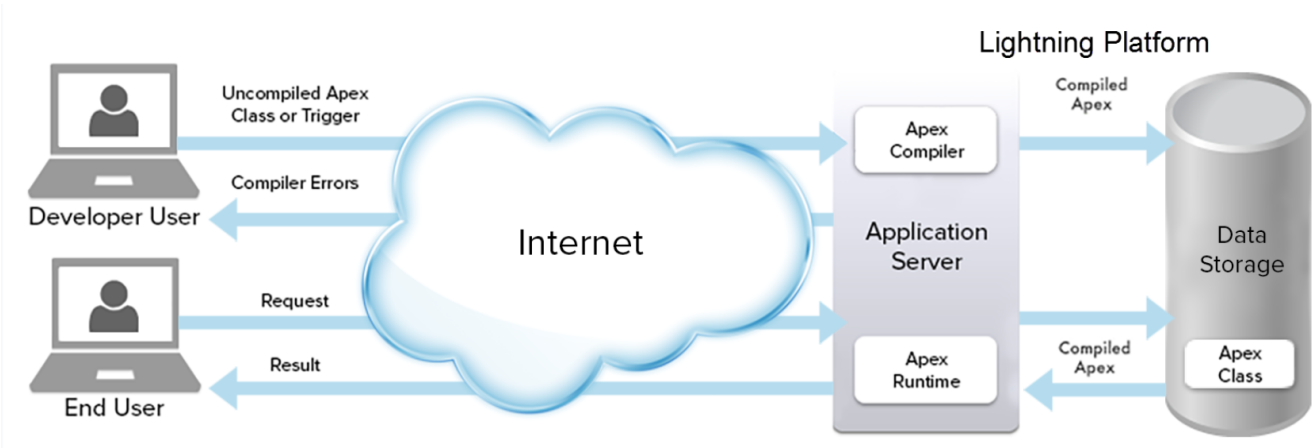
Testing Salesforce apps using Apex

- Overview
- Use case
  - Implementation using VS Code and "sfdx" CLI tool
  - Implementation using "ant migration" tool
- Checking it live
- Tips
  - Check authenticated organizations with the CLI tool
  - Running tests from within VSCode
- References

Overview

In this tutorial, we will implement an Apex application to create a simple business process in Salesforce, a well-known cloud-based CRM.

Apex is an object-oriented programming language, similar to Java, allowing developers to customize Salesforce to their needs; the code is deployed to, compiles and runs in the cloud.



Apex custom applications (code and automated tests) can be developed entirely using a browser and the developer console.

The developer console provides a complete environment for editing and debugging code, including the ability to perform queries in the database. It also allows the implementation of automated tests, running them and seeing their results.

Users and developers can either use SOQL or SOSL to extract information from the DB.

However, in a CI/CD environment, code will be edited elsewhere (e.g. using an IDE such as VSCode) and managed by some VCS (e.g. Git).

The Salesforce CLI (sfdx) is used to interact with Salesforce platform, mainly for deploying code and triggering the test runs. Results can be collected from Salesforce and processed by the CI tool and even sent to a central test management solution like Xray.

Use case

In this tutorial, we will implement a Salesforce application which aims to create an "order" whenever an "opportunity" is transitioned to a closed state if certain minimum conditions are met.

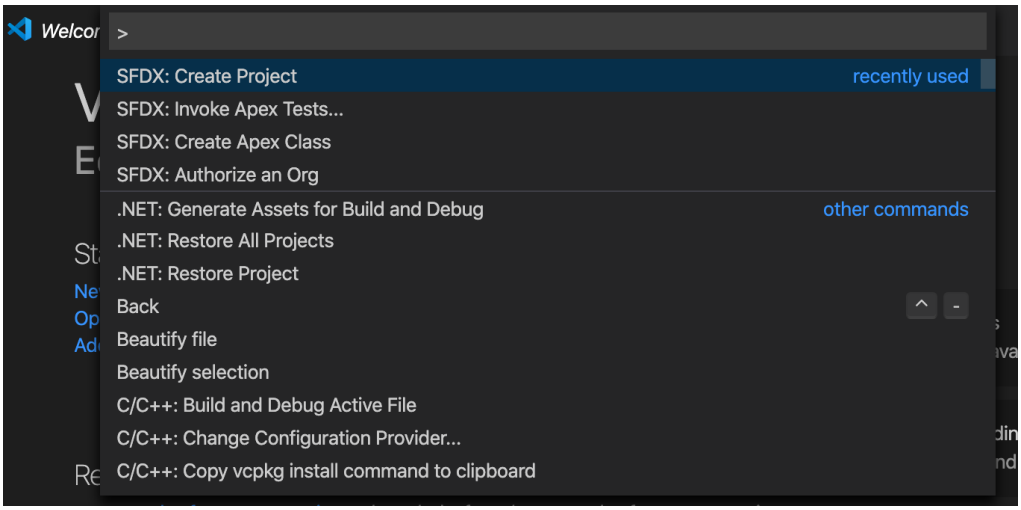
Some fields are copied from the opportunity to the new "order" object.

Implementation using VS Code and "sfdx" CLI tool

Requirements

- Salesforce CLI (i.e. sfdx)
- Salesforce Extension Pack

We start by creating an Apex project with all the necessary files. In VSCode this is straightforward.



We need to review the project configuration file, mainly the instance URL and the API version supported by our application.

sfdx-project.json

```
{
  "packageDirectories": [
    {
      "path": "force-app",
      "default": true
    }
  ],
  "namespace": "",
  "sfdcLoginUrl": "https://test.salesforce.com",
  "sourceApiVersion": "48.0"
}
```

We then can proceed to create the application code and corresponding tests.

