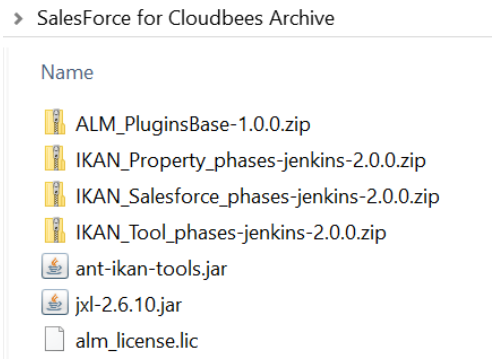
The IKAN Salesforce plugin on CloudBees

Pre-requisites

- You have an up and running Salesforce
- You have Subversion or Git to version your Salesforce objects
- You have an up and running Jenkins/CloudBees installation
- You are familiar with Salesforce, Subversion or Git and Jenkins/CloudBees

If so, the IKAN Salesforce plugin for Jenkins/CloudBees enables teams to have a Jenkins/CloudBees based complete CI/CD solution for Salesforce.

In order to run the complete CI/CD solution for Salesforce you will find in your downloaded Salesforce for CloudBees Archive following:



Installation

We will start with the ANT related files.

Next we will do:

- the ALM_PluginsBase-1.0.0.zip,
- the IKAN_Property_phases-jenkins-2.0.0.zip,
- the IKAN_Tool_phases-jenkins-2.0.0.zip and
- the IKAN_Salesforce_phases-jenkins-2.0.0.zip.

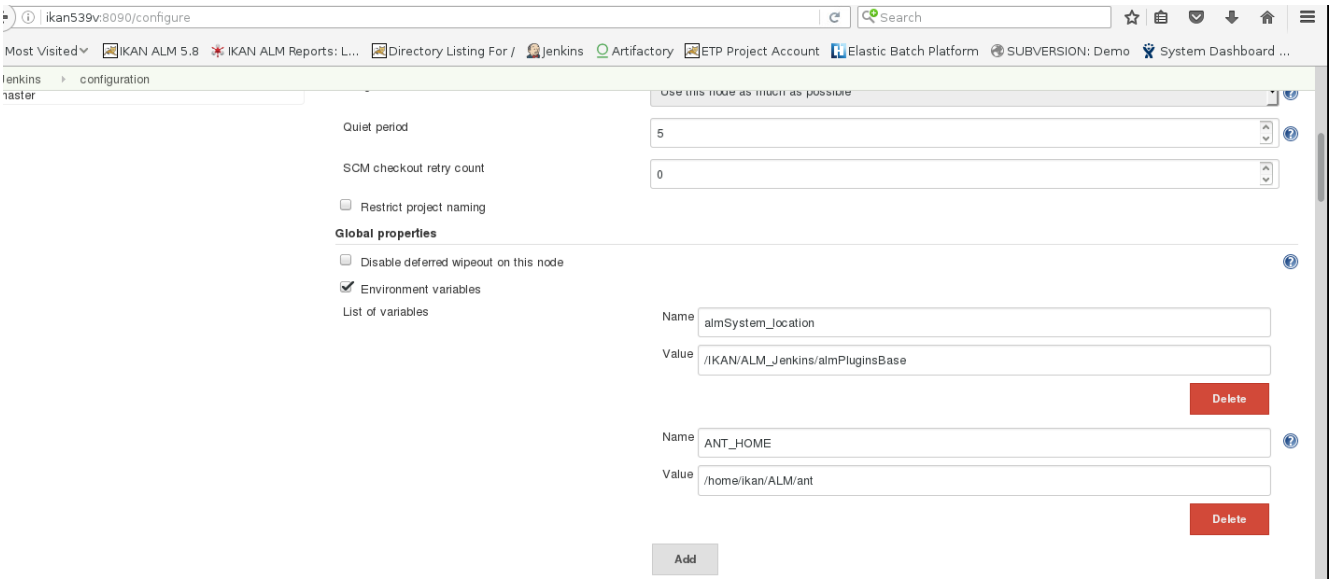
Once that is done we can start with the Installation of the Plugin on Jenkins

The ANT related files

The ANT related files, **ant-ikan-tools.jar** and **jxl-2.6.10.jar** must be copied in the [ANT_HOME]/lib of the Ant installation used by your CloudBees/Jenkins installation.

