Devonfw Testing Guide

2018-02-04

Table of Contents

1. Basics

1.1 Home

1.1.1 What is E2E Allure Test Framework

End to end automation test framework written in Java.

E2E Test Framework for DevOps & Smart Automation
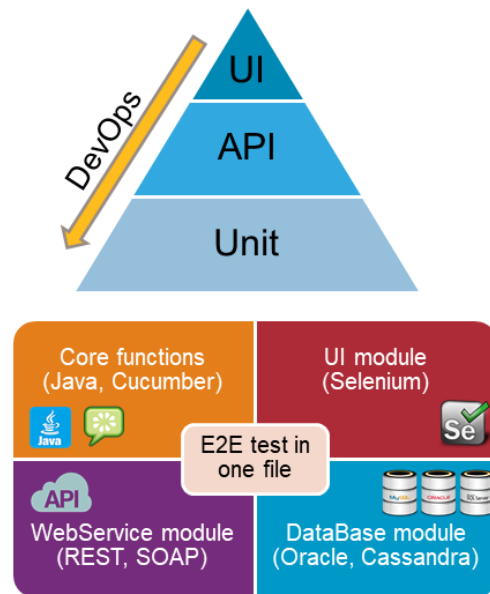
Task Description

- Embrace Test Automation in project with minimal cost and time
- Bring up to speed testers and developers with common quality goal
- Plug in/out framework features to fit project needs
- One test case to validate 3 tiers of app (Database, WebService, UI)
- Accelerate tests execution (parallel execution, env auto-scaling)
- Standardize test develop process

Service provided by Capgemini

- Takeover of operation and therefore delivery of the entire service in accordance with DevOps procedural model.
- Capgemini is partnering with client on implementation of IT Projects Delivery Transformation
- Tool agnostic approach / Innovative architecture – Test Framework with examples
- Test execution infrastructure (Jenkins as code pipeline, Docker images)
- Scalable and resilient UI web browser environment (Selenium Grid)

Benefit for the Customer

- Resilient and robust building and validation process
- Shift quality gates closer to software developer process
- Team quality awareness increase - include Unit Tests, Static Analyze, Security Tests, Performance
- Transparent test execution environment to any infrastructure. Touch base with Cloud solution
- Faster Quality and DevOps driven delivery
- Proven frameworks, technologies and processes

UI
API
Unit

DevOps

Core functions
(Java, Cucumber)

UI module
(Selenium)

E2E test in
one file

WebService module
(REST, SOAP)

DataBase module
(Oracle, Cassandra)

E2E Test Framework modules

UI

UI (Selenium) module
Selenium Grid
Mobile (Appium) app *
Mobile Grid *
Standalone APP *

Security module

API

WebAPI module

DevOps module

Test core module

BACKEND

Database module

* - task not started, to be defined

1.1.2 Benefits to the project

Benefits to project

End to end project verification under one solution

Resilient and robust DevOps driven delivery

Team quality awareness

Shift left quality gates

Easy to enhance

1.1.3 Road map plan

1.1.4 Wiki Structure:

- How to install *Allure Test Framework*

- Allure Test Framework modules:

  - Core test module

  - Selenium test module

  - WebAPI test module

  - Security test module

  - DataBase test module

  - Mobile test module

  - Standalone test module

  - DevOps module

1.2 How to install

1.2.1 Install can be done in two scenarios:

- Out of the box install - Fast and easy

- Advanced installation - How to install all ingredients

Easy out of the box install

1. Java 1.8 JDK 64bit

    - Download and install Java download link

- Windlows Local Environment how to set:

    - **Variable name**: JAVA_HOME | **Variable value**: c:\Where_You've_Installed_Java

    - **Variable name**: PATH | **Variable value**: %JAVA_HOME%/bin;%JAVA_HOME%\lib

- Verify in command line:

```
> java --version
```

2. Download package [Ready to use AllureTestEnvironment](#)

3. To folder C:\ unzip with [7z](#) downloaded Allure Test Framework

   Note: Please double check, place where you have unzipped Allure_Test_Framework

4. In unzipped folder (C:\Allure_Test_Framework\) run *start-eclipse.bat*

5. . Update project structure (*ALT + F5*)

Manual step by step install

1. Java 1.8 JDK 64bit

   • Download and install Java download link

Java SE Development Kit 8u131

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

○ Accept License Agreement   ◉ Decline License Agreement

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.87 MB	jdk-8u131-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.81 MB	jdk-8u131-linux-arm64-vfp-hflt.tar.gz
Linux x86	164.66 MB	jdk-8u131-linux-i586.rpm
Linux x86	179.39 MB	jdk-8u131-linux-i586.tar.gz
Linux x64	162.11 MB	jdk-8u131-linux-x64.rpm
Linux x64	176.95 MB	jdk-8u131-linux-x64.tar.gz
Mac OS X	226.57 MB	jdk-8u131-macosx-x64.dmg
Solaris SPARC 64-bit	139.79 MB	jdk-8u131-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.13 MB	jdk-8u131-solaris-sparcv9.tar.gz
Solaris x64	140.51 MB	jdk-8u131-solaris-x64.tar.Z
Solaris x64	96.96 MB	jdk-8u131-solaris-x64.tar.gz
Windows x86	191.22 MB	jdk-8u131-windows-i586.exe
Windows x64	198.03 MB	jdk-8u131-windows-x64.exe

- Windows Local Environment how to set:

  - **Variable name**: JAVA_HOME | **Variable value**: c:\Where_You've_Installed_Java

  - **Variable name**: PATH | **Variable value**: %JAVA_HOME%/bin;%JAVA_HOME%\lib

- Verify in command line:

```
> java --version
```

2. Maven 3.5

- Download Maven http://www-eu.apache.org/dist/maven/maven-3/3.5.0/binaries/apache-maven-3.5.0-bin.zip

