Implementing Code Coverage

- Introduction
- For Maven
  - Java
    - Maven Properties
    - Surefire (Unit tests)
    - Failsafe (Integration tests)
    - Jacoco
    - Docker Plugin (Integration tests)
  - Javascript
    - Maven properties
    - frontend-maven-plugin
    - Package.json
    - Javascript test Example
    - Concrete example
  - Python
- For Gradle

Introduction

Sonar requires the coverage files to be provided during the project scan. Those data files are generally recorded by plugins, servers, interpreters, ... when executing the tests and made accessible to the sonar scanner.

This depends of the language and packager (like maven/gradle, ..) used in the project to execute the tests and build the artifacts.

Sonar recognizes 2 types of testing that can be executed in a build for the coverage report, and it computes the union of the unit and integration test results.

1. **Unit tests**
   It tests the classes or methods directly, generally by using specific test framework (Junit, testNG, etc ...)
   The "container framework" used behind the scene (Spring/ JBoss/ EJB or whatever) **SHOULD NOT** be used in the Unit test scope.
   External calls to test servers or third party components/anything **MUST NOT** be done here. These type of tests should be executable by anybody locally in an IDE.
   Due to that, it's sometimes required to make use of mock frameworks to replace the code that can't be triggered because it does require unreachable components
   The mocking **SHOULD NEVER** consume the developer's bandwidth, that means mocking the code should be straight-forward.
   Don't waste your time into a complicated implementation, for that there is a second test phase called "integration tests"

   Also **DO NOT** add "isTest" flag in your code to replace the mock, this is a common test anti pattern that leads to useless tests.

2. **Integration tests**
   It tests the code within the "container framework" and can make use of 3rd party components, like databases.
   The 3rd party components could be emulated, or even better, could be physically mounted locally, with the help of a docker engine for instance.
   The life-cycle of the those 3rd party components can be initiated/shut-downed by the build itself in a "Pre-integration/Post-integration" phase.

Here are some tips to setup that coverage per packager and per language

For Maven

When using MAVEN, at least one pom.xml must be defined, the project contains normally a central pom.xml that is used to build the entire project (it can contain multiple maven sub-modules)

In this pom.xml, the sonar plugin must be defined, the command *mvn sonar:sonar* will then execute the scan

```
<plugin>
        <groupId>org.sonarsource.scanner.maven</groupId>
        <artifactId>sonar-maven-plugin</artifactId>
          <version>x.x</version>
</plugin>
```

There is also a need to setup some properties for SONAR in order to indicate how the scans should be done, this include the coverage data files that must be used

```
<properties>
....
    <sonar.core.codeCoveragePlugin>jacoco</sonar.core.codeCoveragePlugin>
    <!-- Default Sonar configuration -->
    <sonar.coverage.jacoco.xmlReportPaths>${project.reporting.outputDirectory}/jacoco-ut/jacoco.xml</sonar.
coverage.jacoco.xmlReportPaths>

....
</properties>
```

Note that if you import the org.onap.oparent.oparent those settings are already included, so there is no need to add them unless you want to override those settings

- Java

  For java there are different plugins/properties that must be included in the pom.xml to execute the tests and record the coverage during the execution

Maven Properties

```
<properties>
...
                <sonar.java.coveragePlugin>jacoco</sonar.java.coveragePlugin>
                <sonar.surefire.reportsPath>${project.build.directory}/surefire-reports</sonar.surefire.
reportsPath>
                <sonar.coverage.jacoco.xmlReportPaths>${project.reporting.outputDirectory}/jacoco-ut
/jacoco.xml</sonar.coverage.jacoco.xmlReportPaths>
                <sonar.projectVersion>${project.version}</sonar.projectVersion>
                <!-- Enable language to disable other language analysis -->
                <sonar.language>java</sonar.language>
                <sonar.exclusions>src/main/resources/**</sonar.exclusions>
...
</properties>
```

Surefire (Unit tests)

This one is used to execute the unit tests, various config are available depending of the project need (https://maven.apache.org/surefire/maven-surefire-plugin/index.html)

```
<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>X.XX.X</version>
        <configuration>
                <forkCount>1</forkCount>
                <reuseForks>true</reuseForks>
                <useSystemClassLoader>false</useSystemClassLoader>
        </configuration>
</plugin>
```

Failsafe (Integration tests)

This one is used to execute the integration tests, various config are available depending of the project need (https://maven.apache.org/surefire/maven-failsafe-plugin/index.html)

