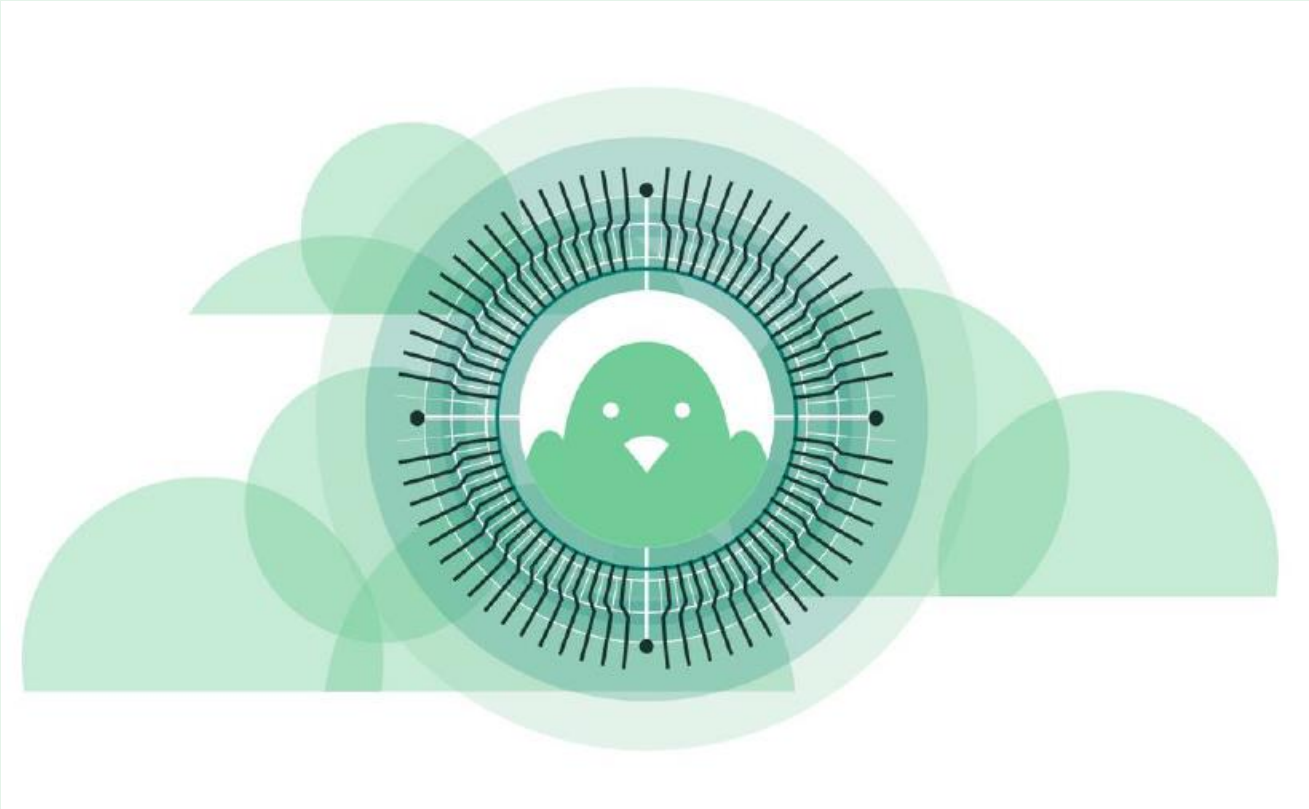
Test Automation of UI Tests using Selenium and Appium



Testbirds

Content

Author: Georg Hansbauer

Georg Hansbauer is co-founder and Managing Director of Testbirds. He is responsible for the development of services and IT infrastructure as well as finance and quality assurance at Testbirds. Georg gathered extensive experience in the field of enterprise testing – from automated tests for complete IT service desks to load testing – and has been in charge of various IT projects for international corporations. He graduated in the elite network Master's program "Finance and Information Management" at the University of Augsburg and the Technical University of Munich in Germany and has a Bachelor's degree in Business Informatics.

1. New Platforms and Processes Require Rethinking Quality Assurance

Just a few years ago, websites were only tested over desktop devices and a handful of browsers. Nowadays, Quality Assurance is facing completely new challenges due to the enormous variety of mobile and smart devices in the market. Test automation has therefore now become a crucial factor for quality control, although most companies are still merely scratching its surface, especially when it comes to mobile applications.

This whitepaper provides an entry into the world of test automation and investigates its challenges through the use of practical use cases and specific fields of application. It provides information for decision makers, who are considering investing in test automation, while also granting insights for developers and users looking to increase their knowledge in this area.