- Unzip, to C:\maven

- Windows Local Environment

  - **Variable name**: M2_HOME | **Variable value**: c:\maven

  - **Variable name**: PATH | **Variable value**: %M2_HOME%\bin

- Verify in command line:

```
> mvn --version
```

3. Eclipse IDE

   - Download and unzip [Eclipse](#)

4. Download Allure Test Framework [source code](#)

5. Import projects in Eclipse

   - Import:

- Projects from folders:

- Open already created projects
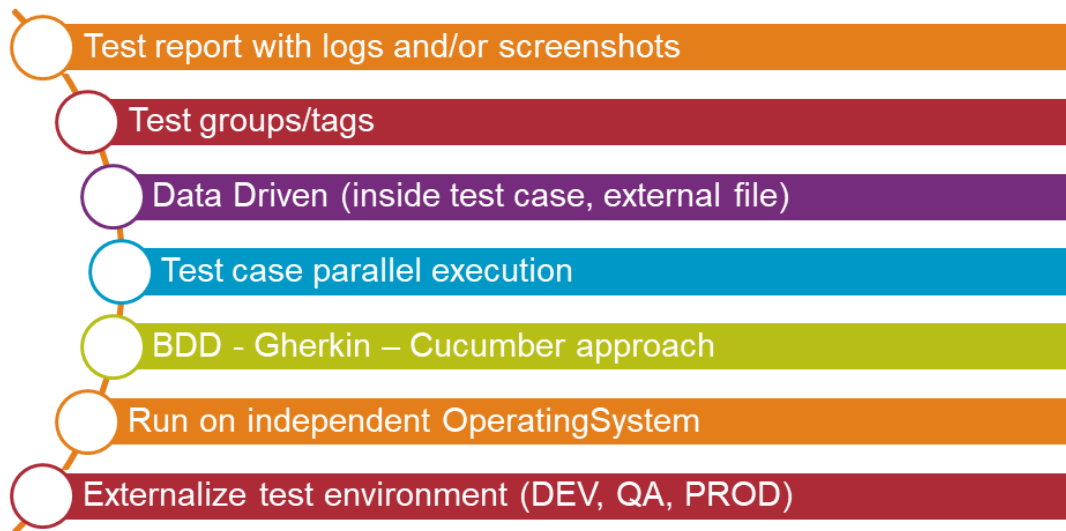
- Update project structure - *ALT + F5*

2. Modules

2.1 Core Test Module

2.1.1 What is Core Test Module

Core functionality ingredients

- Test report with logs and/or screenshots
- Test groups/tags
- Data Driven (inside test case, external file)
- Test case parallel execution
- BDD - Gherkin – Cucumber approach
- Run on independent OperatingSystem
- Externalize test environment (DEV, QA, PROD)

2.1.2 Core Test Module Functions

- Test reports with logs and/or screenshots

- Test groups/tags

- Data driven approach

- Test case parallel execution

- BDD - Gherkin - Cucumber approach

- Run on independent Operating Systems
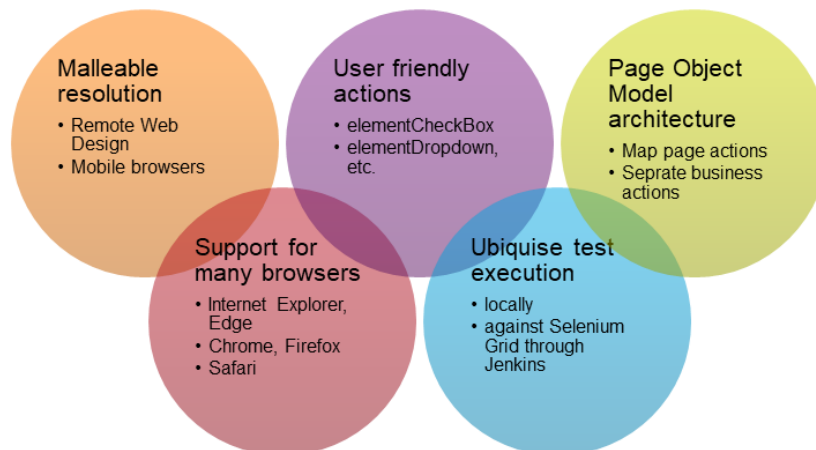
- Externalize test environment (DEV, QA, SIT, PROD)

[[core-test-module_how-to-start?]] == How to start? Read: Framework Test Class

Unresolved directive in DevonfwTesting.asciidoc - include::framework-test-class.asciidoc[]

2.2 Selenium Test Module

2.2.1 What is Allure E2E Selenium Test Module

UI Selenium test module ingredients

2.2.2 Selenium Structure

- [What is Selenium](#)

- [What is WebDriver](#)

- [What is Page Object Model/Pattern](#)

- [List of web elements (Button, Dropdown, Checkbox, Alert Popup, etc.)](#)

2.2.3 Framework Features

- [Construction of Framework Page Class](#)

  - Every Page class must extend BasePage

  - What is isLoaded(), load() and pageTitle() for

  - How to create selector variable - 'private static final By ButtonOkSelector = By.Css(…)'

  - How to prepare everlasting selector - [documentation](#)

  - Method/action naming convention - [documentation](#)

  - Why we should use findDynamicElement() and findElementQuietly() instead of classic Selenium findElement

- List of well-rounded groups of user friendly actions (ElementButton, ElementCheckbox, ElementInput, etc.)

- Verification points of well-defined Page classes and Test classes - documentation

- Run on different browsers: Chrome, Firefox, IE, Safari, Edge

- Run with full range of resolution (mobile and desktop): Testing Response Design Webpage

[[selenium-test-module_how-to-start?]] == How to start? Read: My first Selenium Test

Unresolved directive in DevonfwTesting.asciidoc - include::Building-basic-Selenium-test.asciidoc[]

WebAPI Test Module

Security Test Module

[[security-test-module_what-is-security?]] == What is Security?