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-failsafe-plugin</artifactId>
    <version>X.XX.X</version>
    <executions>
      <execution>
         <id>integration-tests</id>
         <goals>
           <goal>integration-test</goal>
           <goal>verify</goal>
         </goals>
         <configuration>
           <additionalClasspathElements>
              <additionalClasspathElement>${project.build.directory}/classes</additionalClasspathElement>
           </additionalClasspathElements>
           <includes>
             <include>**/*ItCase.java</include>
           </includes>
           <forkCount>1</forkCount>
           <reuseForks>false</reuseForks>
           <useSystemClassLoader>false</useSystemClassLoader>
         </configuration>
      </execution>
    </executions>
</plugin>
```

Jacoco

This one is used to record the coverage of the unit tests and integration tests, also a result merge for the developer IDE (https://www.eclemma.org/jacoco/trunk/doc/maven.html)

```
<plugin>
                                <groupId>org.jacoco</groupId>
                                <artifactId>jacoco-maven-plugin</artifactId>
                                <version>0.8.2</version>
                                <configuration>
                                        <dumpOnExit>true</dumpOnExit>
                                        <includes>
                                                <include>org.onap.*</include>
                                        </includes>
                                        <propertyName>surefireArgLine</propertyName>
                                </configuration>
                                <executions>
                                        <execution>
                                                <id>pre-unit-test</id>
                                                <goals>
                                                        <goal>prepare-agent</goal>
                                                </goals>
                                                <configuration>
                                                        <destFile>${project.build.directory}/coverage-
reports/jacoco.exec</destFile>
                                                        <!-- <append>true</append> -->
                                                </configuration>
                                        </execution>
                                        <execution>
                                                <id>pre-integration-test</id>
                                                <phase>pre-integration-test</phase>
                                                <goals>
                                                        <goal>prepare-agent</goal>
                                                </goals>
                                                <configuration>
                                                        <destFile>${project.build.directory}/coverage-
reports/jacoco-it.exec</destFile>
                                                        <!-- <append>true</append> -->
                                                </configuration>
                                        </execution>
                                        <execution>
                                                <goals>
                                                        <goal>merge</goal>
                                                </goals>
                                                <phase>post-integration-test</phase>
                                                <configuration>
                                                        <fileSets>
                                                                <fileSet implementation="org.apache.
maven.shared.model.fileset.FileSet">
                                                                        <directory>${project.build.
directory}/coverage-reports</directory>
                                                                        <includes>
                                                                                <include>*.exec</include>
                                                                        </includes>
                                                                </fileSet>
                                                        </fileSets>
                                                        <destFile>${project.build.directory}/jacoco-dev.
exec</destFile>
                                                </configuration>
                                        </execution>
                                </executions>
</plugin>
```

Note that for the integration tests, it could more complex than the following Jacoco plugin usage.

If it's a remote server or anything else the jacoco agent must be provided to the running JVM to do the coverage recording (https://www.eclemma.org/jacoco/trunk/doc/agent.html), then brought back to build folder.

Docker Plugin (Integration tests)

This plugin can be used to start containers before doing the integration tests. The following example starts Mariadb and Python before the integration tests phase.

```
<plugin>
                                <groupId>io.fabric8</groupId>
                                <artifactId>docker-maven-plugin</artifactId>
                                <version>0.26.0</version>
                                <configuration>
                                        <verbose>true</verbose>
                                        <apiVersion>1.23</apiVersion>
                                        <images>
                                                <image>
                                                        <name>library/mariadb:10.1.11</name>
                                                        <alias>mariadb</alias>
                                                        <run>
                                                                <env>
                                                                        <MYSQL_ROOT_PASSWORD>secret<
/MYSQL_ROOT_PASSWORD>
                                                                </env>
                                                                <hostname>mariadb</hostname>
                                                                <volumes>
                                                                        <bind>
                                                                                <volume>${project.
basedir}/extra/sql/:/docker-entrypoint-initdb.d</volume>
                                                                                <volume>${project.
basedir}/extra/docker/mariadb/conf1:/etc/mysql/conf.d</volume>
                                                                        </bind>
                                                                </volumes>
                                                                <wait>
                                                                        <log>socket: '/var/run/mysqld
/mysqld.sock'  port: 3306  mariadb.org binary distribution</log>
                                                                        <time>600000</time>
                                                                </wait>
                                                                <ports>
                                                                        <port>${docker.mariadb.port.
host}:3306</port>
                                                                </ports>
                                                        </run>
                                                </image>
                                                <image>
                                                        <name>library/python:2-slim</name>
                                                        <alias>python</alias>
                                                        <run>
                                                                <hostname>python</hostname>
                                                                <volumes>
                                                                        <bind>
                                                                                <volume>${project.
basedir}/src/test/resources/http-cache/:/usr/src/http-cache-app</volume>
                                                                                <volume>${project.
basedir}/src/test/resources/http-cache/example/:/usr/src/http-cache-app/data-cache</volume>
                                                                        </bind>
                                                                </volumes>
                                                                <wait>
                                                                        <tcp>
                                                                                <ports>
                                                                                        <port>8080</port>
                                                                                </ports>
                                                                                <mode>direct</mode>
                                                                        </tcp>
                                                                        <time>120000</time>
                                                                </wait>
                                                                <ports>
                                                                        <port>${docker.http-cache.port.
host}:8080</port>
                                                                </ports>
                                                                <workingDir>/usr/src/http-cache-app<
/workingDir>
                                                                <cmd>
                                                                        <shell>./start_http_cache.sh
${python.http.proxy.param} --python_proxyaddress=localhost:${docker.http-cache.port.host}</shell>
                                                                </cmd>
                                                        </run>
                                                </image>
                                </configuration>
```