You can find where your Jenkins ANT is installed in the Jenkins Configure > Global Properties > Environment variables with name ANT_HOME, Value:

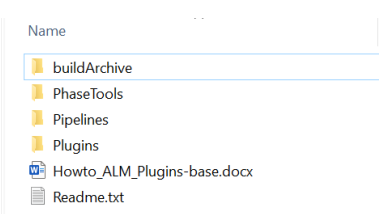
In our example: /home/ikan/ALM/ant/lib



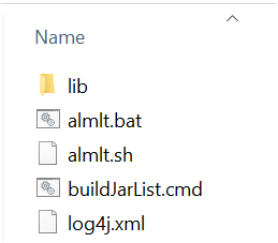
The ALM_PluginsBase-1.0.0.zip

Extract the ALM_PluginsBase.zip:

You will obtain following default directory ALM_PluginsBase with



Open the PhaseTools and the config directory to find the almlt.bat and almt.sh file:



Next, For Windows:

Make sure you use the Jenkins Server user for doing the following:

1. If necessary set the JAVA Home path in the almlt.bat file
2. Make sure you have the license file moved here: **alm_license.lic**
3. Open a console in the directory where you have the almlt.bat file

   Run following command: **almlt.bat install alm_license.lic**

Or For Linux:

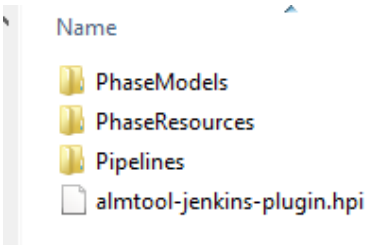
Make sure you use the Jenkins Server user for doing the following:

1. If necessary set the JAVA Home path in the almlt.sh file
2. Make sure you have the license file moved here: **alm_license.lic**
3. Open a terminal in the directory where you have the almlt.sh file

   Run following command: **almlt.sh install alm_license.lic**

IKAN_Property_phases-jenkins-2.0.0.zip

Extract the IKAN_Property_phases-jenkins-2.0.0.zip



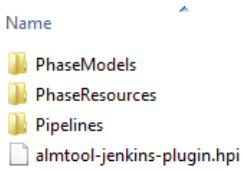
Copy/Paste the PhaseModels, PhaseResources and Pipelines Folder into the ALM_PluginsBase folder

Next Copy/Paste the almproperty-jenkins-plugin-2.0.0-SNAPSHOT.hpi file into the ALM_PluginsBase\Plugins folder.

IKAN_Tool_phases-jenkins-2.0.0.zip

Extract the IKAN_Tool_phases-jenkins-2.0.0.zip.

You will obtain following directory TOOL_phases



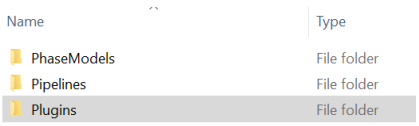
Copy/Paste the PhaseModels, PhaseResources and Pipelines Folders into the ALM_PluginsBasefolder

Next copy the almtool-jenkins-plugin.hpi file and Paste the file into ALM_PluginsBase\Plugins folder.

IKAN_Salesforce_phases-jenkins-2.0.0.zip

Extract IKAN_Salesforce_phases-jenkins-2.0.0.zip:

You will obtain following directory Salesforce_phases



Copy/Paste the PhaseModels and Pipelines Folders into the ALM_PluginsBasefolder

Next open the Plugins folder and copy/Paste the almSalesforce-jenkins-plugin-2.0.0-SNAPSHOT.hpi file into ALM_PluginsBase\Plugins folder.

CI/CD on Jenkins/CloudBees for Salesforce

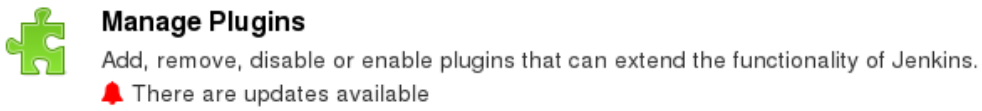
First you need to install the Plugin and make sure your Jenkins/CloudBees configuration is OK.

