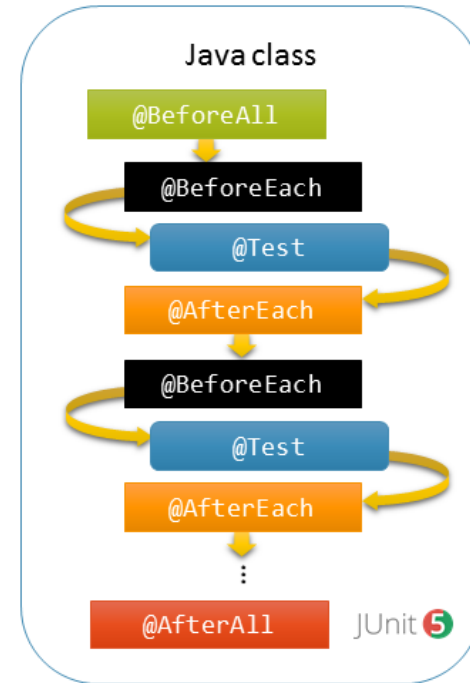
    @BeforeEach
    void setup() {
        // setup each test
    }

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

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

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

}
```

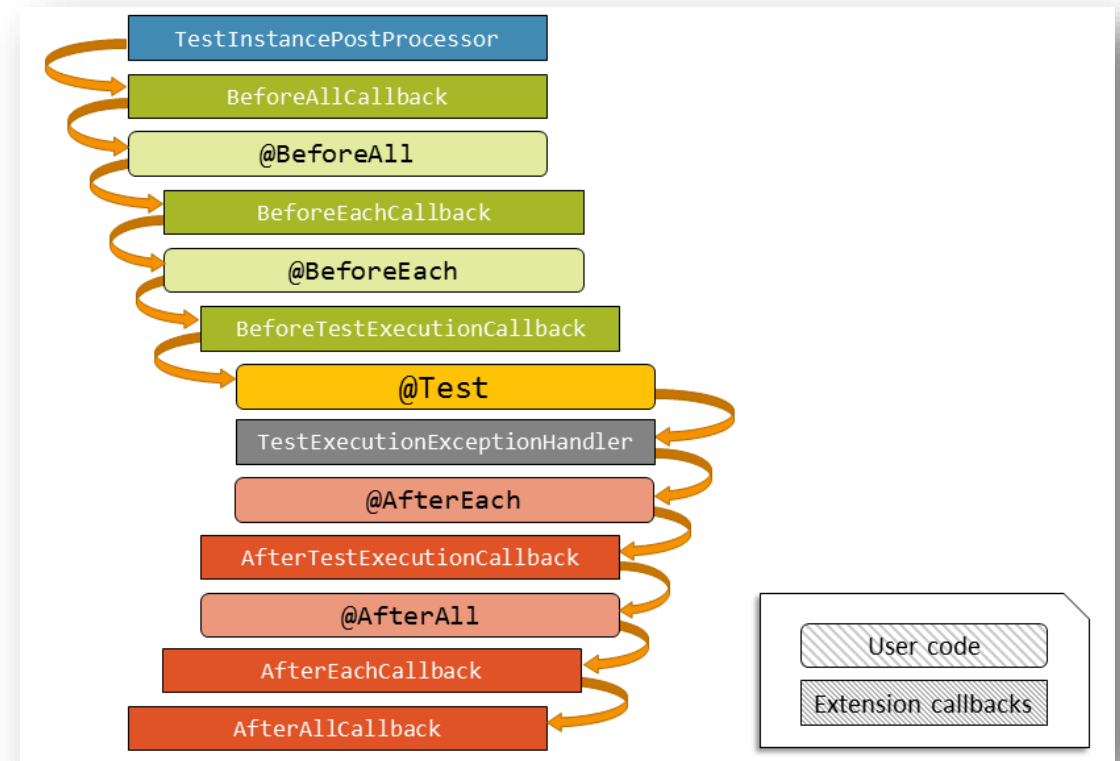


Selenium-Jupiter - Motivation

- The **extension model** of JUnit 5 allows to add custom features to the programming model through extension points:

1. Custom logic in the test lifecycle
2. Dependency injection in test methods and constructors
3. Test templates
4. Test conditional execution

Very convenient
for Selenium!



Selenium-Jupiter - Objective

- **Selenium-Jupiter** is a JUnit 5 extension aimed to ease the use of **Selenium** from Java tests
- Thanks to the Jupiter extension model, the required boilerplate to use Selenium from JUnit 5 is minimum
- Moreover, it allows to use browser and Android devices in **Docker** containers in a effortless manner

Fork me on GitHub