Our application is composed of a trigger (OpportunityTrigger), helper (OpportunityTriggerHelper) and the test class (OpportunityTriggerTest).

main/default/classes/OpportunityTriggerHelper.cls

```
public class OpportunityTriggerHelper {

    public static void createOrdersforOpps(List<Opportunity> opps){
        List<Order> ordersToInsert = new List<Order>();
        for(Opportunity opp: opps){
            //Create an Order for each closed won opportunity
            Order ord = new Order();
            ord.Name = opp.Name + 'Order';
            ord.AccountId = opp.AccountId;
            ord.Pricebook2Id = opp.Pricebook2Id;
            ord.Status = 'Draft';
            ord.EffectiveDate = opp.CloseDate;
            ord.OpportunityId = opp.Id;
            ord.Type = 'Opportunity Order';
            ord.Amount__c = opp.Amount-1; // ups, there's a bug here :)

            ordersToInsert.add(ord);
        }
        if(ordersToInsert.size() > 0){
            insert ordersToInsert;
        }
    }
}
```

main/triggers/OpportunityTrigger.cls

```
public class OpportunityTriggerHelper {

    public static void createOrdersforOpps(List<Opportunity> opps){
        List<Order> ordersToInsert = new List<Order>();
        for(Opportunity opp: opps){
            //Create an Order for each closed won opportunity
            Order ord = new Order();
            ord.Name = opp.Name + 'Order';
            ord.AccountId = opp.AccountId;
            ord.Pricebook2Id = opp.Pricebook2Id;
            ord.Status = 'Draft';
            ord.EffectiveDate = opp.CloseDate;
            ord.OpportunityId = opp.Id;
            ord.Type = 'Opportunity Order';
            ord.Amount__c = opp.Amount-1;  // THIS BUG WAS ADDED ON PURPOSE

            ordersToInsert.add(ord);
        }
        if(ordersToInsert.size() > 0){
            insert ordersToInsert;
        }
    }
}
```

We will implement two basic tests: one that checks the successful case, where the "order" is created, and another for checking invalid data.

Asserts are provided by the System class.

The test class should have the **@isTest** annotation and so the test methods. Previous versions used the "testMethod" classifier in the method statement but the annotation is more clear.

Each test runs in a transaction, thus any object that is created within the test is discarded when it finishes.

test/default/classes/OpportunityTriggerTest.cls

```
@isTest
public class OpportunityTriggerTest {

    @isTest
    public static void testValidOpportunity(){

        //Account the will be related with the Opportunities
        Account acc = new Account();
        acc.Name = 'Test Account';
        insert acc;

        Id pricebookId = Test.getStandardPricebookId();

        List <Opportunity> oppsToInsert = new List <Opportunity>();

        //Opportunity to insert a related Order
        Opportunity opp = new Opportunity();
        String temp_opportunity_name = 'Test Opportunity';
        opp.Name = temp_opportunity_name;
        opp.AccountId = acc.Id;
        opp.CloseDate = system.today();
        opp.Probability = 70;
        opp.StageName = 'Closed Won';
        opp.Amount = 15000;
        opp.Pricebook2Id = pricebookId;
        oppsToInsert.add(opp);

        insert oppsToInsert;
                Opportunity lastOpp;

        Opportunity[] opps = [SELECT Id, Amount FROM Opportunity WHERE OwnerId = :UserInfo.getUserId() and name
= :temp_opportunity_name];
        System.assert((opps.size() > 0),'no opportunity was found');
        lastOpp = opps[0];

        Order order = [SELECT Id, Status, Amount__c from order where Opportunity.Name = :temp_opportunity_name];
                System.assert(order != NULL);
        System.assertEquals(opp.Amount, lastOpp.Amount);
        System.assertEquals(opp.Amount, order.Amount__c);
        System.assertEquals(order.Status, 'Draft');
    }

    @isTest
    public static void testInvalidOpportunity(){

        //Account the will be related with the Opportunities
        Account acc = new Account();
        acc.Name = 'Test Account';
        insert acc;

        Id pricebookId = Test.getStandardPricebookId();

        List <Opportunity> oppsToInsert = new List <Opportunity>();

        //Opportunity to hit the validation rule
        Opportunity opp1 = new Opportunity();
        opp1.Name = 'Test Opportunity1';
        opp1.AccountId = acc.Id;
        opp1.CloseDate = system.today();
        opp1.Probability = 70;
        opp1.StageName = 'Closed Won';
        opp1.Amount = 0;
        oppsToInsert.add(opp1);

        //Check that the Opportunity hit the validation rule
        try{
            insert oppsToInsert;
        }catch (exception e){
            system.assert(e.getMessage().contains('To save this Opportunity as Close Won you need to fill the
Pricebook field and the Amount needs to be greater than 0'));
        }
```

```
    }

}
```

After the code and tests are implemented, they need to be deployed to Salesforce using the `sfdx` tool. However, before being able to use any of `sfdx` commands, we need to authenticate it with Salesforce.

```
sfdx force:auth:web:login -r https://test.salesforce.com -a TestOrg1
```

This will store some tokens under the folder `$HOME/.sfdx/`, which will be used for subsequent `sfdx` commands.

We can also give an alias to the Salesforce organization we are connecting to. It's import to use the correct instance URL (in this case we're using the "test.salesforce.com" instance).