Once that is done we can start creating projects.

Let us start with the **Plugin install and the Jenkins/CloudBees configuration.**

Start Jenkins, go to Manage Jenkins, Manage Plugins and select the Advanced tab.

Manage the Plugins



Go to Upload plugin and browse for the *.hpi files described below. You can find the plugins into the ALM_PluginsBase\Plugins folder.

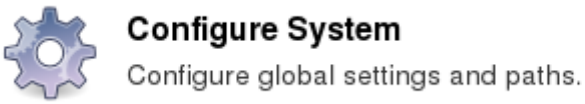
- The almbase-jenkins-plugin-1.0.0-SNAPSHOT.hpi
- The almproperty-jenkins-plugin-2.0.0-SNAPSHOT.hpi
- The almtool-jenkins-plugin-2.0.0-SNAPSHOT.hpi
- The almSalesforce-jenkins-plugin-2.0.0-SNAPSHOT.hpi.

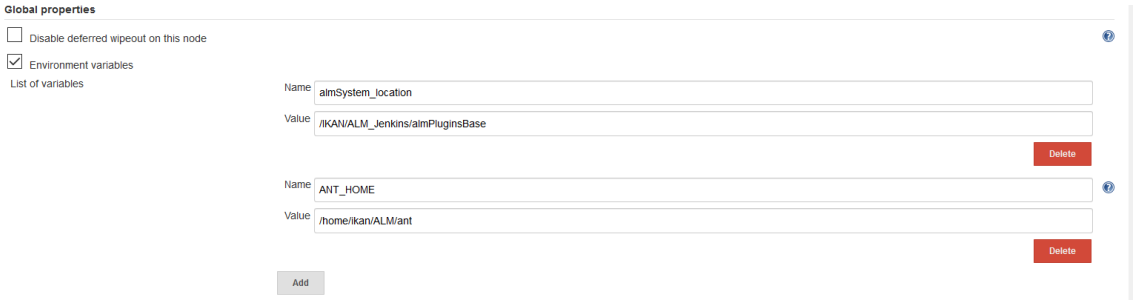
Next make sure that following standard plugins are available and enabled. If not available, download them first:

- Build Pipeline Plugin
- Build Timeout
- Pipeline Utility Steps
- Workspace Cleanup Plugin

Next, go to Manage Jenkins, Configure System, Global properties, Environment variables

Jenkins/CloudBees configuration





Select Environment variables

Add following variables: almSystem_location and ANT_HOME

As name, set: almSystem_location

As value: the location where you have unzipped the ALM_PluginBase installation folder.