```
<dependency>
  <groupId>io.github.bonigarcia</groupId>
  <artifactId>selenium-jupiter</artifactId>
  <version>3.2.0</version>
  <scope>test</scope>
</dependency>
```

Maven

```
dependencies {
  testCompile("io.github.bonigarcia:selenium-jupiter:3.2.0")
}
```



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

Selenium-Jupiter - Local browsers

- Selenium-Jupiter uses the **dependency injection** mechanism to instantiate/release WebDriver objects before/after tests

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class ChromeAndFirefoxJupiterTest {

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

    @Test
    public void testWithFirefox(FirefoxDriver firefoxDriver) {
        // Use Firefox in this test
    }

    @Test
    public void testWithChromeAndFirefox(ChromeDriver chromeDriver,
        FirefoxDriver firefoxDriver) {
        // Use Chrome and Firefox in this test
    }
}
```

We simply need to specify the type of browser to be used, as test or constructor parameters:

- ChromeDriver
- FirefoxDriver
- OperaDriver
- SafariDriver
- EdgeDriver
- InternetExplorerDriver
- HtmlUnitDriver
- PhantomJSDriver
- AppiumDriver

Internally, Selenium-Jupiter uses **WebDriverManager** to resolve properly the required binary drivers to control local browsers

Selenium-Jupiter - Remote browsers

- Selenium-Jupiter provides the annotation `@DriverUrl` to locate the **Selenium or Appium server** and `@DriverCapabilities` to specify the desired capabilities

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.Capabilities;
import org.openqa.selenium.remote.RemoteWebDriver;
import io.github.bonigarcia.DriverCapabilities;
import io.github.bonigarcia.DriverUrl;
import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class SauceLabsJupiterTest {

    @DriverUrl
    String url = "https://ondemand.eu-central-1.saucelabs.com/wd/hub";

    @DriverCapabilities
    DesiredCapabilities capabilities = new DesiredCapabilities();
    {
        capabilities.setCapability("username", "<my-saucelabs-user>");
        capabilities.setCapability("accessKey", "<my-saucelabs-key>");
        capabilities.setCapability("browserName", "Chrome");
        capabilities.setCapability("platform", "Windows 10");
        capabilities.setCapability("version", "59.0");
        capabilities.setCapability("name", "selenium-jupiter-and-saucelabs");
    }

    @Test
    void testWithSaucelabs(RemoteWebDriver driver) {
        // test
    }
}
```



Selenium-Jupiter - Remote browsers

- Selenium-Jupiter provides the annotation `@DriverUrl` to locate the **Selenium or Appium server** and `@DriverCapabilities` to specify the desired capabilities

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.remote.DesiredCapabilities;

import io.appium.java_client.AppiumDriver;
import io.github.bonigarcia.seljup.DriverCapabilities;
import io.github.bonigarcia.seljup.DriverUrl;
import io.github.bonigarcia.seljup.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class AppiumWithGlobalOptionsChromeJupiterTest {

    @DriverUrl
    String url = "http://localhost:4723/wd/hub";

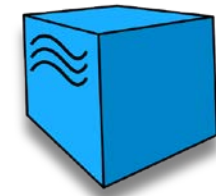
    @DriverCapabilities
    DesiredCapabilities capabilities = new DesiredCapabilities();
    {
        capabilities.setCapability("browserName", "chrome");
        capabilities.setCapability("deviceName", "Samsung Galaxy S6");
    }

    @Test
    void testWithAndroid(AppiumDriver<WebElement> driver) {
        // test
    }
}
```