After authentication is done, we can deploy our application.

```
sfdx force:source:deploy -p force-app --json --loglevel fatal -u TestOrg1
```

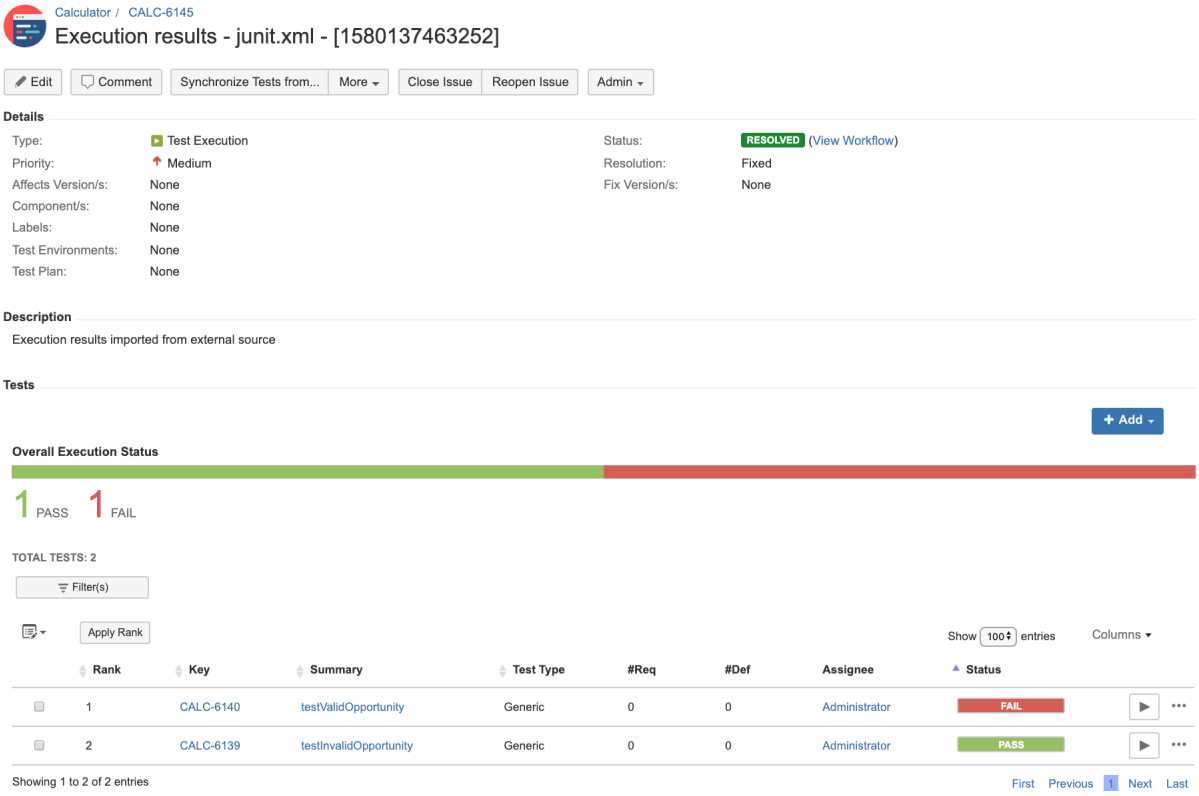
Whenever an application code is deployed, tests may also run automatically; this depends on the configuration and on the kind of environment where the code has been deployed to (more info here).

However, we may control and enforce which tests to run, based on the name of the test suite or of the test classes. The creation of a JUnit XML report is also possible.

Note that tests will run remotely, thus any change to the application code or tests needs to be deployed beforehand.

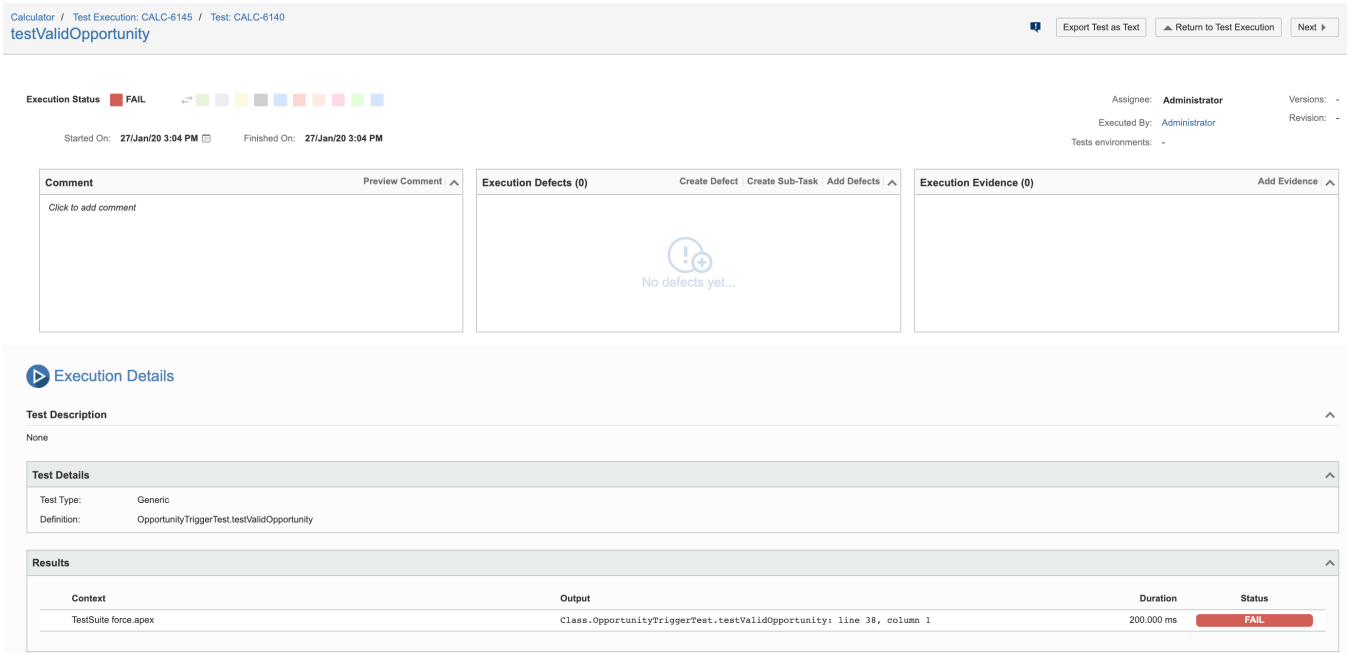
```
sfdx force:apex:test:run --resultformat human --resultformat junit --outputdir reports  --testlevel
RunSpecifiedTests -n OpportunityTriggerTest -u TestOrg1 > reports/junit.xml
```