Application Security is concerned with **Integrity**, **Availability** and **Confidentiality** of data processed, stored and transferred by the application.

Application Security is a cross-cutting concern which touches every aspect of the Software Development Lifecycle. You can introduce some SQL injection flaws in your application and make it exploitable, but you can also expose your secrets due to poor secret management process (which will have nothing to do with code itself), and fail as well.

Because of this, and many other reasons, not every aspect of security can be automatically verified. Manual tests and audits will be still needed. Nevertheless, every security requirement which are automatically verified, will prevent code degeneration and misconfiguration in a continuous manner.

[[security-test-module_how-to-test-security?]] == How to test Security?

Security tests can be performed in many different ways like:

- **Static Code Analysis** - improves the security by (usually) automated code review. Good way to search after vulnerabilities, which are 'obvious' on the code level (like e.g. SQL injection). The downside is that the professional tools to perform such scans are very expensive and still produce many false positives.

- **Dynamic Code Analysis** - tests are run against a working environment. Good way to search after vulnerabilities, which require all client- and server-side components to be present and running (like e.g. Cross-Site Scripting). Tests are performed in a semi-automated manner and require a proxy tool (like e.g. OWASP ZAP)

- **Unit tests** - self written and maintained tests. They work usually on the HTTP/REST level (as this defines the trust boundary between the client and the server) and run against a working environment. Unit tests are best suited to verify requirements which involve business knowledge of the system or which assure secure configuration on the HTTP level.

In the current release of the Security Module the main focus will be **Unit Tests**.

Although the most common choice of environment for security tests to run on will be **integration** (as the environment offers the right stability and should mirror the production closely), it is not uncommon for some security tests to run on production as well. This is done for e.g. TLS configuration testing to ensure proper configuration of the most relevant environment in a continuous manner.

Scope definition

Unresolved directive in DevonfwTesting.asciidoc - include::Database-test-module.asciidoc[]

17

Mobile Test Module

Standalone Test Module

DevOps Module

How we see DevOps

DevOps consists of a mix of three key components in a technical project: * People skills and mind set * Processes * Tools

In **E2E Allure Test Framework** we would like to address majority of these components.

QA Team Goal

As QA engineers we always take a lot of care about product code quality.

Therefore, we also have to understand, **test case is also a code which has to be validated** against quality gates. As a result, we must **test our developed test case** just as it was done during standard Software Delivery Life Cycle.

Well rounded test case production process

- How we define top notch test cases develop process in **E2E Allure Test Framework**

Well defined Test Case develop process
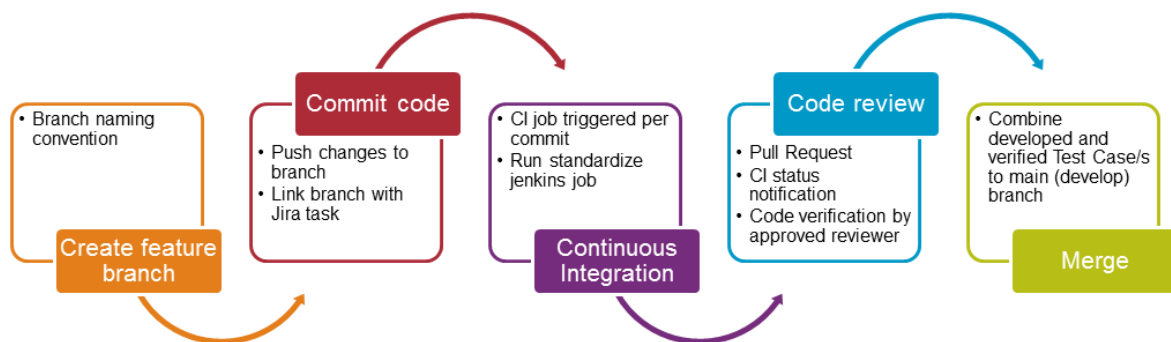
[[devops-module_continuous-integration-(ci)-and-continuous-delivery-(cd)]] == Continuous Integration (CI) and Continuous Delivery (CD)

- Continuous Integration (CI) - procedure where quality gates validate test case creation process

- Continuous Delivery (CD) - procedure where we include as smoke/regression/security created test cases, validated against CI
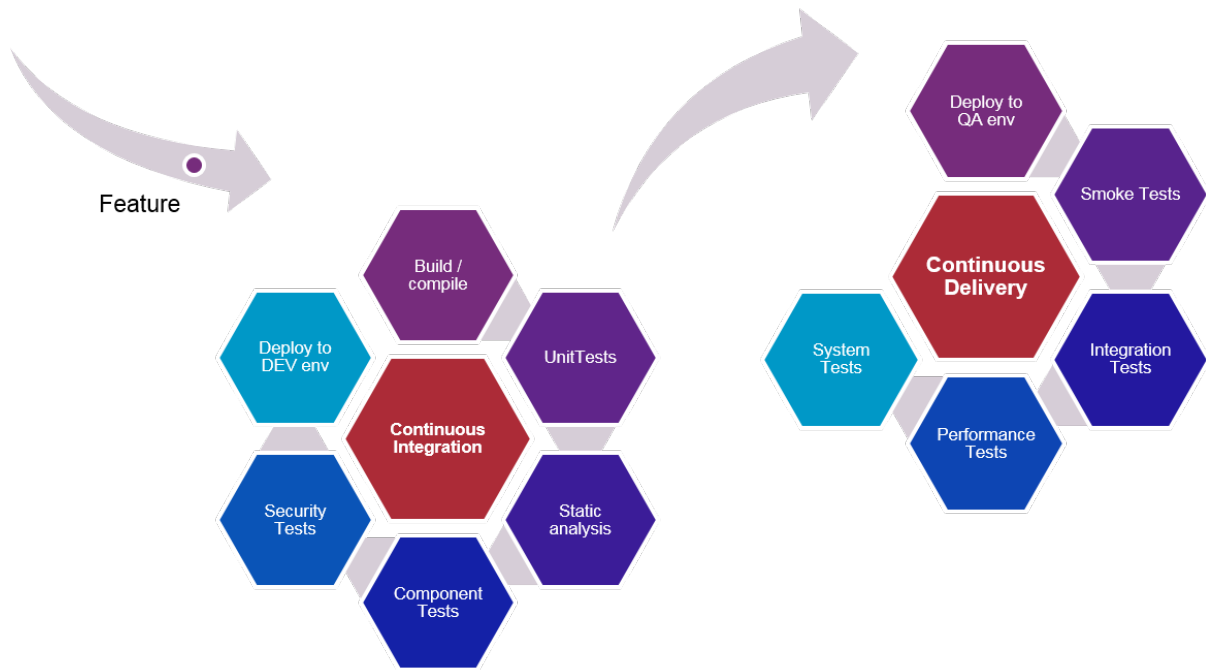
What will you receive in this DevOps module