The classic waterfall model with its stiff development and testing phases is increasingly replaced by agile methods. Short iterations, so called springs, are causing short testing phases with a high rerun frequency. DSoftware is being updated in small cycles due to approaches like Continuous Integration In parallel to development, also testing takes place in close intervals (so called Continuous Testing) and challenges internal as well as external teams.

The classic waterfall model, with its rigid development and testing phases, is more and more often being replaced by agile methods. Short iterations, so called sprints, are causing shorter testing phases with a high rerun frequency. Due to methods such as Continuous Integration, software is now being updated in smaller cycles. In parallel to development, testing is also taking place in close intervals (otherwise known as Continuous Testing) posing significant challenges for internal and external teams.

Furthermore, costs are on the rise due to the fact that not only new, but also existing components and features need to be continuously tested. A combination of manual and automated tests is therefore an ideal way to overcome this challenge. While new features can easily and quickly be tested manually, test automation is a cheaper and more resource effective option for testing existing core functions in the long term. Recurring regression tests, for example, are therefore ideal for test automation.

Finally, an important prerequisite for agile development is a smooth collaboration between software developers and IT companies, which is summed up by the buzzword DevOps. A central idea to DevOps is to automate deployment and testing process, which can make this working relationship much easier.

Test automation is therefore becoming an essential element of agile software development. But what are the concrete applications when we look at it in practice?

This whitepaper offers explanations to the following questions:

1. How can developers reduce manual testing efforts with UI automation using Selenium and Appium?
2. How can QA Managers integrate automated tests into their existing development cycles?
3. What does the corresponding testing infrastructure look like?

2. Test Automation

2.1. Areas of application

A question that is often asked is whether to perform manual or automated testing? The first step is to determine whether the manual alternative is more economical. Another aspect to consider is that tight project deadlines often create time restrictions, which makes test automation appear to be the ideal solution. Nevertheless, test automation is far from a universal solution. In contrast to manual testing, the entry barrier for test automation is higher. Creating the necessary test cases is complex and therefore cost intensive. QA teams need to have the required expertise to perform test automation, otherwise external consultation is a must. After initial setup, most of the work is purely maintenance and extension of existing testing scripts, which is comparatively inexpensive. Automated tests can therefore be effectively integrated as a set component of the development process.

Another advantage of test automation is the reduction of human error. This method is especially applicable to critical core functions such as login, registration, booking and purchasing processes, as those components rarely ever change.

Manual testing on the other hand offers the possibility to gain subjective user impressions on how they rate the usability, the harmonic design or the logical structure of a digital product. This information is essential when looking at design and for this reason is particularly important during early development stages and the final version of the product.

Fast feedback which is possible through test automation is a great help for developers, especially in an agile environment with its short development cycles. This is due to the possibility to test immediately after changes in the code. With Continuous Integration systems this process can be automated in a efficient way. The test results are thereupon transparent and available for everyone involved within a short span of time.

Quick feedback is possible through test automation, which is a great help to developers, especially those working in an agile environment with short development cycles. This is due to the fact that they are able to test immediately after changes in code take place. With Continuous Integration systems this process can be automated in an efficient manner. The test results are transparent and available to everyone involved in a short time span.

Generally speaking, determining which testing method is feasible and cost efficient needs to happen on a case-by-case basis. The more often a test is executed the more profitable it can be to create a testing script, which can then be adjusted if the code changes. As a general rule of thumb, it is useful to use both automated and manual testing. The question, however, is which testing method is more suitable for which component.

Manual	Automated

• Tests are performed infrequently
• Software constantly and largely changes (high maintenance of the test script)
• Early development stage of software
• Missing know-how concerning test automation processes
• The test scope needs to be operated by humans (behavior cannot be simulated through automated test cases)
  ○ Exploratory tests
  ○ Usability tests
	• High frequency of regression tests
	• Recurring high time requirements when testing manually
	• Short release cycles
	• System environment is also a part of the test scope (Live Monitoring)
	• Significantly more cost efficient
	• Scaling, among other on additional test environments
	• Integration of tests directly into the development process (Continuous Integration)

Figure 1: Application scenarios for manual and automated testing

2.2. API Examples: Selenium and Appium

The creation of automated tests requires a specific framework. In this section, two of the most common frameworks are presented: Selenium for browser based tests in desktop environments and Appium for mobile browsers and apps.

Introduction to Selenium

Selenium is a framework for automating website and mobile apps tests. This is done by simulating user activities and by taking remote control of browsers. Actions like opening a page, scrolling, clicking on single elements or text fields are all possible.

In this manner Selenium offers countless possibilities to automate the process of a website test. These commands can be used in several different programming languages. This is how a test script emerges to check a website's functions: it performs all the aforementioned actions one after another and evaluates results.