Example: /IKAN/ALM_Jenkins/almPluginsBase

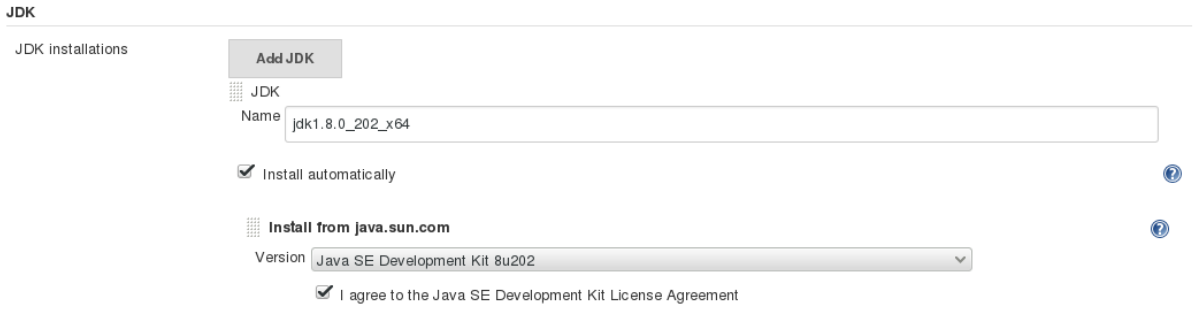
As name, set: ANT_HOME

As value: the location where you have your ANT_HOME

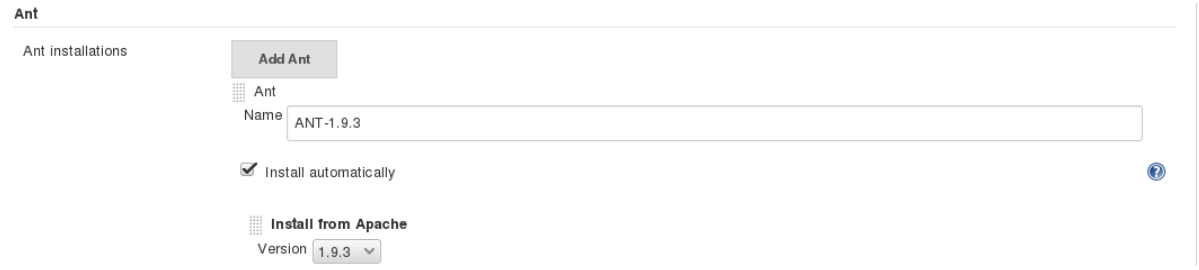
Example: /home/ikan/ALM/ant/lib

Next, go to Manage Jenkins, Global Tool Configuration

Add, if not available a JDK Installation:



Add, if not available an ANT Installation:



The Salesforce Jenkins/CloudBees solution

As mentioned in the beginning of this document there are a number of pre-requisites:

- You have an up and running Salesforce
- You have Subversion or Git to version your Salesforce objects
- You have an up and running Jenkins/CloudBees installation
- You are familiar with Salesforce, Subversion or Git and Jenkins/CloudBees

Good! Now we will explain the Groovy scripts we have available and next we will explain how to setup a project, type Pipeline, in Jenkins/CloudBees

We have following Salesforce Groovy scripts for you:

- Salesforcebulkretrieve
- Salesforcedeploy
- Salesforcedeploycancel
- Salesforcedeployrecentvalidation
- Salesforcedescribemetadata
- Salesforcelistmetadata
- Salesforceretrieve

Next we will give you some examples of possible Cloudbees/Jenkins Pipelines using some standard IKAN scripts and the salesforceRetrieve and salesforceDeploy scripts.

Jenkins Sample Pipelines for Salesforce

Here we give you some examples on how to create CloudBees/Jenkins Pipelines using following Groovy scripts

- almPipeline_sfInit.groovy
- almPipeline_sfRetrieve.groovy
- almPipeline_sfBuild.groovy
- almPipeline_sfDeploy.groovy

They allow you to have two scenarios:

Start from the Package.xml file or the VCR content (package.xml file and all other files).

Scenario one: Start from the Package.xml

In the first one Jenkins extracts the Package.xml file (or more) of the VCR Trunk or Branch Repository and puts the results in the Jenkins Workspace.

Next we use the INIT script that copies the Jenkins Workspace content to an IKAN Workspace.

Next the RETRIEVE task uses the Package.XML file to know what he needs to retrieve from SalesForce and puts it, by default in a zip file.

Next you can deploy or put files everywhere.

Pipelines used: INIT, RETRIEVE and next DEPLOY

Scenario two: Start from the VCR content (package file and all other files).

In the second one Jenkins extracts the files content of the VCR Trunk or Branch Repository and puts the results in the Jenkins Workspace.

Next the BUILD script uses all files and puts them, by default in an archive (zip file).

Next you can deploy.

Pipelines used: BUILD and DEPLOY

Now we will explain how you can setup these different Pipelines.

First Pipeline: Extracting SalesForce data files using a Version Control Repository (VCR)

Here we explain how you can create a Jenkins' project starting from a Version Control Repository:

GIT or Subversion and by selecting mainly the Package.XML file.