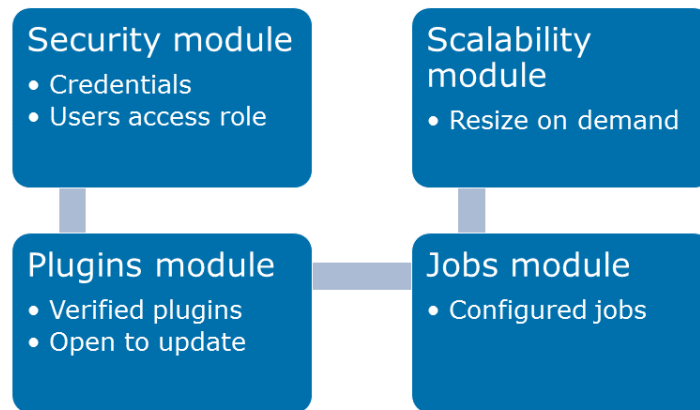# DevOps infrastructure ingredients
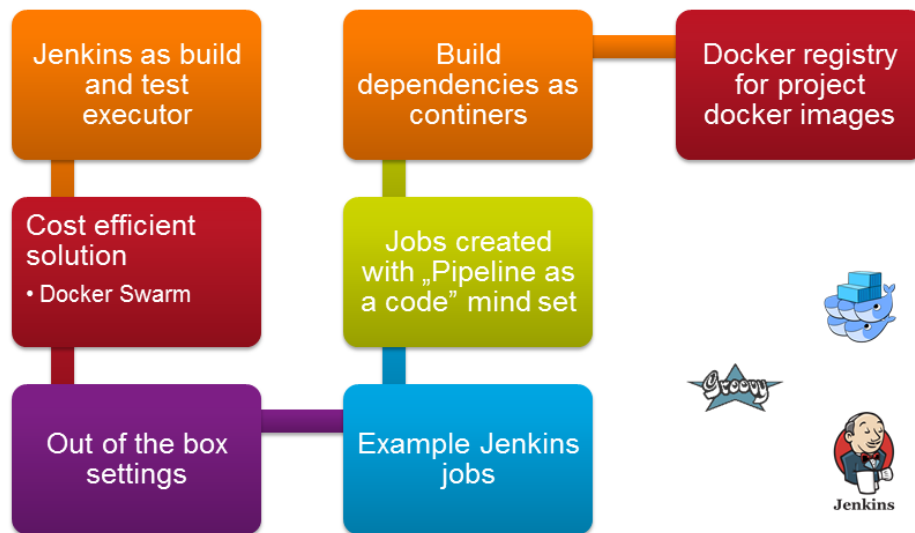
What will you gain with our DevOps module

The CI procedure has been divided into transparent modules. This solution makes configuration and maintenance very easy because you can manage versions and customize the configuration
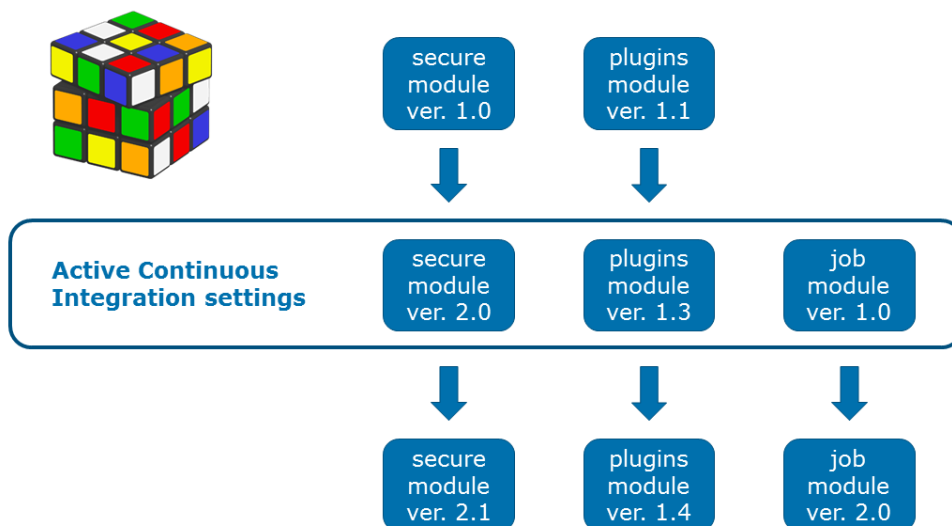
Superior Continuous Integration ingredients

Your CI process will be matched to the current project. You can easily go back to the previous configuration, test a new one or move a selected one to other projects.