A big advantage of Selenium is its compatibility with common systems, as this is an important requirement for testing desktop devices.

Client APIs	Platforms	Browser
• Java • PHP • Python • JavaScript • and many more	• Windows • Linux • OS X • Android • iOS	• Chrome • Firefox • Internet Explorer (from Version 7) • Safari • and many more

Figure 2: Compatibility of Selenium

Local Testing

The broad support of different browsers and platforms is made possible by Selenium's architecture:

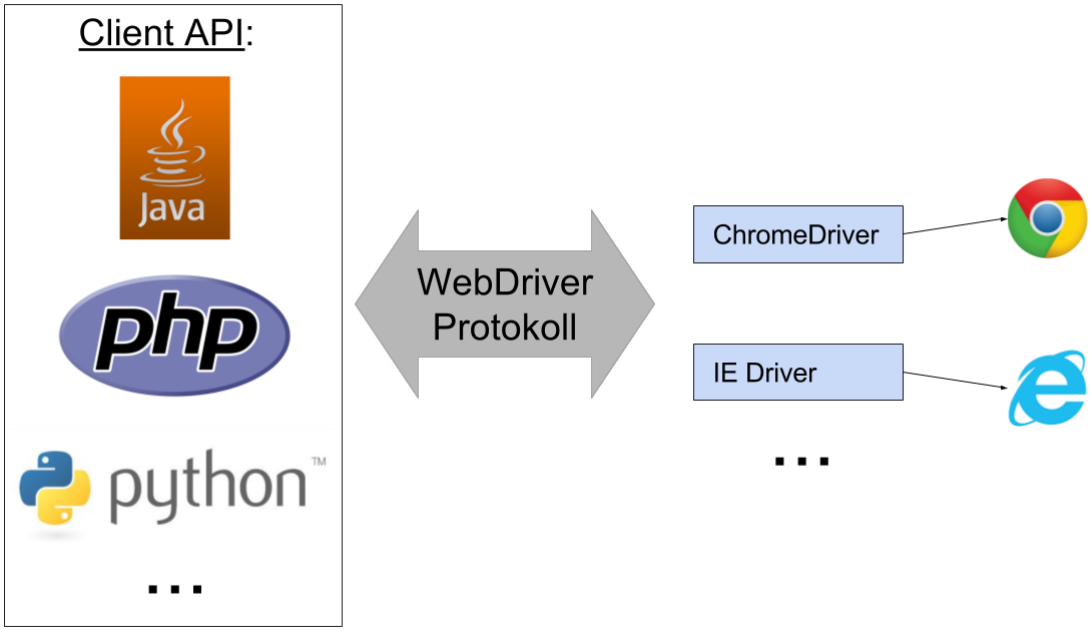


Figure 3: Local Testing with Selenium by Remotely Controlling Browsers

Figure 3 shows the two essential components of Selenium:

- **Selenium WebDriver**: A tool for automating web application testing through W3C standardised interface, which receives commands via HTTP. There are browser and platform specific implementations (ChromeDriver, IE Driver…), which are performed on the same system as the test browser. The corresponding WebDriver implementation starts the browser and performs the desired actions.
- **Selenium Client API**: The interface for the most popular programming languages are addressed by test scripts. The Client API translates those calls into WebDriver commands, which are sent to the WebDriver and performed on the browser.

Selenium Grid

The Selenium Grid allows the performance of test scripts on other systems than your own. In the process, a hub and multiple nodes are merged to one grid. This is what it looks like:

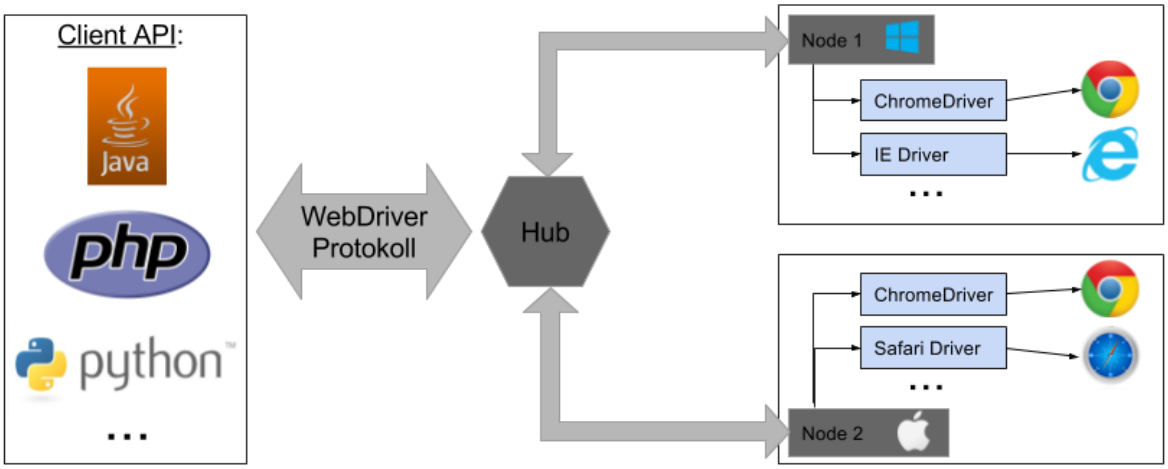


Figure 4: Testing in a Selenium Grid with browsers on various systems

The nodes are located on various machines with different possible operating systems (Windows, Linux and OS X). They address the WebDrivers, that are installed there, which once again perform the commands in the particular browsers.

The Hub and Nodes are connected through the network. In this scenario, testing scripts communicate only with the hub, which then shares the commands with the node that is currently being used. The selection of systems and browsers, on which tests should take place, happens through so called DesiredCapabilities (see Chapter 4).

The Selenium Grid is especially suitable for advanced application scenarios. It provides the advantage of testing on more machines than are available internally. This is particularly beneficial when it comes to Continuous Integration processes, in which tests are started by, for example, Jenkins.

Appium

Appium is an extension of Selenium for testing mobile apps and websites on Android and iOS devices. It enables the performance of Selenium tests in mobile browsers and the testing of native apps. Appium enhances the functionality of the WebDriver to make important actions on mobile devices such as multi touch capabilities or pushing physical buttons possible. Appium be easily integrated into an existing Selenium Grid or used as an autonomous framework.

2.3. Test Suite Example: Jubula

One simple and accessible alternative to test case creation through the use of a programming language is Jubula, an open source tool created by the Eclipse Foundation.

Jubula enables the testing of websites and applications without any programming knowledge. This is achieved by creating test cases in a graphic user interface. First, the testing process needs to be defined and is afterwards automated with single testing steps acquired from an extensive library.

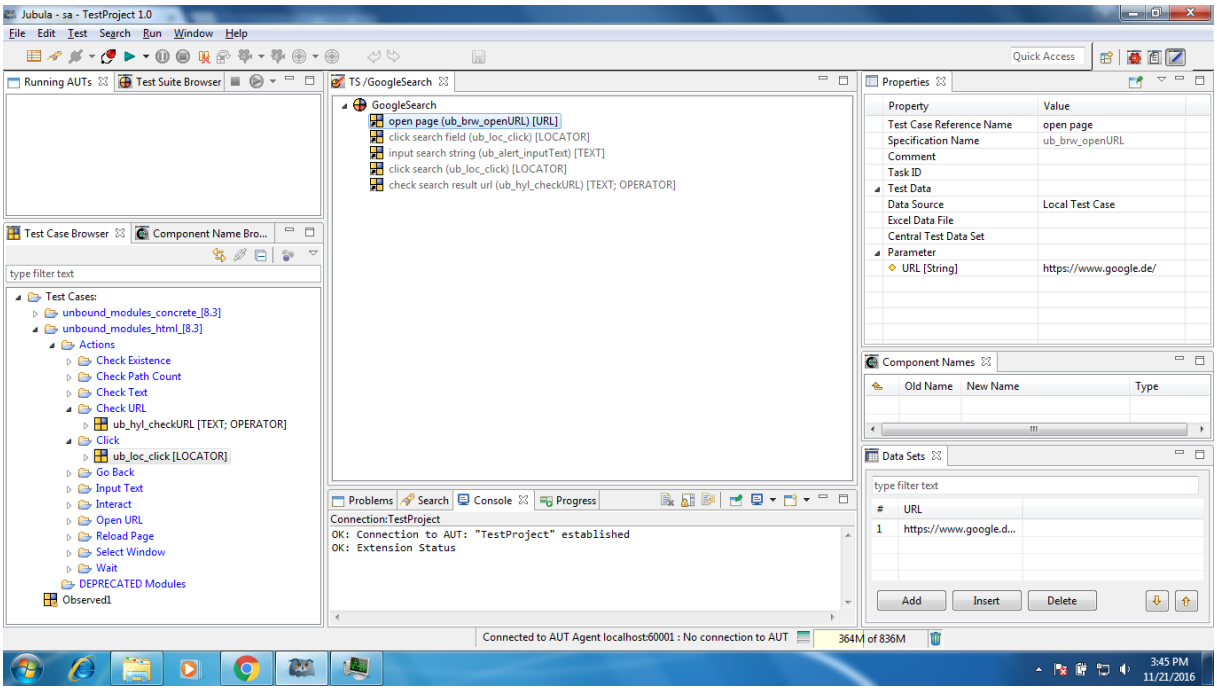


Figure 5: Jubula's User Interface

When creating the test, abstract names are used for the needed GUI elements on the website, such as "Username entry field". If a functional prototype of the website exists, those abstract names are linked to the actual technical elements. Therefore, test creation with Jubula can be performed independently in the development stage of applications and websites.