Selenium-Jupiter - *Dockerized* browsers

- Selenium-Jupiter provides seamless integration with **Docker**
- The annotation `@DockerBrowser` is used to declare a *dockerized* browsers. The supported browser are
 - 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 devices are maintained in the **docker-android** project (by Budi Utomo)



Selenium-Jupiter - Dockerized browsers

- Browser in Docker containers example:

```
import static io.github.bonigarcia.BrowserType.CHROME;
import static io.github.bonigarcia.BrowserType.FIREFOX;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.remote.RemoteWebDriver;

import io.github.bonigarcia.DockerBrowser;
import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class DockerChromeJupiterTest {

    @Test
    public void testChrome(@DockerBrowser(type = CHROME) RemoteWebDriver driver) {
        // test
    }

    @Test
    public void testChromeWithVersion(@DockerBrowser(type = FIREFOX, version = "66.0")
        RemoteWebDriver driver) {
        // test
    }
}
```

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

If version is not specified, the latest stable will be used. For that, Selenium-Jupiter internally connects to **Docker Hub** to find out the latest version (*ever green Docker browser*)

The parameter *version* admits the following special values: *latest*, *latest-**, *beta*, *unstable*

Selenium-Jupiter - *Dockerized* browsers

- Browsers in Docker containers can be used to create **performance** tests:

```
import static io.github.bonigarcia.BrowserType.CHROME;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.remote.RemoteWebDriver;

import java.util.List;

import io.github.bonigarcia.DockerBrowser;
import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class DockerChromeJupiterTest {

    static final int NUM_BROWSERS = 10;

    @Test
    public void testPerformance(
        @DockerBrowser(type = CHROME, size = NUM_BROWSERS) List<RemoteWebDriver> driverList) {
        // test
    }
}
```

In this test, we will have 10 Chrome browsers ready to be used by the test logic

Selenium-Jupiter - *Dockerized* browsers

- Android in Docker container example:

```
import static io.github.bonigarcia.BrowserType.ANDROID;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.remote.RemoteWebDriver;

import io.github.bonigarcia.DockerBrowser;
import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class DockerAndroidCustomJupiterTest {

    @Test
    public void testAndroid(@DockerBrowser(type = ANDROID, version = "8.1",
        deviceName = "Nexus S") RemoteWebDriver driver) {
        // test
    }
}
```

