Integrate Allure Reports

In this lesson, you'll learn how to integrate Allure reporting to our project and produce visually appealing test reports.

WE'LL COVER THE FOLLOWING

What is Allure?

Allure Framework is a flexible and lightweight test reporting tool that shows a very concise representation of test execution in a very intuitive web report.

Installing Allure via command line

Mac

```
brew install allure
```

Linux

```
sudo apt-add-repository ppa:qameta/allure
sudo apt-get update
sudo apt-get install allure
```

Windows

To install scoop, follow the link.

```
scoop install allure
```

Manual installation

Alternatively, we can download the latest Allure command-line binary from link, extract, and add to classpath.

Mac / Linux

```
export PATH=$PATH:</path/allure/bin>
```

Windows

```
set PATH=%PATH%;<\path\allure\bin>;
```

Adding dependency

Gradle (in `build.xml`)

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath "io.qameta.allure:allure-gradle:2.8.1"
    }
}

plugins {
    id 'io.qameta.allure'
}

def allure_version = '2.13.1'

allure {
    version = allure_version
```

```
    autoconfigure = true
    aspectjweaver = true
    clean = true
    allureJavaVersion = allure_version
    resultsDir = file('test-output/allure-results')
    reportDir = file('test-output/allure-reports')
   downloadLink = "https://repo.maven.apache.org/maven2/io/qameta/allure/allure-commandline/${allure_version}/allure-commandline-${allure_version}.zip"
}
```

Opening Allure report

```
./gradlew allureServe
```

Maven (in `pom.xml`) #

```
<properties>
    <aspectj.version>1.9.5</aspectj.version>
</properties>

<dependencies>
    <dependency>
        <groupId>io.qameta.allure</groupId>
        <artifactId>allure-testng</artifactId>
        <version>2.13.1</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.22.2</version>
            <configuration>
                <testFailureIgnore>false</testFailureIgnore>
                <argLine>
                    -javaagent:"${settings.localRepository}/org/aspectj/aspectjweaver/${aspectj.version}/aspectjweaver-${aspectj.version}.jar"
                </argLine>
            </configuration>
            <dependencies>
                <dependency>
                    <groupId>org.aspectj</groupId>
```

```
                    <artifactId>aspectjweaver</artifactId>
                    <version>${aspectj.version}</version>
                </dependency>
            </dependencies>
        </plugin>
        <plugin>
            <groupId>io.qameta.allure</groupId>
            <artifactId>allure-maven</artifactId>
            <version>2.10.0</version>
            <configuration>
                <reportVersion>2.13.1</reportVersion>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Opening Allure report

```
mvn allure:serve
```

Understanding Allure

Allure has few annotations for marking the life cycle of test execution.

- `@Step`
- `@Attachment`

`@Step`

Any action that constitutes a testing scenario is marked with `@Step`.

```
@Step("opening base url")
public void openUrl(String url) {
    ....
}
```

Here, by default, the name of the step will be taken if the name parameter is not mentioned in `@Step`. We can also have placeholders too in the name parameter. For more information, please follow the link.

This can also be done programmatically, using the below code.

```
import static io.qameta.allure.util.AspectUtils.getName;
import static io.qameta.allure.util.AspectUtils.getParameters;
import static io.qameta.allure.util.ResultsUtils.getStatus;
```

```
import static io.qameta.allure.util.ResultsUtils.getStatusDetails;
import io.qameta.allure.AllureLifecycle;

import io.qameta.allure.Step;
import io.qameta.allure.model.Parameter;
import io.qameta.allure.model.Status;
import io.qameta.allure.model.StepResult;
import io.qameta.allure.Allure;

public void openBaseUrl() {

    StepResult result = new StepResult().setName("open base url");
    Allure.getLifecycle().startStep(UUID.randomUUID().toString(), result);
    try {
        ...
        ...
        getLifecycle().updateStep(s -> s.setStatus(getStatus(e).orElse(Sta
tus.PASSED));
    } catch(Exception e) {
        getLifecycle().updateStep(s -> s.setStatus(getStatus(e).orElse(Sta
tus.BROKEN));
        throw e;
    }

}
```