There is also a reporting tool included that collects test results, presents them in an appealing manner and if an error is encountered, automatically generates screenshots for documentation.

Jubula is based on a client-server architecture that also allows distributed testing. By doing so, test execution can easily be bound to a Continuous Integration System.

Other than websites, desktop applications can also be tested with Jubula, for example based on Swing, SWT or JavaFX toolkits. Various platforms such as Windows and Linux/Unix are also supported.

3. Test Environments for Automated Tests

3.1. The Challenge of Platform Diversity

Applications today need to run flawlessly on desktop devices as well as on mobile platforms. In addition, there is software, such as Java and Flash, that is needed to execute certain applications. The combination of possibilities of different operating systems and software programs are seemingly never ending. For this reason, the creation of corresponding test environments is costly and takes a lot of time.

Virtual machines (VMs) can help reduce these expenses. When creating test environments for automated tests, which are conducted by virtual machines, the configuration and maintenance expenses are only slightly lower than when using physical environments. Additionally, for manual tests, virtual machines are being used more often as no QA department is able to cover all the various devices on the market with their own device pool any longer.

The biggest platform variety – and therefore also the biggest challenge when it comes to testing - is found with mobile applications. Especially the market for Android devices is enormously fragmented due to the huge number of manufacturers and operating system versions. Also iOS apps should run smoothly on various different devices and with several OS versions.

The largest variety of platforms, and therefore also the largest challenge when it comes to testing, is found when looking at mobile applications. The market for Android devices is especially fragmented due to the huge number of manufacturers and operating systems in the market. iOS apps also need to run smoothly on the various different devices and all their OS versions.

This creates a situation where test environments for automated testing need to cover an ever growing number of device, browser and operating system combinations. Thanks to short development cycles, they also need to be available as quickly as possible. In addition, it's useful to test various environments in parallel with each other as this minimises the duration of single test phases.

3.2. Cloudtesting with TestChameleon™

The Software-as-a-Service-Solution TestChameleon™ is a part of Testbirds' Cloud Solutions. It creates an enormous amount of virtual machines for manual and automated testing in a matter of minutes. This includes desktop based systems as well as mobile devices and a variety of other software components. Other than the execution of manual tests, TestChameleon™ lets you perform automated test cases in Selenium or Appium with the help of a suitable API. In contrast to testing in physical environments, the costs for maintenance and the resources for creating the test environment are reduced to a minimum.

A Solution to Platform Diversity

When using TestChameleon™ you are able to choose the number of virtual machines and the necessary combinations of operating systems, browsers and other software with the help of a web configurator. Various software packages and plugins, like Java and Flash, as well as their different versions, can also be selected.

TestChameleon™ has been developed to specifically meet the needs of modern Quality Assurance. Manual tests, such as the reproduction of bugs or usability problems can be performed through the HTML5 frontend.

To support automation projects, APIs for Selenium and Appium are available. Once these have been integrated, the test environment can be defined. In doing so, TestChameleon™ serves as a management tool for automated tests and can be linked to Continuous Integration Systems such as Jenkins.

All test environments are created from scratch to ensure a maximum level of security. There is also no reuse of systems that have been previously used. After the test, the environment is completely deleted. It is also possible to connect local networks through a standardised IPSec VPN solution.

Overview of TestChameleon™

- Virtual environment creation tool, SaaS solution in the cloud
- Around 2.5 million possible combinations of devices, operating systems and software packages
- Ability to connect internal test servers
- High scalability and individual configurations
- Suitable for automated and manual testing
- Server hosting only takes place in Germany

Architecture and Technology

In most cases, TestChameleon™ is used as Software-as-a-Service solution. There is also the possibility to install it in on-premises in companies computer centres.

Tests can be created either manually by using the web frontend or automated via APIs. The entire management happens by using a controller. The web interface functions as a relatively simple frontend. To raise capacities, only further VM hosts are added, whose allocation happens by using a corresponding algorithm. When installing TestChameleon™ on-premise, there are several drivers for virtualization solution such as Libvirt/KVM or VMware.