```
                                <executions>
                                        <execution>
                                                <id>docker-start-for-it</id>
                                                <phase>pre-integration-test</phase>
                                                <goals>
                                                        <goal>start</goal>
                                                </goals>
                                        </execution>
                                        <execution>
                                                <id>docker-stop-for-it</id>
                                                <phase>post-integration-test</phase>
                                                <goals>
                                                        <goal>stop</goal>
                                                </goals>
                                        </execution>
                                </executions>
</plugin>
```

- ## Javascript

  For the javascript testing, the goal is to use JEST framework to execute the unit testing and record the coverage (Mocking is also possible with JEST)

  The setup is based on NPM, so it uses a maven plugin that handles that (similar idea to docker-maven-plugin in the Integration testing)

  Beforehand the Javascript code must be copied to the target folder, plus the package.json file that will be used by NPM

NOTE : For SonarCloud Users or SonarQube 7.8+, there is a need to have the node.js executable available on the build node, the frontend-maven-plugin below does install it as part of the build

```
                <testResources>
                        <!-- Copy the NPM package.json for UI javascript testing framework -->
                        <testResource>
                                <directory>src/test/javascript</directory>
                                <includes>
                                        <include>**/**.json</include>
                                </includes>
                                <filtering>true</filtering>
                                <targetPath>${project.build.directory}/my-ui</targetPath>
                        </testResource>
                        <testResource>
                                <directory>src/main/resources/META-INF/resources/designer</directory>
                                <filtering>false</filtering>
                                <targetPath>${project.build.directory}/my-ui/designer</targetPath>
                        </testResource>
                </testResources>
```

Maven properties

By default do not specify any language to SONAR so that it can scan all languages used in the project

```
                <sonar.javascript.lcov.reportPaths>${project.build.directory}/my-ui/coverage/lcov.info</sonar.
javascript.lcov.reportPaths>
                <sonar.projectVersion>${project.version}</sonar.projectVersion>
                <sonar.sources>src/main,${project.build.directory}/my-ui/designer</sonar.sources>
                <!-- DO NOT SPECIFY THAT so it scans javascript <sonar.language>java</sonar.language>-->
                 <sonar.exclusions>src/main/resources/**,**/my-ui/designer/lib/*</sonar.exclusions>
                <!-- NEW for SQ7.8 or SonarCloud - Need to have node.js executable available for analysis to
run, see frontend-maven-plugin config below for install location -->
                <sonar.nodejs.executable>${project.build.directory}/my-ui/node/node</sonar.nodejs.executable>
```

frontend-maven-plugin

This one is used to setup the required libraries and environment to execute the test under JEST framework ((https://github.com/eirslett/frontend-maven-plugin)

```
        <plugin>
                        <groupId>com.github.eirslett</groupId>
                        <artifactId>frontend-maven-plugin</artifactId>
                        <version>1.6</version>
                        <configuration>
                            <installDirectory>${project.build.directory}/my-ui</installDirectory>
                            <workingDirectory>${project.build.directory}/my-ui</workingDirectory>
                            <skip>${maven.test.skip}</skip>
                        </configuration>

                        <executions>
                            <execution>
                                <id>install_node_and_npm</id>
                                <goals>
                                    <goal>install-node-and-npm</goal>
                                </goals>
                                <phase>test</phase>
                                <configuration>
                                    <nodeVersion>v8.11.1</nodeVersion>
                                    <npmVersion>5.6.0</npmVersion>
                                </configuration>
                            </execution>
                            <execution>
                                <id>npm_install</id>
                                <goals>
                                    <goal>npm</goal>
                                </goals>
                                <phase>test</phase>
                                <configuration>
                                    <arguments>install</arguments>
                                </configuration>
                            </execution>
                            <execution>
                                <id>npm_test</id>
                                <goals>
                                    <goal>npm</goal>
                                </goals>
                                <phase>test</phase>
                                <configuration>
                                    <arguments>run-script test:coverage</arguments>
                                </configuration>
                            </execution>