`@Attachment`

The below method annotated with `@Attachment` should return either a `String` or `byte[]`. For attaching a file to the report, do as follows:

```
@Attachment(value = "adding log", type = "text/plain")
public String addAttachment() {
    return "hello";
}
```

Alternatively, for doing the same programmatically without using `@Attachment` annotation, do as follows:

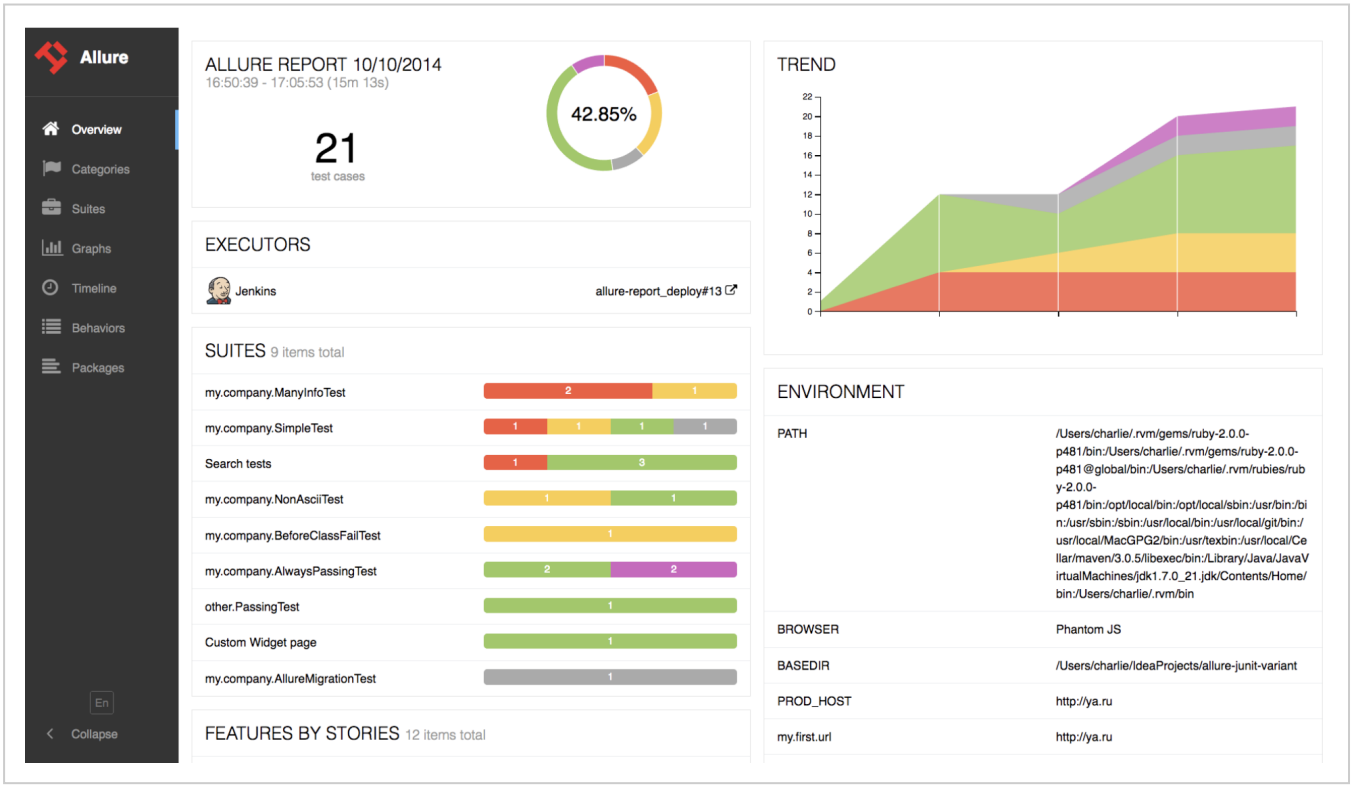
```
public void openBaseUrl() {
    ..........
    Allure.getLifecycle().addAttachment("adding log", "text/plain", ".txt"
, "hello");

}
```

Multiple overloaded methods are also available. Please follow the link for more information.

Additionally, please follow the link for more comprehensive information about Allure.

Sample report

Following is a sample report generated by Allure framework:



In this lesson, we learned how to integrate Allure to our test project. In the next section, you'll learn about designing UI test automation framework.