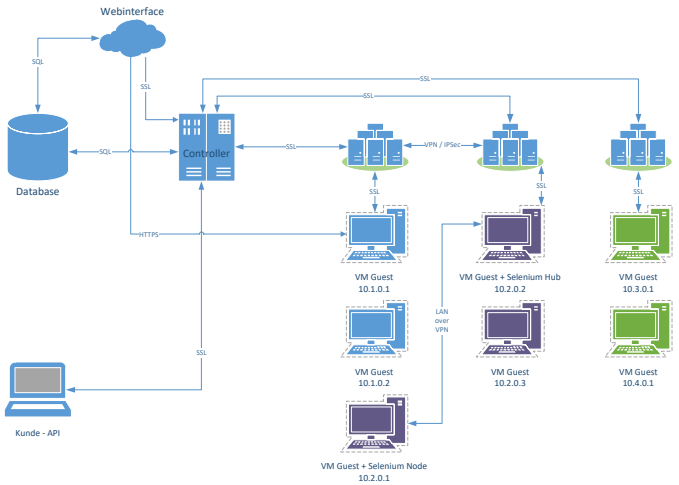


Figure 6: TestChameleon™ Architecture

Testing Process and Execution

The following section of this whitepaper shows how TestChameleon™ can be used for manual testing. By using the web frontend, individual VMs can be created. By creating an account at nest.testbirds.com it is possible to create VMs and control them via a web interface.

With just a few clicks a single individual VM can be created. In doing so, the operating system, browser and further software needs to be selected (see figure 7).

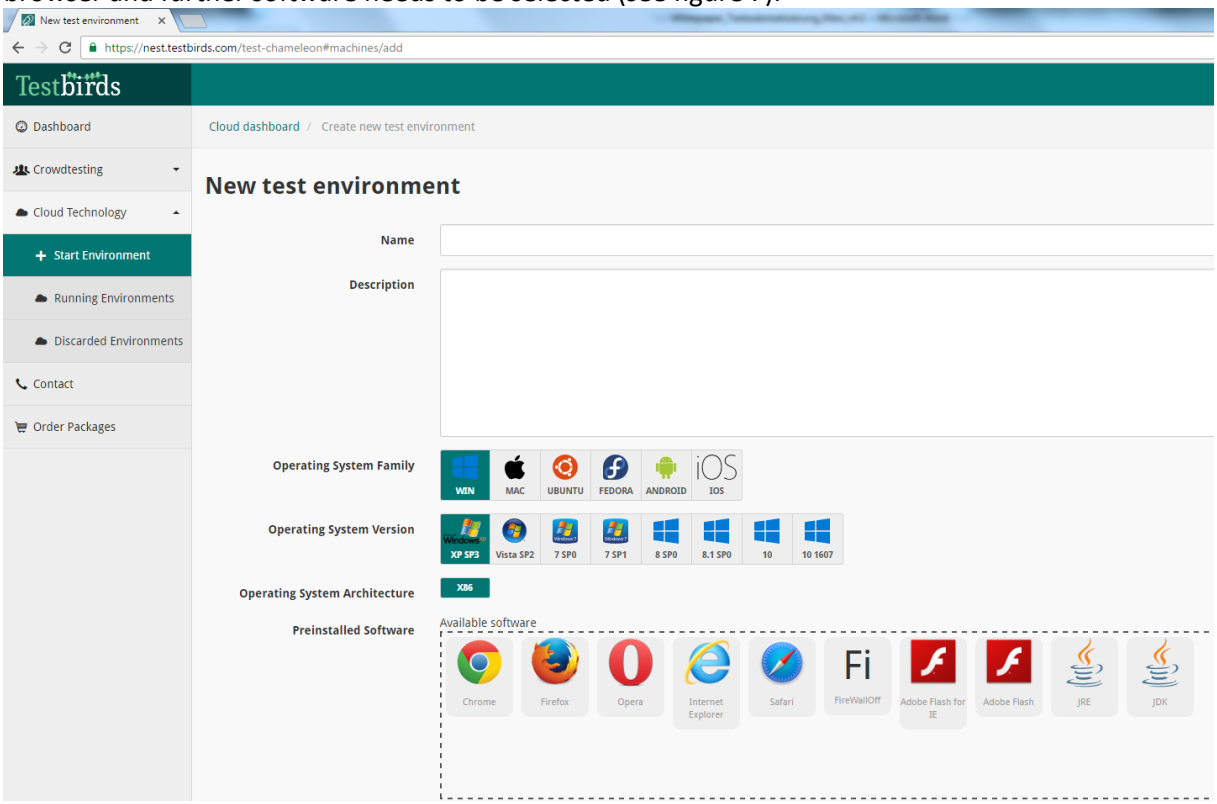


Figure 7: Configurator for VM Creation

The environment is then available within just a few minutes. It can be accessed directly in the browser via WebVNC (clean JavaScript, no plugin installation) or by using an external VNC Client (see figure 8).