Setup your own proven CI modules

DevOps module supports delivery model in which executors are made available to the user as needed. It has advantages such as: * Save computing resources * Eliminate guessing on your infrastructure capacity needs * Stop spending time on running and maintaining additional executors

Classic executor architecture for Continuous Integration
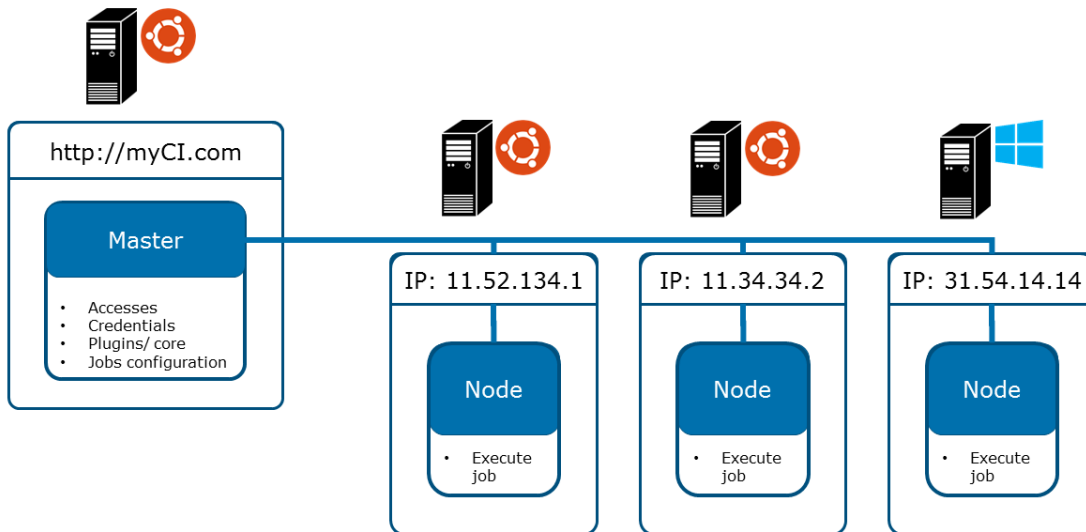
Innovative executor architecture for Continuous Integration
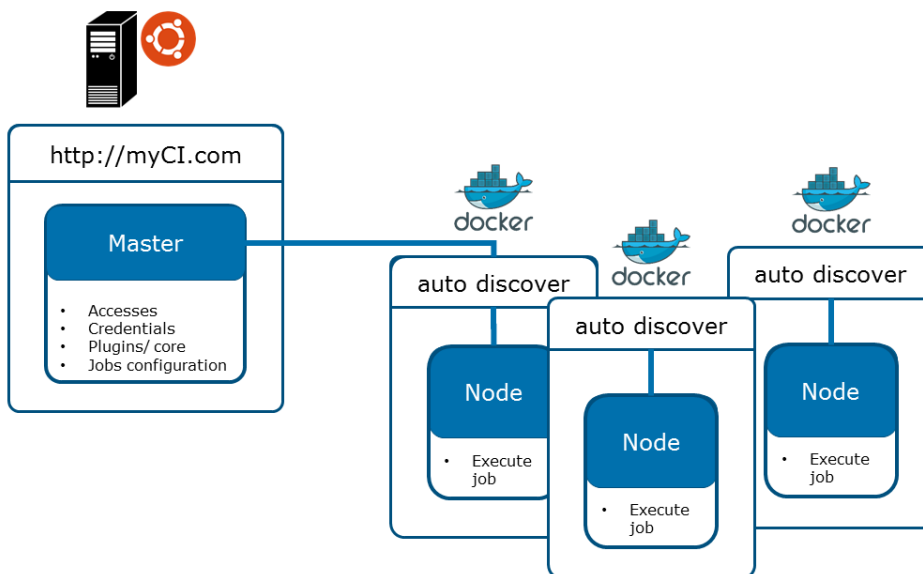
Benefits

Modularity
Build your own CI

Secure
Restricted access to credentials

Autoscaling and service discovery
Take as you need

How to build this DevOps module

If you want to install the module, please click the link below. Installation should not take more than a few minutes * DevOps module installation

Once you have implemented the module, you can learn more about:

• Building jobs & Running builds

• Docker commands

Continuous Integration

Embrace quality with Continuous Integration while you produce test case/s.

Overview

There are two ways to set up your Continuous Integration environment:

1. Create a Jenkins instance from scratch (e.g. by using the Jenkins Docker image)

   Using a clean Jenkins instance requires the installation of additional plugins. The plugins required and their versions can be found on [this page](#).

2. Use thre pre-configured custom Docker image provided by us

   No more additional configurations is required (but optional) using this custom Docker image. Additionally, this Jenkins setup allows to be dynamically scaled across multiple machines and even the cloud (AWS, Azure, Google Cloud etc.).