Groovy script used: almPipeline_sfInit.groovy

This Groovy script must first be added to the VCR project (trunk and/or branch you are working with)

Groovy script description:

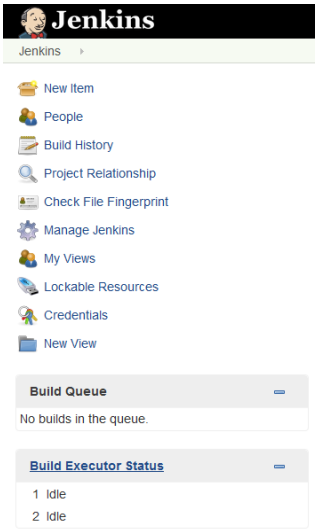
The script contains IKAN Phases that copy the check-outed files from Subversion or Git and put them from the Jenkins' default workspace to a work folder.

There are 3 IKAN phases included:

- "propertyInitial" phase will create a property file using some Jenkins' parameters. (Look at the property model file for more details)
- "propertyBuild" phase uses the first property file and will create a new property file with the correct Build properties for the next step. (Look at the property model file for more details)
- "copySourceToTarget" phase will copy the selected files to the other work folder or an archive.

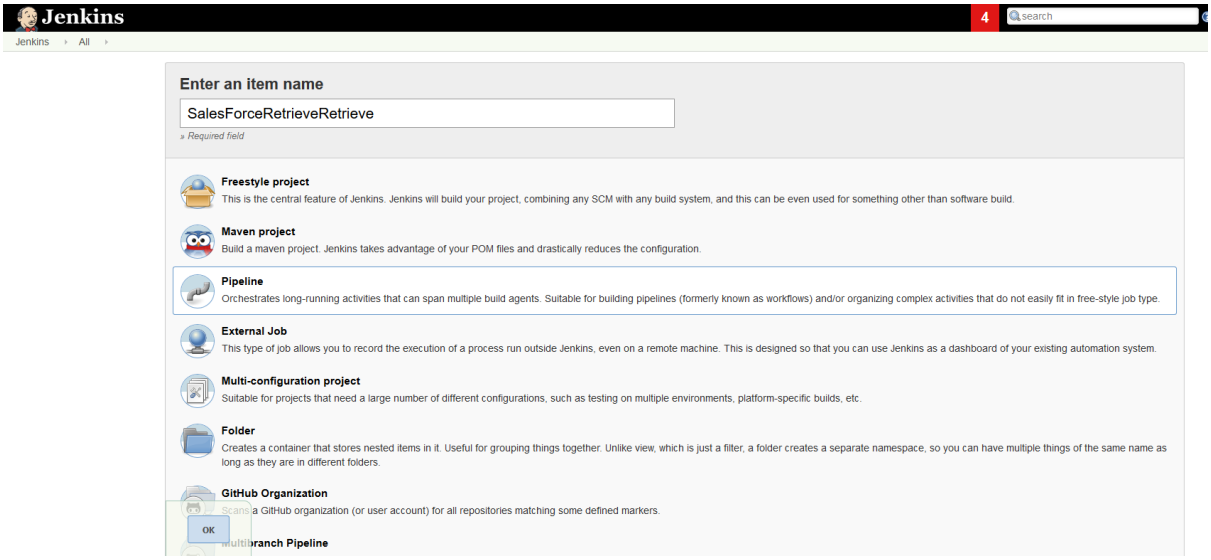
In our sample pipeline we will use Subversion

Go to the Jenkins start page and click on New Item:



Enter an Item Name, select ‘Pipeline’ project and click the OK button:

Item Name, by example: SalesforceRetrieveRetrieve



Now we will go to the Advanced Project Options tab and we will go to the Pipeline tab:

In the definition we select ‘Pipeline script from SCM’