After successfully running tests and generating the JUnit XML report (e.g. junit.xml), it can be imported to Xray (either by the REST API or by using one of the available CI addons or even through **Import Execution Results** action within the Test Execution).



Each check is mapped to a Generic Test in Jira, and the **Generic Test Definition** field contains the name of the test class followed by the name of the test method.

The Context section contains information about the log/assertion.



Implementation using "ant migration" tool

Requirements

- Ant Migration Tool
- force-deploy-with-xml-report-task

In this scenario, we're using the Ant Migration Tool to manage and deploy code changes to Salesforce.

This tailored version of ant provides some specific Salesforce tasks to manage the deployment of our applications.

However, as of version 47.0 of this tool, it does not provide a way to generate JUnit XML reports. Fortunately, there is an open-source project ("force-deploy-with-xml-report-task") that is able to extend "ant" to provide this capability.

You need to install the ant-salesforce.jar obtained from Salesforce to your maven repository.

```
mvn install:install-file -Dfile=ant-salesforce.jar -DgroupId=com.force.api -DartifactId=ant-salesforce -
Dversion=47.0.0 -Dpackaging=jar
```

Then you can build "force-deploy-with-xml-report-task", after updating it's pom.xml to reflect the version of the downloaded ant-salesforce.jar.

The file structure for our application can be based on the "sample" project provided in the zip file of the Ant Migration Tool.

We can use the code provided in the previous section and organize it as follows.

```
.
 build.properties
 build.xml
 codepkg
    classes
       OpportunityTriggerHelper.cls
       OpportunityTriggerHelper.cls-meta.xml
       OpportunityTriggerTest.cls
       OpportunityTriggerTest.cls-meta.xml
    package.xml
    triggers
        OpportunityTrigger.trigger
        OpportunityTrigger.trigger-meta.xml
 lib
    ant-salesforce.jar
    force-deploy-with-xml-report-task.jar
 mypkg
    objects
 package.xml
 removecodepkg
    destructiveChanges.xml
    package.xml
 reports
    TEST-Apex.xml
 unpackaged
     package.xml
```

The root build.xml file needs to be updated to produce the JUnit XML report; this is done in the deployCode task (we kept the original deployCode task just for comparison purpose).