Jenkins Overview
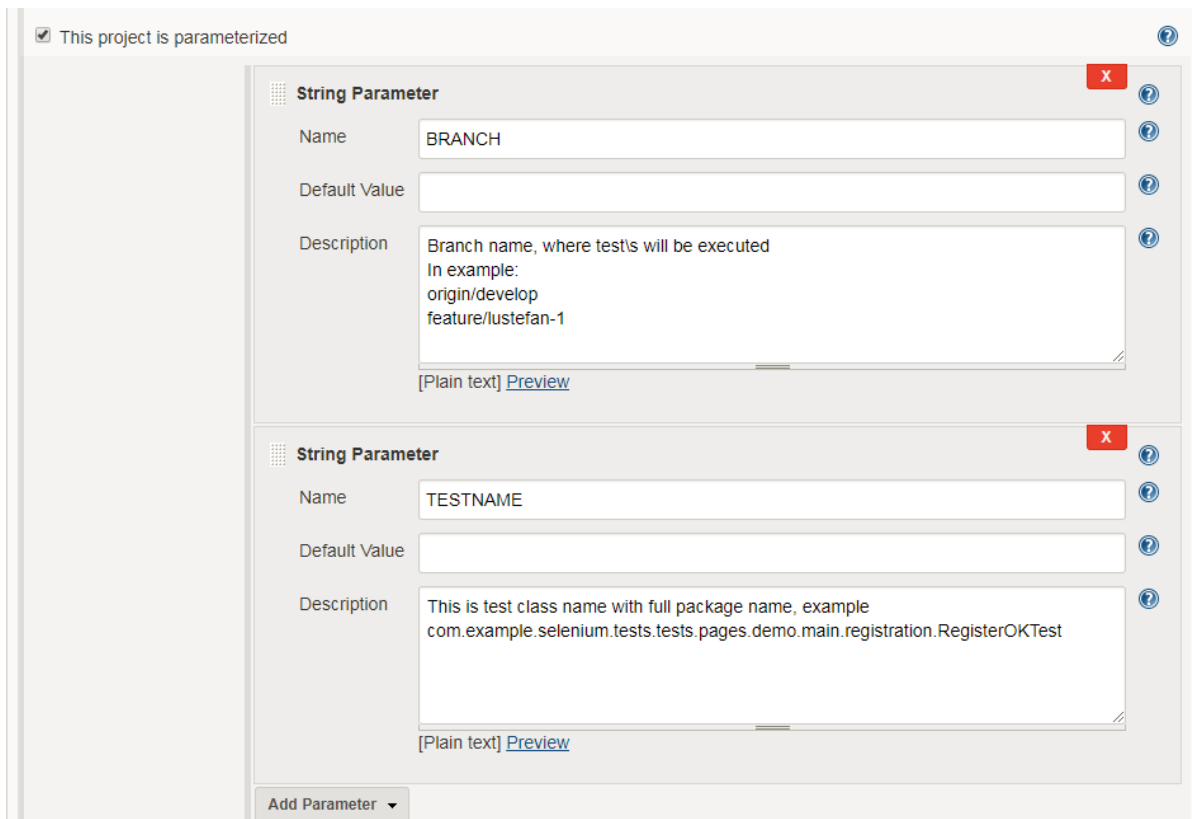
Jenkins is an Open Source Continuous Integration Tool. It allows the user to create automated build jobs which will run remotely on so called *Jenkins Slaves*. A build job can be triggered by several events, for example on new pull request on specified repositories or timed (e.g. at midnight).

Jenkins Configuration

Tests created by using the testing framework can be easily implemented on a Jenkins instance. The following chapter will describe such a job configuration. If you're running your own Jenkins instance you may have to install additional plugins listed on the page [Jenkins Plugins](#) for a trouble-free integration of your tests.
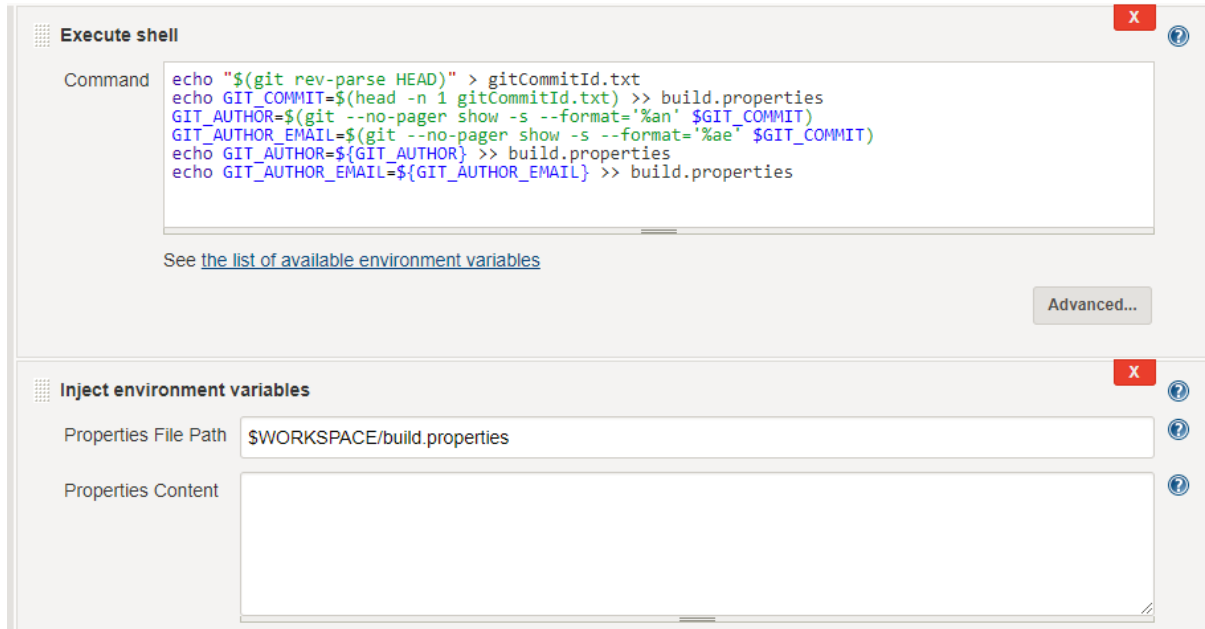
Initial Configuration

The test job is configured as a so-called *parametrized* job. This means, after starting the job, parameters can be specified, which will then be used in the build process. In this case, *branch* and *testname* will be expected when starting the job. These parameters specify which branch in the code repository should be checked out (possibly feature branch) and the name of the test, that should be executed.

Build Process Configuration

- The first step inside the build process configuration is to get the author of the commit that was made. The mail will be extracted and gets stored in a file called *build.properties*. This way, the author can be notified if the build fails.

Execute shell

Command
```
echo "$(git rev-parse HEAD)" > gitCommitId.txt
echo GIT_COMMIT=$(head -n 1 gitCommitId.txt) >> build.properties
GIT_AUTHOR=$(git --no-pager show -s --format='%an' $GIT_COMMIT)
GIT_AUTHOR_EMAIL=$(git --no-pager show -s --format='%ae' $GIT_COMMIT)
echo GIT_AUTHOR=${GIT_AUTHOR} >> build.properties
echo GIT_AUTHOR_EMAIL=${GIT_AUTHOR_EMAIL} >> build.properties
```
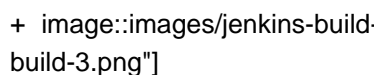
See the list of available environment variables

Advanced...