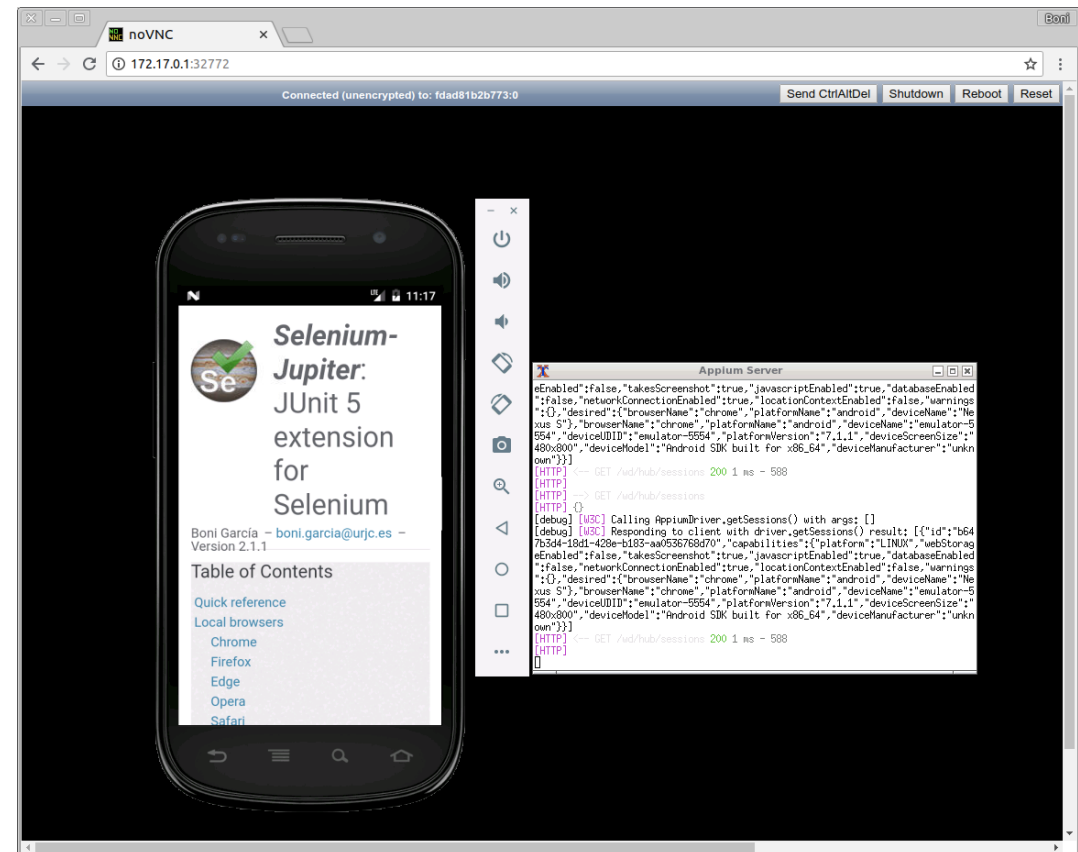
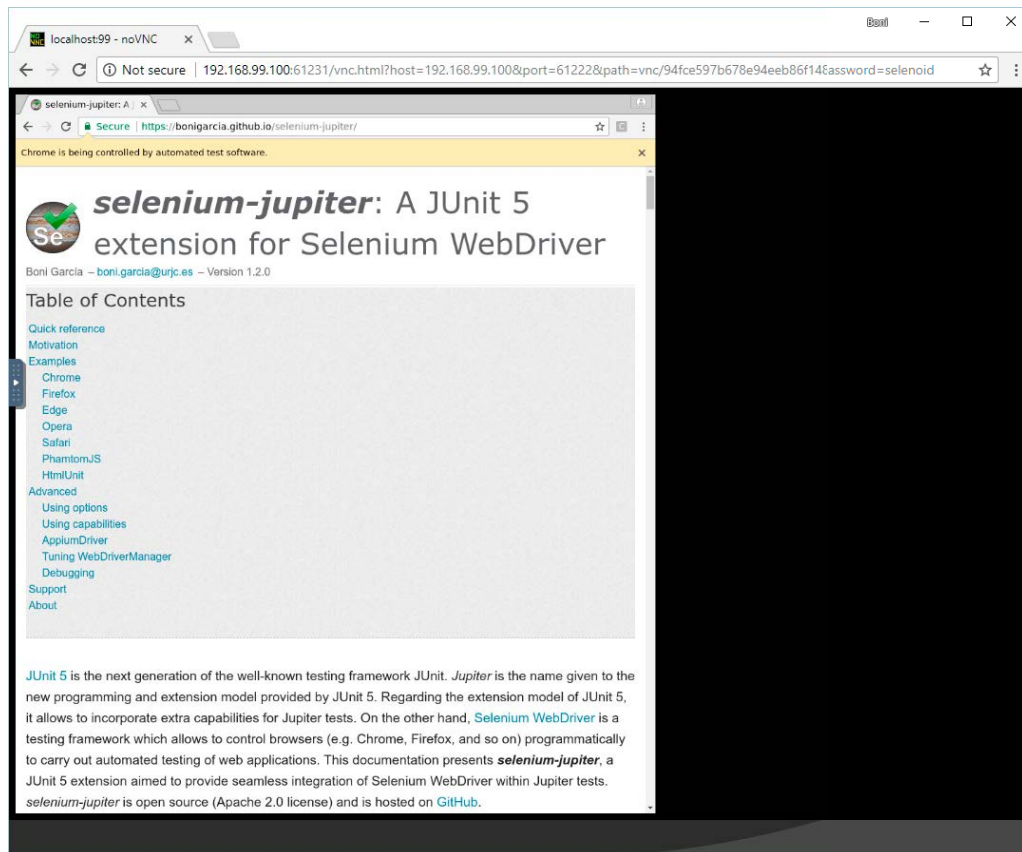
When using Android in Docker containers, the type of device can be specified