/build.xml

```
<project name="Sample usage of Salesforce Ant tasks" default="test" basedir="." xmlns:sf="antlib:com.salesforce"
>
```

```
    <property file="build.properties"/>
    <property environment="env"/>

    <!-- Setting default value for username, password and session id properties to empty string
         so unset values are treated as empty. Without this, ant expressions such as ${sf.username}
         will be treated literally.
    -->
    <condition property="sf.username" value=""> <not> <isset property="sf.username"/> </not> </condition>
    <condition property="sf.password" value=""> <not> <isset property="sf.password"/> </not> </condition>
    <condition property="sf.sessionId" value=""> <not> <isset property="sf.sessionId"/> </not> </condition>

    <taskdef resource="com/salesforce/antlib.xml" uri="antlib:com.salesforce">
        <classpath>
            <pathelement location="../ant-salesforce.jar" />
        </classpath>
    </taskdef>

    <!-- Test out deploy and retrieve verbs for package 'mypkg' -->
    <target name="test">
      <!-- Upload the contents of the "mypkg" package -->
      <sf:deploy username="${sf.username}" password="${sf.password}" sessionId="${sf.sessionId}"
serverurl="${sf.serverurl}" maxPoll="${sf.maxPoll}" deployRoot="mypkg" rollbackOnError="true"/>
      <mkdir dir="retrieveOutput"/>
      <!-- Retrieve the contents into another directory -->
      <sf:retrieve username="${sf.username}" password="${sf.password}" sessionId="${sf.sessionId}"
serverurl="${sf.serverurl}" maxPoll="${sf.maxPoll}" retrieveTarget="retrieveOutput" packageNames="MyPkg"/>
    </target>

    <!-- Retrieve an unpackaged set of metadata from your org -->
    <!-- The file unpackaged/package.xml lists what is to be retrieved -->
    <target name="retrieveUnpackaged">
      <mkdir dir="retrieveUnpackaged"/>
      <!-- Retrieve the contents into another directory -->
      <sf:retrieve username="${sf.username}" password="${sf.password}" sessionId="${sf.sessionId}"
serverurl="${sf.serverurl}" maxPoll="${sf.maxPoll}" retrieveTarget="retrieveUnpackaged" unpackaged="unpackaged
/package.xml"/>
    </target>

    <!-- Retrieve all the items of a particular metadata type -->
    <target name="bulkRetrieve">
      <sf:bulkRetrieve username="${sf.username}" password="${sf.password}" sessionId="${sf.sessionId}"
serverurl="${sf.serverurl}" maxPoll="${sf.maxPoll}" metadataType="${sf.metadataType}" retrieveTarget="
retrieveUnpackaged"/>
    </target>

    <!-- Retrieve metadata for all the packages specified under packageNames -->
    <target name="retrievePkg">
      <sf:retrieve username="${sf.username}" password="${sf.password}" sessionId="${sf.sessionId}"
serverurl="${sf.serverurl}" maxPoll="${sf.maxPoll}" retrieveTarget="retrieveOutput" packageNames="${sf.pkgName}"
/>
    </target>

    <!-- Deploy the unpackaged set of metadata retrieved with retrieveUnpackaged and run tests in this
organization's namespace only-->
    <target name="deployUnpackaged">
      <sf:deploy username="${sf.username}" password="${sf.password}" sessionId="${sf.sessionId}"
serverurl="${sf.serverurl}" maxPoll="${sf.maxPoll}" deployRoot="retrieveUnpackaged" rollbackOnError="true"/>
    </target>

    <!-- Deploy a zip of metadata files to the org -->
    <target name="deployZip">
      <sf:deploy username="${sf.username}" password="${sf.password}" sessionId="${sf.sessionId}"
serverurl="${sf.serverurl}" maxPoll="${sf.maxPoll}" zipFile="${sf.zipFile}" pollWaitMillis="1000"
rollbackOnError="true"/>
    </target>

    <!-- Shows deploying code & running tests for code in directory -->
    <target name="deployCode">
      <!-- Upload the contents of the "codepkg" directory, running the tests for just 1 class -->

  <path id="ant.additions.classpath">
```

```
      <fileset dir="lib/"/>
  </path>

     <taskdef
      name="sfdeploy"
      classname="com.salesforce.ant.DeployWithXmlReportTask"
      classpathref="ant.additions.classpath"
      />
      <delete dir="./reports" quiet="true"/>

      <sfdeploy username="${sf.username}" password="${sf.password}" sessionId="${sf.sessionId}" serverurl="${sf.
serverurl}" maxPoll="${sf.maxPoll}" deployRoot="codepkg" testLevel="RunSpecifiedTests" rollbackOnError="true"
          junitreportdir="./reports"
          >
               <runTest>OpportunityTriggerTest</runTest>
      </sfdeploy>
    </target>

      <!-- Shows deploying code & running tests for code in directory -->
      <target name="deployCodeOrig">
        <!-- Upload the contents of the "codepkg" directory, running the tests for just 1 class -->
        <sf:deploy username="${sf.username}" password="${sf.password}" sessionId="${sf.sessionId}"
serverurl="${sf.serverurl}" maxPoll="${sf.maxPoll}" deployRoot="codepkg" testLevel="RunSpecifiedTests"
rollbackOnError="true">
           <runTest>OpportunityTriggerTest</runTest>
        </sf:deploy>
      </target>


         <!-- Shows deploying code with no TestLevel sepcified -->
    <target name="deployCodeNoTestLevelSpecified">
      <sf:deploy username="${sf.username}" password="${sf.password}" sessionId="${sf.sessionId}"
serverurl="${sf.serverurl}" maxPoll="${sf.maxPoll}" deployRoot="codepkg" rollbackOnError="true"/>
    </target>

        <!-- Shows deploying code and running tests only within the org namespace -->
        <target name="deployCodeRunLocalTests">
          <sf:deploy username="${sf.username}" password="${sf.password}" sessionId="${sf.sessionId}"
serverurl="${sf.serverurl}" maxPoll="${sf.maxPoll}" deployRoot="codepkg" rollbackOnError="true"  testlevel="
RunLocalTests"/>
        </target>

    <!-- Shows removing code; only succeeds if done after deployCode -->
    <target name="undeployCode">
      <sf:deploy username="${sf.username}" password="${sf.password}" sessionId="${sf.sessionId}"
serverurl="${sf.serverurl}" maxPoll="${sf.maxPoll}" deployRoot="removecodepkg"/>
    </target>

    <!-- Shows retrieving code; only succeeds if done after deployCode -->
    <target name="retrieveCode">
      <!-- Retrieve the contents listed in the file codepkg/package.xml into the codepkg directory -->
      <sf:retrieve username="${sf.username}" password="${sf.password}" sessionId="${sf.sessionId}"
serverurl="${sf.serverurl}" maxPoll="${sf.maxPoll}" retrieveTarget="codepkg" unpackaged="codepkg/package.xml"/>
    </target>

    <!-- Shows deploying code, running all tests, and running tests (1 of which fails), and logging. -->
    <target name="deployCodeFailingTest">
      <!-- Upload the contents of the "codepkg" package, running all tests -->
      <sf:deploy username="${sf.username}" password="${sf.password}" sessionId="${sf.sessionId}"
serverurl="${sf.serverurl}" maxPoll="${sf.maxPoll}" deployRoot="codepkg" testLevel="RunAllTestsInOrg"
rollbackOnError="true" logType="Debugonly"/>
    </target>

    <!-- Shows check only; never actually saves to the server -->
    <target name="deployCodeCheckOnly">
      <sf:deploy username="${sf.username}" password="${sf.password}" sessionId="${sf.sessionId}"
serverurl="${sf.serverurl}" maxPoll="${sf.maxPoll}" deployRoot="codepkg" checkOnly="true"/>
    </target>

        <!-- Shows quick deployment of recent validation. Set the property sf.recentValidationId to your recent
check only deployment Id -->
```

```
        <target name="quickDeploy">
          <sf:deployRecentValidation  username="${sf.username}" password="${sf.password}" sessionId="${sf.
sessionId}" serverurl="${sf.serverurl}" maxPoll="${sf.maxPoll}" recentValidationId="${sf.recentValidationId}"/>
        </target>

        <!-- Shows cancel deployment of deploy request either pending or in progress. Set property sf.requestId
to Id of pending or in progress deploy request -->
        <target name="cancelDeploy">
          <sf:cancelDeploy  username="${sf.username}" password="${sf.password}" serverurl="${sf.serverurl}"
maxPoll="${sf.maxPoll}" requestId="${sf.requestId}"/>
        </target>

        <!-- Retrieve the information of all items of a particular metadata type -->
    <target name="listMetadata">
      <sf:listMetadata username="${sf.username}" password="${sf.password}" sessionId="${sf.sessionId}"
serverurl="${sf.serverurl}" metadataType="${sf.metadataType}"/>
    </target>

        <!-- Retrieve the information on all supported metadata type -->
    <target name="describeMetadata">
      <sf:describeMetadata username="${sf.username}" password="${sf.password}" sessionId="${sf.sessionId}"
serverurl="${sf.serverurl}"/>
    </target>
</project>
```