Inject environment variables

Properties File Path $WORKSPACE/build.properties

Properties Content

- Next up, Maven will be used to check if the code can be compiled, without running any tests.

Invoke top-level Maven targets

Maven Version   v3.3.9

Goals   clean install test-compile -DskipTests=true

Advanced...

After making sure that the code can be compiled, the actual tests will be executed.

+ image::images/jenkins-build-3.png["Starting the actual tests", width="450", link="images/jenkins-build-3.png"]
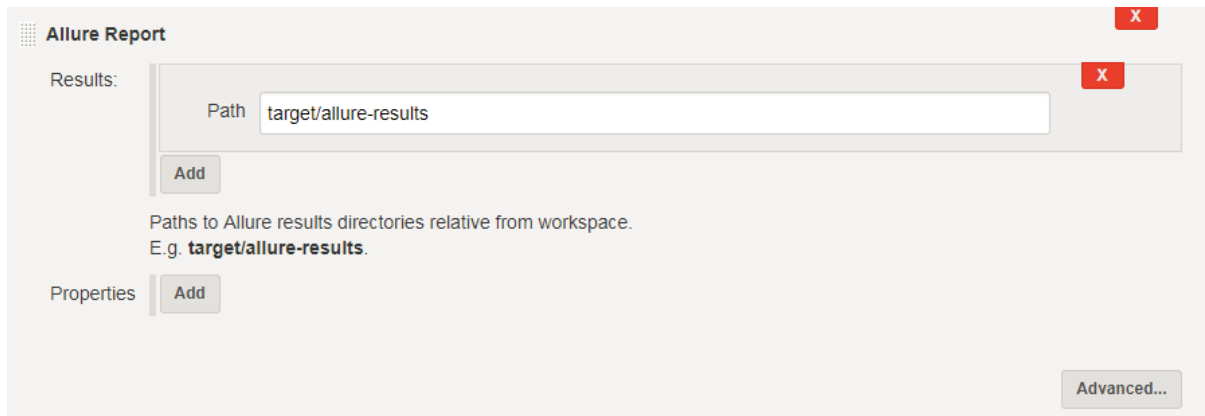
- Finally, reports will be generated.

Invoke top-level Maven targets

Maven Version   v3.3.9

Goals   site

Advanced...

Post Build Configuration

- At first, the results will be imported to the Allure System
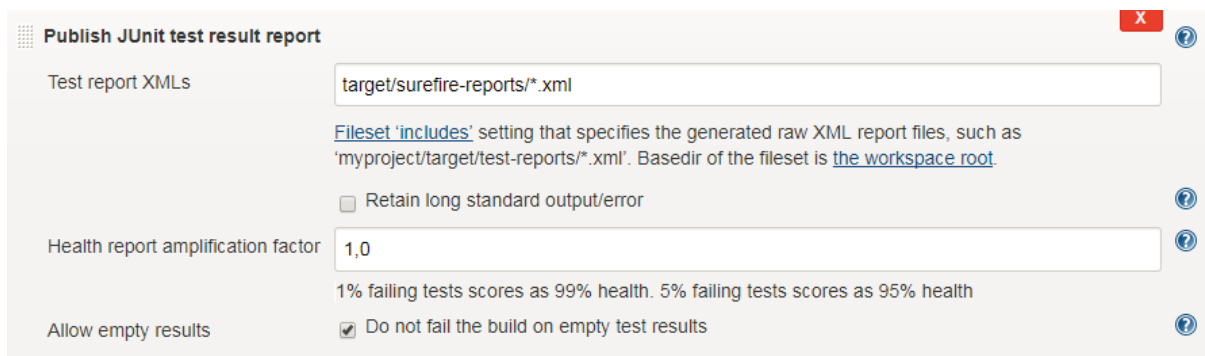
- JUnit test results will be reported as well. Using this step, the test result trend graph will be displayed on the Jenkins job overview.
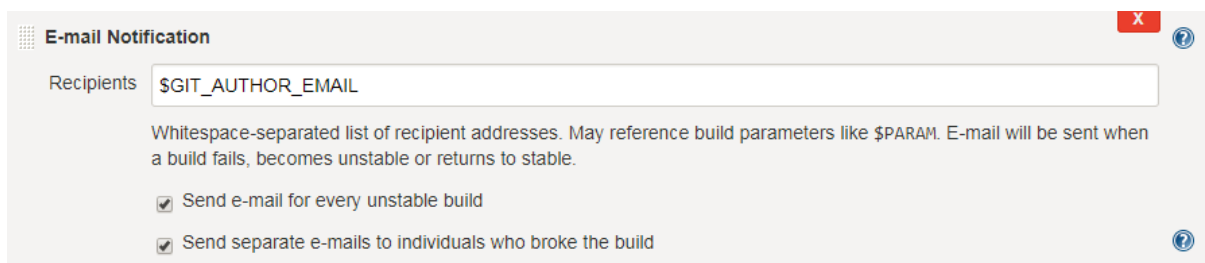
- Finally, an E-Mail will be sent to the previously extracted author of the commit.
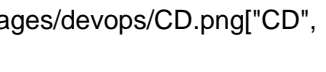
Using the Pre-Configured Custom Docker Image

If you are starting a new Jenkins instance for your tests, we'd suggest to use the pre-configured Docker image. This image already contains all configurations and additional features.

The configurations that are made are e.g. Plugins and Pre-Installed job setup samples. This way, you don't have to set up the entire CI-Environment from ground up.

The additional features from this docker image allow the dynamic creation and deletion of Jenkins slaves, by creating Docker containers. Also, Cloud Solutions can be implemented to allow wide-spread load balancing.