| 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 |

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

Selenium-Jupiter - Dockerized browsers

- When using Docker containers, it is possible to interact with the remote session using **VNC** (and also recording these sessions)



Selenium-Jupiter - Test templates

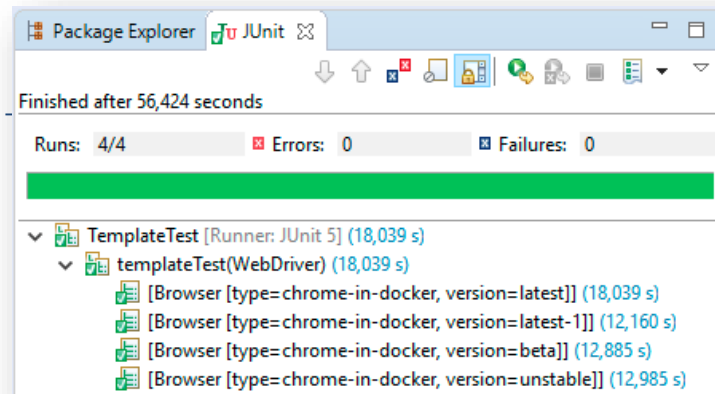
- Selenium-Jupiter use the JUnit 5's support for test templates
- A template defines the number and types of browser used by a test:
 1. By means of a JSON file:

```
import org.junit.jupiter.api.TestTemplate;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.WebDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class TemplateTest {

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



```
{
  "browsers": [
    {
      "type": "chrome-in-docker",
      "version": "latest"
    },
    {
      "type": "chrome-in-docker",
      "version": "latest-1"
    },
    {
      "type": "chrome-in-docker",
      "version": "beta"
    },
    {
      "type": "chrome-in-docker",
      "version": "unstable"
    }
  ]
}
```

Selenium-Jupiter - Test templates

- Selenium-Jupiter use the JUnit 5's support for test templates
- A template defines the number and types of browser used by a test:

2. Programmatically:

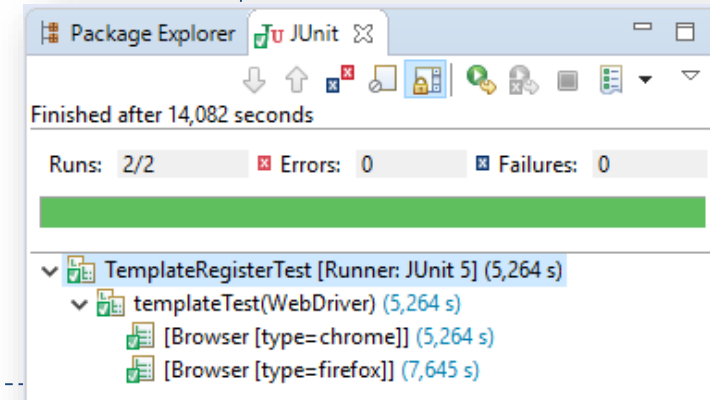
```
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.TestTemplate;
import org.junit.jupiter.api.extension.RegisterExtension;
import org.openqa.selenium.WebDriver;
import io.github.bonigarcia.BrowserBuilder;
import io.github.bonigarcia.BrowsersTemplate.Browser;
import io.github.bonigarcia.SeleniumExtension;

public class TemplateRegisterTest {

    @RegisterExtension
    static SeleniumExtension seleniumExtension = new SeleniumExtension();