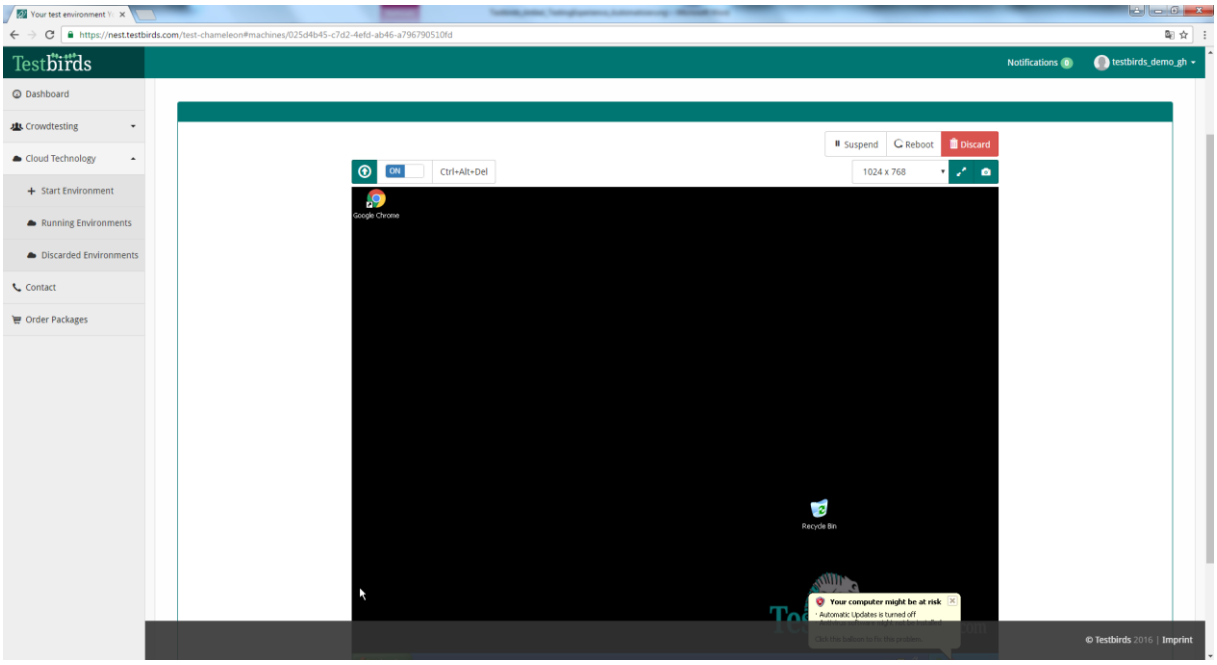


Figure 8: Access to the Test Environment via WebVNC

Depending on the operating system and device type, there is also a broad range of functions available through a tool bar.

For desktop environments those functions are:

- Restart and Pause the environment
- Upload local files
- Share the browser clipboard with the test environment
- Resolution adjustment
- Fullscreen mode
- Save a screenshot

Figure 9: Toolbar for Desktop Environments

For mobile devices the following functions are available:

- App installation
- Fullscreen mode
- Save a screenshot in original resolution
- Device rotation (portrait and landscape)
- Activation of single hardware buttons such as the home button

Figure 10: Toolbar for Mobile Devices

4. Practicle Example

4.1. Local Testing with Selenium

After a brief introduction into automated testing with Selenium, this section will present a concrete example of automated testing through displaying how test cases can be migrated from a local browser through to TestChameleon™. It is possible to increase the coverage of various platforms and browsers without major changes to the testing scripts taking place.

All examples of code that are presented in this whitepaper can be found on Github. This code can be tested on a local browser without having access to a dedicated TestChameleon™ account. The object being tested is a web application that has been created for demonstration purposes.

The following excerpts of code show the structure of a Selenium test script in Java. They can be translated into other programming languages without too much effort. To start testing, a so called WebDriver is needed. When testing on a locally installed Chrome browser, the WebDriver can be created in the following manner:

```
WebDriver driver = new ChromeDriver();
```

The object `driver` is now being used to remotely control the browser. As a second step, it is often useful to open a URL in said browser:

```
driver.get("https://demo.testchameleon.com");
```

The result is directly visible: A browser window opens up and the designated page begins to load. At the beginning, certain elements are usually selected that should be tested:

```
WebElement button = driver.findElement(By.id("submit"));
WebElement input = driver.findElement(By.id("username"));
List<WebElement> es = driver.findElements(By.className("user-profile"));
```

Further options to pick specific elements are among others: `By.tagName`, `By.cssSelector` and `By.xpath`.