Continuous Delivery

Include quality with Continuous Delivery during product release. image::images/devops/CD.png["CD", width="450", link="image/devops/CD.png"]

Overview

CD from Jenkins point of view does not change a lot from Continuous Integration one.

Jenkins Overview

For Jenkins CD setup please use the same Jenkins settings as for CI link The only difference is:

• What type of test you we will execute. Before we have been picking test case(s), however now we will choose test suite(s)

• Who will will trigger given Smoke/Integration/Performance job

• What is the name of official branch. This branch ought to use be used always in every CD execution. It will be either **master**, or **develop**.

Jenkins for Smoke Tests

In point where we input test name - $TESTNAME (link), please input test suite which merge by tags -(link) how to all test cases need to run only smoke tests.
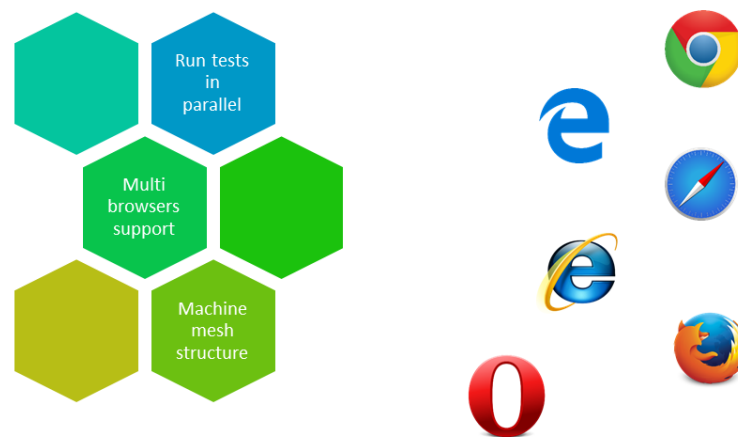
Jenkins for Performance Tests

Under construction - added when WebAPI module is included.

Selenium Grid

What is Selenium Grid

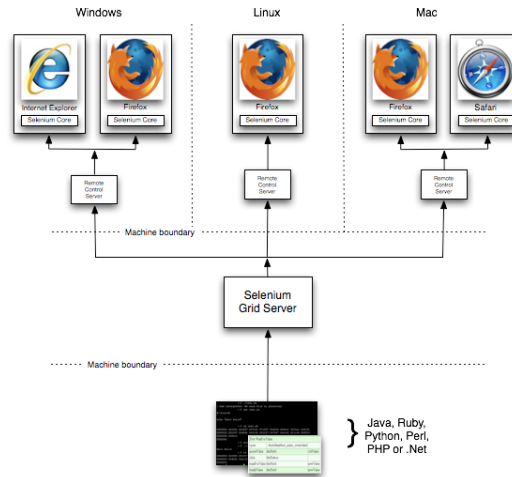Selenium Grid – where we use it

Allows to run web/mobile browsers test cases to full fill bedrock factors, such as: * Independence infrastructure , similar to end users * Scalable infrastructure (\~50 simultaneous sessions at once) * Huge variety of web browsers (from mobile to desktop) * Continuous Integration and Continuous Delivery process * Support multi type programming languages (java, javascript, python, …)

In daily base test case execution/develop test automation engineer uses his local environments. However created browser test case must be able to run on any other infrastructure. Therefore Selenium Grid allows us this portability.

Selenium Grid Structure

Selenium Grid - structure

Windows | Linux | Mac

Internet Explorer | Firefox | Firefox | Firefox | Safari
Selenium Core

Remote Control Server

Machine boundary

Selenium Grid Server

Machine boundary

} Java, Ruby, Python, Perl, PHP or .Net

Full documentation for Selenium Grid can be found here and here.

As vanila flavor Selenium Grid is based on two, not to much complicated, ingredients:

1. **Selenium Hub** - as a one machine, accepts connections to grid from test cases executors. Moreover it plays managerial role in connection to/from Selenium Nodes

2. **Selenium Node** - from one to many machines, where on each machine is installed browser used during test case execution.

How to setup

There are two solutions:

- Classic, static solution - link

- Cloud, scalable solution - link

Pros and cons of both solutions:

Single Selenium Grid - comparison

	Cost to setup (20 instances/browsers)	Cost to scale up (down) new 20 instances/browsers	Team responsibility	Resilient and robust	Portability
Classic solution	• 200 $ /month (VMs)	• + 20 $ /month (VM) + 4 hour SeleniumGrid specialist	• Works as centralized solution used across all Test departments	• No replication. Manual actions needed to recover	• Manual infrastructure setup, for each browser (Chrome, Firefox, IE, Safari, Edge)
Cloud solution	• 10 $ /month (VMs)	• + 10 $ /month (VMs) + 0.25 hour junior team member	• Works as centralized per department or decentralized solution per team	• Auto recovery. Balance number of active instances/ browsers through swarm of active VMs	• Auto deploy script for Chrome and Firefox. Open interface to add manually IE, Safari, Edge

*) Classic solution – selenium grid run as a Java standalone application
*) Cloud solution – selenium grid run as a Docker container

VM – Virtual Machine

[[selenium-grid_how-to-use-it-with-**e2e-allure-test-frameworks**]] == How to use it with **E2E Allure Test Frameworks**

Run this command either in Eclipse or in Jenkins:

```
> mvn test -Dtest=com.capgemini.ntc.selenium.tests.samples.resolutions.ResolutionTest -
DseleniumGrid="http://10.40.232.61:4444/wd/hub" -Dos=LINUX -Dbrowser=chrome
```

As a result of this command:

* *-Dtest=com.capgemini.ntc.selenium.features.samples.resolutions.ResolutionTest* - name of test case to execute

* *-DseleniumGrid="http://10.40.232.61:4444/wd/hub"* - IP address of Selenium Hub

* *-Dos=LINUX* - what operating system must taken during test case execution

  * *-Dbrowser=chrome* - what type of browser will be used during test case execution