Code can be deployed and tests run; rollback is performed if tests fail.

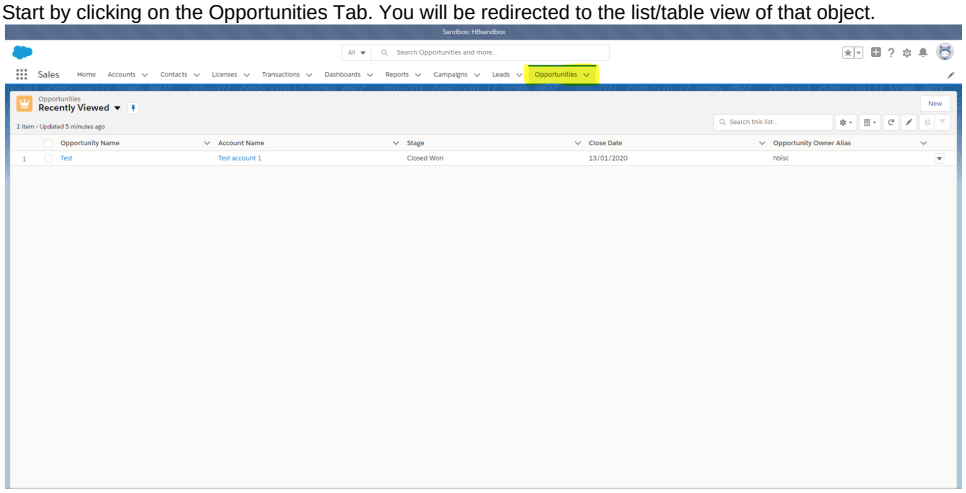
```
$  ant deployCode
Buildfile: /Users/smsf/exps/salesforce/create_order_from_opportunity_using_ant/create_order_from_opportunity
/build.xml

deployCode:
   [delete] Deleting directory /Users/smsf/exps/salesforce/create_order_from_opportunity_using_ant
/create_order_from_opportunity/reports
 [sfdeploy] Request for a deploy submitted successfully.
 [sfdeploy] Request ID for the current deploy task: 0Af3N000005MfsoSAC
 [sfdeploy] Waiting for server to finish processing the request...
 [sfdeploy] Request Status: Pending
 [sfdeploy] Request Status: Pending
 [sfdeploy] Request Status: Failed

BUILD FAILED
/Users/smsf/exps/salesforce/create_order_from_opportunity_using_ant/create_order_from_opportunity/build.xml:74:
Failures:
Test failure, method: OpportunityTriggerTest.testValidOpportunity -- System.AssertException: Assertion Failed:
Expected: 15000, Actual: 14999.00 stack Class.OpportunityTriggerTest.testValidOpportunity: line 38, column 1


There are 6 flows that have no coverage:
         - CancelEvent
         - CreateEvent
         - License_Booleans_Flow
         - License_Process
         - OnCalendlyActionCreated
         - Parent_Booleans_Flow


Total time: 26 seconds
```

The JUnit XML report is stored under the `reports/` folder. It can then be submitted to Xray using the REST API, the UI or using one of the available CI addons.

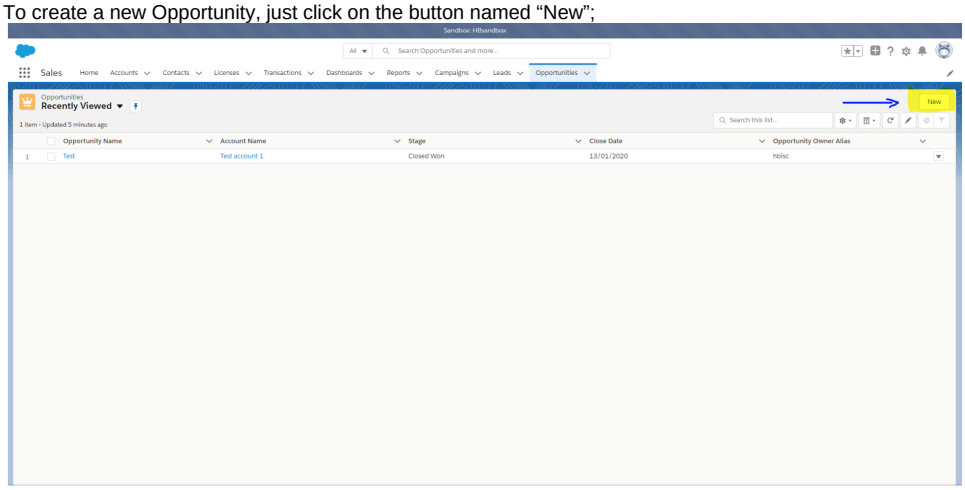
Checking it live

After deploying and testing your code, you may see it live.