    @BeforeAll
    static void setup() {
        Browser chrome = BrowserBuilder.chrome().build();
        Browser firefox = BrowserBuilder.firefox().build();
        seleniumExtension.addBrowsers(chrome, firefox);
    }

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



Selenium-Jupiter - Configuration

- Selenium-Jupiter is also highly configurable with:

1. Environment variables. For example

```
export SEL_JUP_VNC=true
```

```
export SEL_JUP_RECORDING=true
```

2. Java properties. For example:

```
mvn test -Dsel.jup.vnc=true
```

```
gradle test -Dsel.jup.vnc=true
```

3. Configuration manager in Java. For example:

```
@RegisterExtension
static SeleniumExtension seleniumExtension = new SeleniumExtension();

@BeforeAll
static void setup() {
    seleniumExtension.getConfig().setVnc(true);
    seleniumExtension.getConfig().setRecording(true);
}
```

Selenium-Jupiter - Beyond Java

- Selenium-Jupiter can be also used:

1. As CLI (command line interface) tool:

```
> java -jar selenium-jupiter-3.2.0-fat.jar chrome
[INFO] Using SeleniumJupiter to execute chrome (latest) in Docker
[INFO] Using CHROME version 73.0 (latest)
[INFO] Starting Docker container aerokube/selenoid:1.8.4
[DEBUG] Creating WebDriver for CHROME at http://172.17.0.1:32784/wd/hub
Jan 07, 2019 6:55:17 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: OSS
[INFO] Starting Docker container psharkey/novnc:3.3-t6
[INFO] Session id 8edd28c130bb2bc62f8e4467c20f4dc0
[INFO] VNC URL (copy and paste in a browser navigation bar to interact with remote session)
[INFO]
http://172.17.0.1:32785/vnc.html?host=172.17.0.1&port=32784&path=vnc/8edd28c130bb2bc62f8e44
67c20f4dc0&resize=scale&autoconnect=true&password=selenoid
[INFO] Press ENTER to exit

[INFO] Stopping Docker container aerokube/selenoid:1.8.4
[INFO] Stopping Docker container psharkey/novnc:3.3-t6
```