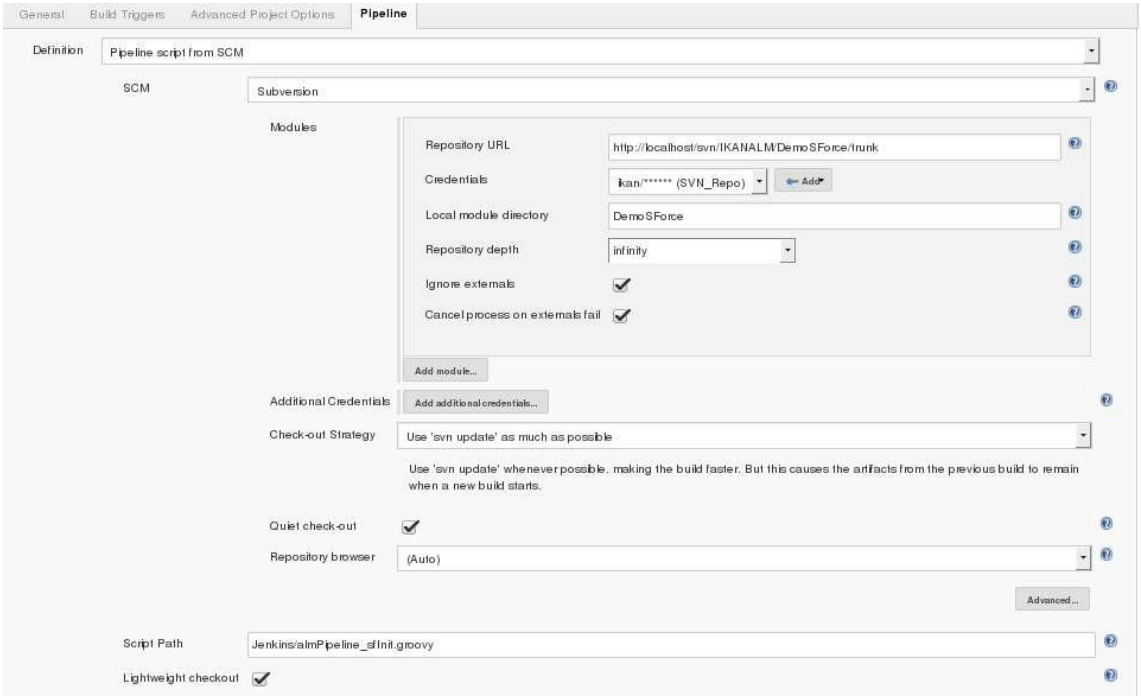
As SCM we select Subversion

For Subversion we enter:

- the Repository URL
- user/password (credentials)
- local module directory (your Subversion SalesForce project folder)

In the script path location you enter the almPipeline_sfInit.groovy file relative location and click save.

In our case we have created a folder Jenkins in our Subversion project and added the script there.



With the last Jenkins versions, the script must be included in the VCR project, because it is verified by the SCM step. Then, the Script Path will be a relative location in the VCR project.

Next go to the General tab and Select: “This project is parameterized” to add two parameters, type string.

You can find what parameters to add in the almPipeline_sfInit.groovy Groovy script. Search for "Parameters of the Jenkins Project"

```
// Parameters of the Jenkins Project
// project_name = 'MyProject'
// branch_name = 'trunk'
```

Project_name is the name of your project in the VCR Trunk or Branch

Branch_name is the name of your VCR Trunk or Branch

Second Pipeline: Extracting Salesforce data files using a package.xml file

Groovy script used: almPipeline_sfRetrieve.groovy

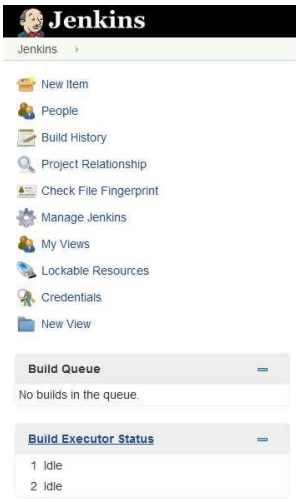
Groovy script description:

The script contains IKAN Phases to retrieve the files directly from your Salesforce environment and will put them into an Archive (used for deployment or to put into VCR project).