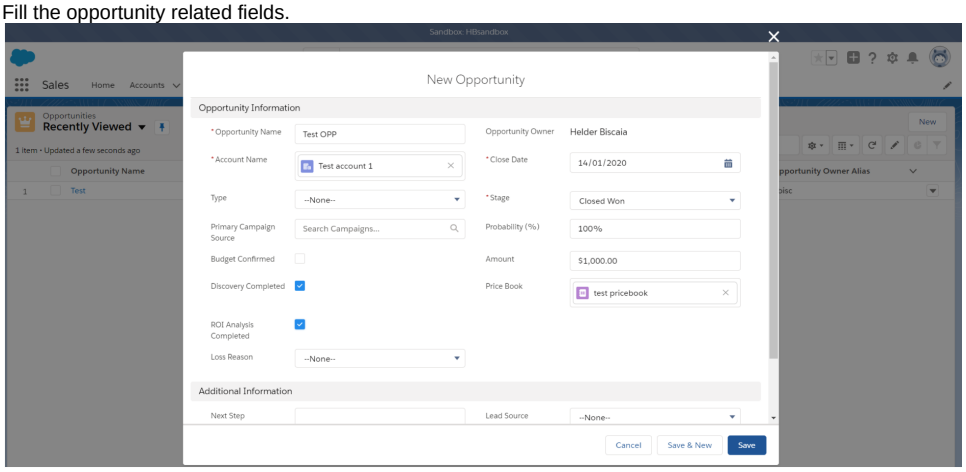
1. Start by clicking on the Opportunities Tab. You will be redirected to the list/table view of that object.



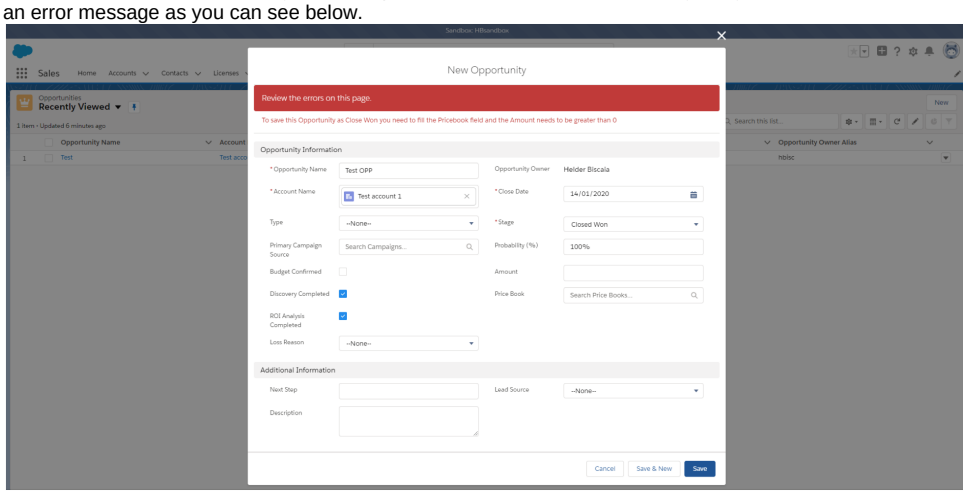
2. To create a new Opportunity, just click on the button named “New”;



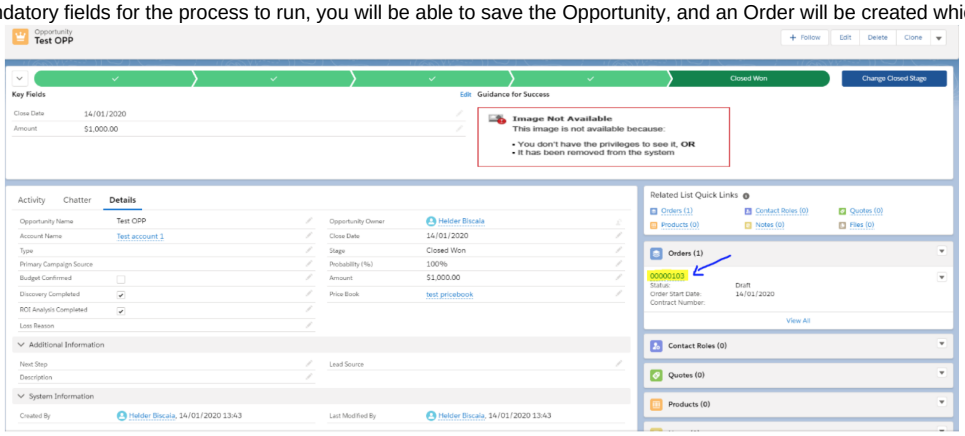
3. Fill the opportunity related fields.