Selenium-Jupiter allows to control Docker browsers through VNC

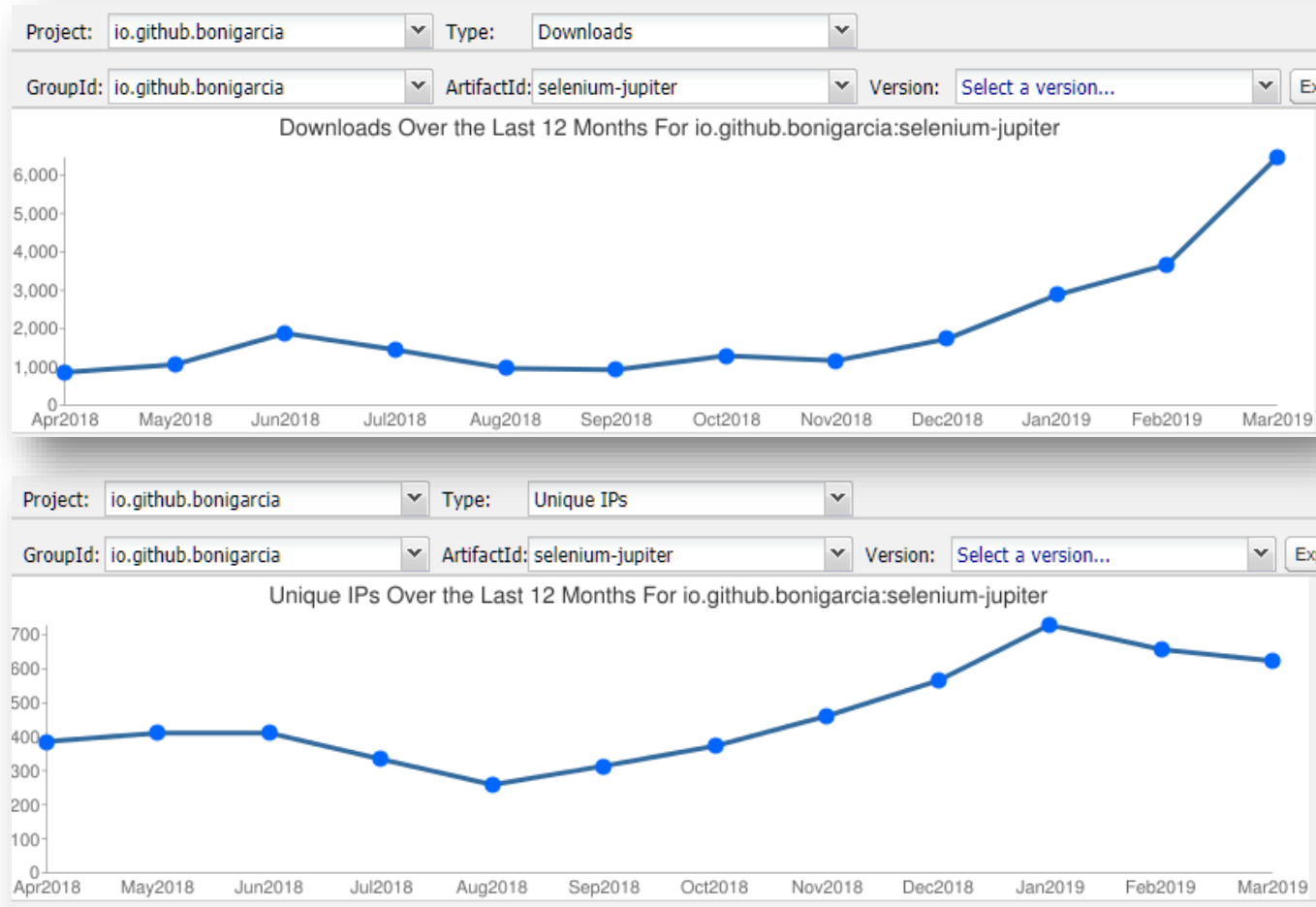
2. As server (using a REST-like API):

```
java -jar webdrivermanager-3.2.0-fat.jar server
[INFO] Selenium-Jupiter server listening on http://localhost:4042/wd/hub
```

Selenium-Jupiter becomes a **Selenium Server**

Selenium-Jupiter - Conclusions

- Selenium-Jupiter is a JUnit 5 extension for Selenium (WebDriver, Grid) and Appium



Selenium-Jupiter - Conclusions

- Selenium-Jupiter has much more features such as:
 - Using WebDriver `@Options` in tests (e.g. `ChromeOptions`, `FirefoxOptions`, etc.)
 - Screenshots at the end of test (as PNG image or Base64)
 - Integration with Jenkins (publishing test results in the Jenkins GUI)
 - Integration with Genymotion (cloud provider for Android devices)
 - Generic driver (configurable type of browser)
 - Single session (reuse browser in different tests)

<https://bonigarcia.github.io/selenium-jupiter/>
- Selenium-Jupiter is evolving in the near future. Its roadmap includes:
 - Improve test template support (e.g. specifying options within the template)
 - Improve scalability for performance tests (candidate technology: **Kubernetes**)

Toolbox for Selenium Tests in Java: WebDriverManager and Selenium-Jupiter

Thank you very much!
Q&A

Boni García



boni.garcia@urjc.es



<http://bonigarcia.github.io/>



[@boni_gg](https://twitter.com/boni_gg)



<https://github.com/bonigarcia>