This selection allows interaction with said elements. For example, it is possible to perform mouse clicks and key board actions with:

```
button.click();
input.sendKeys("username");
```

A full test of a login form could look like this:

```
driver.get(URL + "/login.html");
driver.findElement(By.id("username")).sendKeys("username");
driver.findElement(By.id("password")).sendKeys("password");
driver.findElement(By.id("submit")).click();
```

```
Assert.assertThat(driver.getCurrentURL(), endsWith("/dashboard.html"));
```

Additionally, there are enhanced WebDriver interfaces available for Appium such as AndroidDriver.

4.2. Automated Testing with TestChameleon™

TestChameleon™ provides a Selenium Grid, which enables the same interaction as with a grid created by oneself. The only difference is that the existing infrastructure automatically creates virtual machines with the desired configurations in the background, on which the test is performed.

In the Grid, and in the background of TestChameleon™, WebDrivers can be requested with DesiredCapabilites. By doing so, it is possible to define the system that should be tested in arbitrary accuracy. DesiredCapabilities look as follows:

```
DesiredCapabilities dc1 = DesiredCapabilities.chrome();
DesiredCapabilities dc2 = DesiredCapabilities.firefox();
dc2.setPlatform(Platform.WIN);
dc2.setJavaScriptEnabled(true);
…
```

A full list of such DesiredCapabilities can be found here:
https://github.com/SeleniumHQ/selenium/wiki/DesiredCapabilities

After the DesiredCapabilities are defined, the Hub requests a RemoteWebDriver that matches those exact requirements:

```
RemoteWebDriver driver = new RemoteWebDriver(HUB_URL, dc2);
```

In the background, TestChameleon™ starts a virtual machine that satisfies the exact requirements desired. Usually, within only a few minutes, a `RemoteWebDriver` is then accessible for interaction.

Further information and code examples can be found in the TestChameleon™ documentary:
https://confluence.testbirds.com/display/TED

5. Summary

Due to an increase in agile development, a fast, efficient and extensive Quality Assurance process is a crucial factor for the success of software. Therefore, it is important to continuously test both existing components as well as new features, which can quickly amount to a high level of expenditure. For this reason, a combination of manual as well as automated testing is often recommended. While new features are easier and faster to test manually, automation is a resource and cost saving alternative in the long term.

Although the setup for automated testing processes and the creation of the necessary scripts initially takes a huge amount of effort, this approach is still worthwhile, particularly when looking at regression testing core functions. Selenium for desktop application sand Appium for mobile browsers and native apps offer the options to test all components and functions accordingly.

Exactly like in a manual testing environment, the increasing variety of device, software and operating system combinations is a serious challenge, even when looking at automated testing. As previously proven, the SaaS solution, TestChameleon™, grants the ability to simultaneously test a system on a huge variety of virtual machines via existing frameworks such as Selenium or Appium.

While practical cases for getting started with Selenium are available on Github, the additional usage of virtual machines with TestChameleon™ can also be tested for free in a 30-day trial after registering to Testbirds' platform.

TestChameleon™ 30 Days Demo

Feel free to contact us for further questions or remarks concerning the content of this whitepaper. We are happy to get in touch with you via e-mail info@testbirds.com or via phone +44 203 129 5012.

6. Glossary

Agile Software Development

- The classic waterfall model with rigid development and testing phases is more frequently being replaced with agile methods. Short iterations, often called sprints, lead to frequent and short testing phases.

Appium

- Appium is an extension of selenium designed for testing mobile apps and websites on Android and iOS. It enables to perform Selenium tests in mobile browers as well as the testing of native apps.

CI Systems

- These systems are designed to support Continuous Integration. This includes for example Jenkins, Cruise Control, TeamCity, Bamboo and Gitlab CI.

Continuous Integration

- The continuous rebuilding and automated testing of software is central to this approach. In parallel to development, testing also takes places on a regular basis, (also known as Continuous Testing). Results are accessible to all parties involved.

DevOps

- DevOps describes the collaboration between software developers (Development) and IT businesses (Operations).

Jubula

- With this test suite, tests can be created without any programming knowledge. Jubula is an alternative to tools such as Selenium or Appium.

Regression Test

- After a bugtest, QA managers use regression tests to make sure that the bug removal process did not interfere with other functioning components in other parts of the system.

Selenium

- Selenium is a framework to automate tests of websites and mobile apps. It simulates user activities and therefore remotely controls browsers.

Software-as-a-Service (SaaS)

- Software-as-a-Service refers to applications that exist in the cloud. The client can access software online and does not have to install it locally.

Test Environment

- This is the environment in which software is tested. With TestChameleon™ this environment is located in the cloud.

UI Test

- A UI test checks the User Interface. As this tends to be time consuming, these kind of tests are often automated when possible.

VM

- A VM, short for virtual machine, enables the usage of devices, operating systems, browsers and other software in a virtual environment instead of through a purely physical basis.