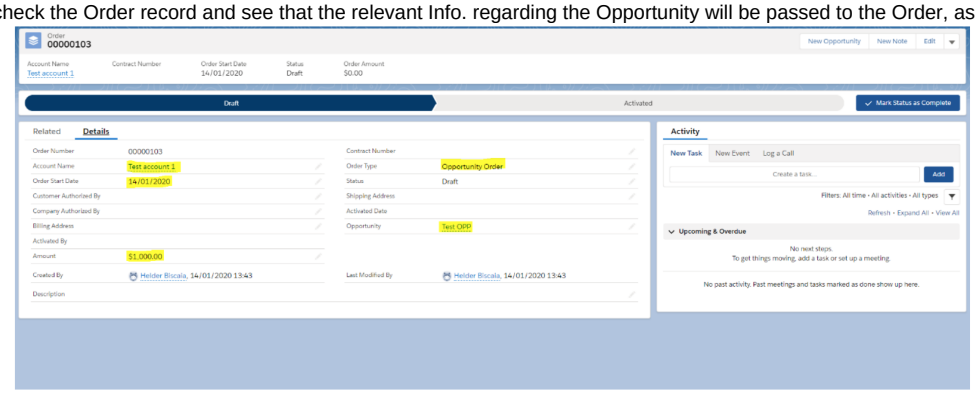
4. For the process to run, you need to fill certain fields, but only if the “**Stage**” field has the value of “**Closed won**”, if so, you need to fill the fields “**Pri ce Book**” and the “**Amount**” or you will be presented with a validation rule. If you try to save without those fields filled, you will be presented with an error message as you can see below.



5. After filling the mandatory fields for the process to run, you will be able to save the Opportunity, and an Order will be created which will be related to this Opportunity.



6. Finally, you can check the Order record and see that the relevant Info. regarding the Opportunity will be passed to the Order, as you can see in the image below.



Tips

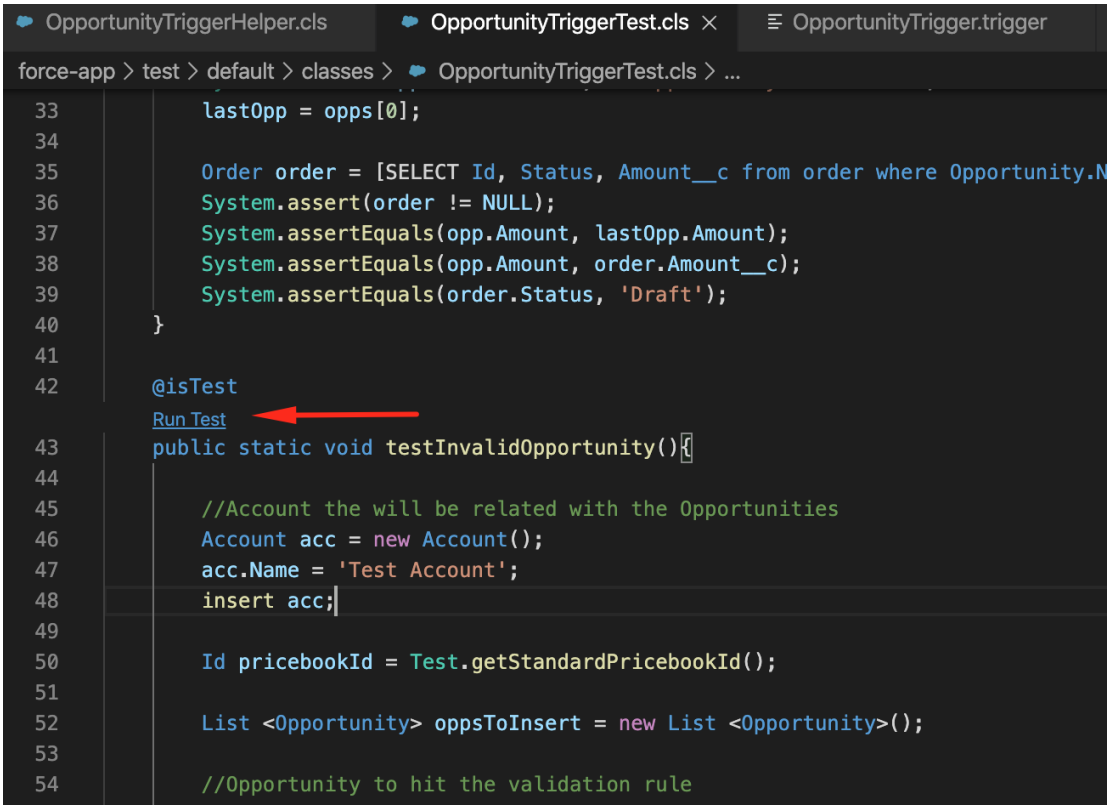
Check authenticated organizations with the CLI tool

You can quickly assess the connected/authenticated

```
$ sfdx force:org:list
=== Orgs
  ALIAS     USERNAME                                ORG ID              CONNECTED STATUS

  TestOrg1  xxxxxxxxxxxxxx@xpand-it.com.hbsandbox  00D3N0000008oIsUAI  Connected
```

Running tests from within VSCode

It's possible to trigger the running of tests from VSCode; however, updated code must be deployed first to the Salesforce organization.



References

- https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_dev_guide.htm
- https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_testing.htm
- https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_reference.htm
- https://developer.salesforce.com/tools/sfdxcli
- https://www.tutorialspoint.com/apex/apex_testing.htm
- SOQL vs SOSL
- https://trailhead.salesforce.com/en/content/learn/modules/apex_testing
- https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_qs_HelloWorld.htm
- https://developer.salesforce.com/docs/atlas.en-us.daas.meta/daas/commondeploymentissues.htm
- https://github.com/beamso/force-deploy-with-xml-report-task