                        </executions>
    </plugin>
```

Package.json

This file must be stored in your project and will be used by the Npm maven plugin to setup the javascript environment.

Here is an example to setup an angular environment, others libraries can be setup depending of the need

```
{
    "scripts": {
        "test": "jest",
        "test:watch": "jest --watch",
        "test:coverage": "jest --coverage"
    },
    "jest": {
        "verbose": true,
        "coverageDirectory": "${project.build.directory}/my-ui/coverage",
        "collectCoverageFrom": [
            "**/designer/**/*.{js,jsx}",
            "!**/designer/lib/**"
        ],
        "rootDir": "${project.build.directory}/my-ui",
        "roots": [
            "${project.basedir}/src/test/javascript/",
            "<rootDir>/designer/"
        ],
        "moduleDirectories": [
            "${project.build.directory}/my-ui/node/node_modules",
            "${project.build.directory}/my-ui/node_modules",
            "<rootDir>/designer"
        ],
        "coverageReporters": [
            "lcov"
        ]
    },
    "devDependencies": {
        "angular": "1.3.2",
        "angular-resource": "1.3.2",
        "angular-cookies": "1.3.2",
        "angular-route": "1.3.2",
        "angular-mocks": "1.3.2",
        "angular-animate": "1.3.2",
        "angular-sanitize": "1.3.2",
        "angular-touch": "1.3.2",
        "angular-dialog-service": "5.3.0",
        "angular-loading-bar": "0.9.0",
        "jquery": "3.3.1",
        "popper.js": "1.14.4",
        "bootstrap": "4.1.1",
        "angular-ui-bootstrap": "2.5.6",
        "jest": "^23.6.0",
        "jest-cli": "^21.2.1"
    }
}
```

Javascript test Example

Here is a short example of an angular controller test

```
require('jquery/dist/jquery.min.js');
require('angular/angular.min.js');
require('angular-mocks/angular-mocks.js');
require('angular-route/angular-route.min.js');
require('angular-resource/angular-resource.min.js');
require('angular-cookies/angular-cookies.min.js');
require('angular-animate/angular-animate.min.js');
require('angular-sanitize/angular-sanitize.min.js');
require('angular-touch/angular-touch.min.js');
require('popper.js/dist/umd/popper.min.js');
require('bootstrap/dist/js/bootstrap.min.js');
require('angular-ui-bootstrap/dist/ui-bootstrap-tpls.js');
require('angular-loading-bar/src/loading-bar.js');
require('angular-dialog-service/dist/dialogs.js');
require('scripts/app.js');
require('scripts/DashboardCtrl.js');


describe('Dashboard ctrl tests', function() {

    beforeEach(angular.mock.module('clds-app'));

    var $controllerService;

    beforeEach(angular.mock.inject(function(_$controller_) {
        $controllerService = _$controller_;
    }));

    describe('$scope.showPalette', function() {

        it('test showPalette', function() {

            var $scopeTest = {};
            var $rootScopeTest = {};
            var $resourceTest = {};
            var $httpTest = {};
            var $timeoutTest = {};
            var $locationTest = {};
            var $intervalTest = function(){};
            var $controllerDashboard = $controllerService('DashboardCtrl', {
                '$scope' : $scopeTest,
                '$rootScope' : $rootScopeTest,
                '$resource' : $resourceTest,
                '$http' : $httpTest,
                '$timeout' : $timeoutTest,
                '$location' : $locationTest,
                '$interval' : $intervalTest
            });
            $scopeTest.showPalette();
            expect($rootScopeTest.isModel).toEqual(true);
        });
    });
});
```

Concrete example

For more information, here are some examples of javascript coverage integration :

CLAMP:

https://gerrit.onap.org/r/#/c/70581/1

(add your project Review here)

Python

Your `pom.xml` file needs to specify several sonar properties. In particular, the **`<sonar.python.coverage.reportPaths>`** variable specifies the location of the `coverage.xml` file that is generated by `tox`. (Previous versions of sonar used the singular form of this variable; make certain that it has an "s" at the end of its name.)

```
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <sonar.sources>.</sonar.sources>
    <sonar.junit.reportsPath>xunit-results.xml</sonar.junit.reportsPath>
    <sonar.python.coverage.reportPaths>coverage.xml</sonar.python.coverage.reportPaths>
    <sonar.language>py</sonar.language>
    <sonar.pluginname>python</sonar.pluginname>
    <sonar.inclusions>**/**.py</sonar.inclusions>
    <sonar.exclusions>tests/*,setup.py</sonar.exclusions>
  </properties>
```

Your `tox.ini` file should list both `py36` and `py37`. Because some of our build environments might be missing one of these python versions, you may also set the `skip_missing_interpreters` variable to `true`.

```
[tox]
envlist = py36,py37
skip_missing_interpreters = true
```

For Gradle

Section TDB