There are 3 IKAN phases included and an 'archive' step:

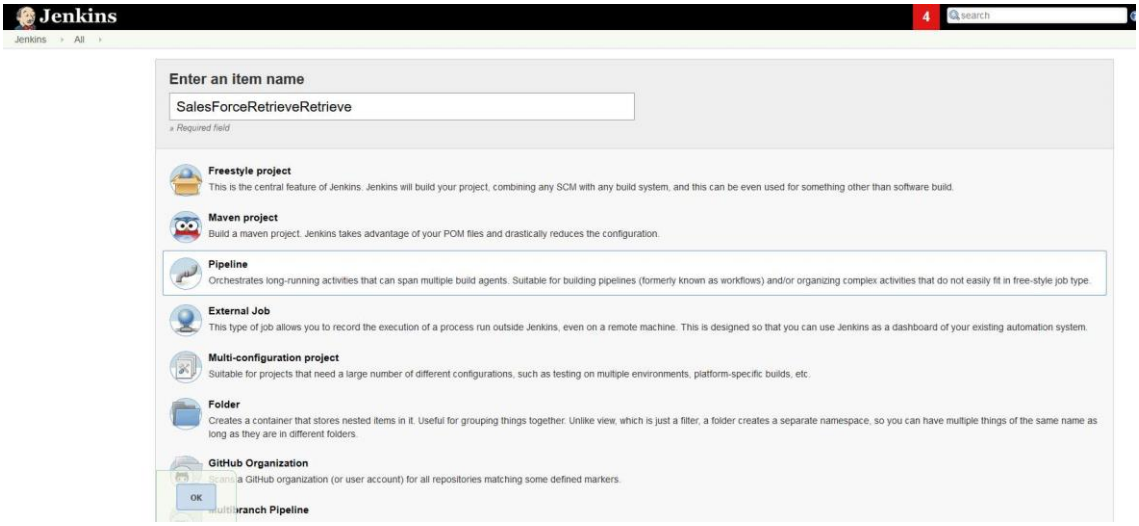
- "propertyInitial" phase will create a property file using some Jenkins' parameters. (Look at the property model file for more details)
- "propertyBuild" phase uses the first property file and will create a new property file with the correct Build properties for the next step. (Look at the property model file for more
- details)
- "salesforceRetrieve" phase will connect to Salesforce for extracting and storing the selected files into a target folder as a zip file or a project folder. (Look at the sfRetrieve*_xml.model models for details)
- The last step will create an archive from the Retrieved(extracted) output.

Go to the Jenkins start page and click on New Item:

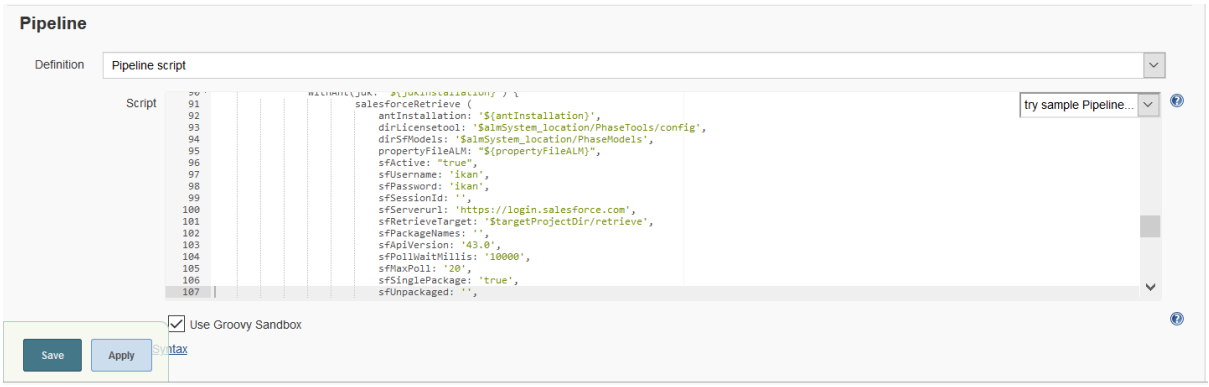


Enter an Item Name, select ‘Pipeline’ project and click the OK button:

Item Name, by example: SalesForceRetrieveRetrieve



Next go to the Pipeline and copy the almPipeline_sfRetrieve.groovy Pipeline script here and save:



Make sure you have the right values for your Salesforce for:

```
echo "Start Salesforce Retrieve"
withEnv(["ANT_ARGS=-q -propertyfile ${propertyFileALM}"]) {
    withAnt(jdk: "${jdkInstallation}") {
        salesforceRetrieve (
            antInstallation: '${antInstallation}',
            dirLicensetool: '$almSystem_location/PhaseTools/config',
            dirSfModels: '$almSystem_location/PhaseModels',
            propertyFileALM: "${propertyFileALM}",
            sfActive: "true",
            sfUsername: 'ikan',
            sfPassword: 'ikan',
            sfSessionId: '',
            sfServerurl: 'https://login.salesforce.com',
            sfRetrieveTarget: '$targetProjectDir/retrieve',
            sfPackageNames: '',
            sfApiVersion: '43.0',
            sfPollWaitMillis: '10000',
            sfMaxPoll: '20',
            sfSinglePackage: 'true',
            sfUnpackaged: '',
            sfUnzip: 'true',
            sfTrace: 'false'
```

Next go to the General tab and Select: “This project is parameterized” to add two parameters, type string.

You can find what parameters to add in the almPipeline_sfRetrieve.groovy Groovy script. Search for “Parameters of the Jenkins Project”

```
// Parameters of the Jenkins Project
// project_name = 'MyProject'
// branch_name = 'trunk'
```

Standard, the results will be put in a ZIP file.

We have defined the sfUnzip = true. Which means that the results will be stored unzipped in the sfRetrieveTarget location.

Third Pipeline: Build Salesforce package(s) or unpackaged files

Groovy script used: almPipeline_sfBuild.groovy

Groovy script description:

The script contains IKAN Phases for copying the check-outed files coming from Subversion or Git and put to the Jenkins' workspace by default to a work folder.

There are 3 IKAN phases included and an 'archive' step:

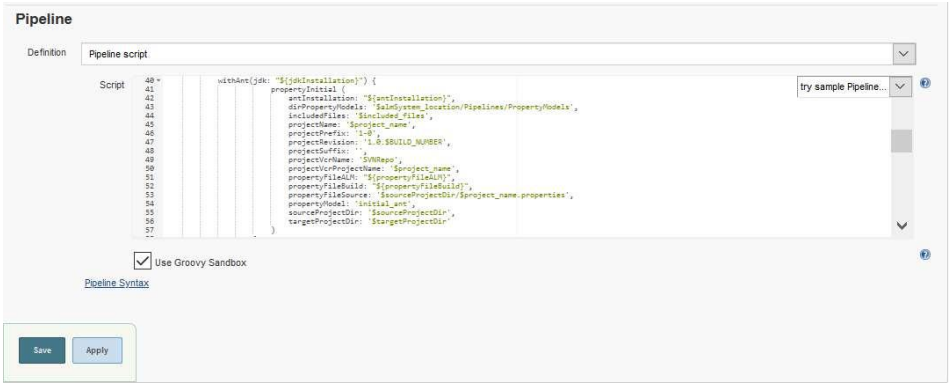
- "propertyInitial" phase will create a property file using some Jenkins' parameters. (Look at the property model for details)
- "propertyBuild" phase using the first property file will create a new one with useful Build properties for the next step. (Look at the property model for details)
- "copySourceToTarget" phase will copy the selected files to other work folder.
- The last step will create an archive of the copied files that later will be used for deployment.

Here we follow the same process:

Create a new item, called SalesForceBranchArchive, type Pipeline.

Here you need also to add the two parameters: select: "This project is parameterized" to add two parameters, type string for Project_name and Branch_name

Next install the pipeline script with the content of the almPipeline_sfBuild.groovy file.



Standard, the results will be put in a ZIP file.

We have defined the sfunzip = true. Which means that the results will be stored unzipped.

Fourth Pipeline: Deploy Salesforce package(s) or unpackaged files

Groovy script used: almPipeline_sfDeploy.groovy

Groovy script description:

Here this Pipeline will use the files stored in the Archive and in addition you can select just the files you want to use for deployment to Test or Production.

The script contains IKAN Phases for retrieving the files coming from your Salesforce environment and put them to an Archive (to deploy or to put to a VCR project).

There are 4 IKAN phases included:

- “propertyInitial” phase will create a property file using some Jenkins’ parameters. (Look at the property model file for more details)
- “propertyDeploy” phase uses the first property file and will create a new one with useful Deploy properties for the next step. (Look at the property model for more details)
- “copySourceToTarget” phase will copy the selected files to another work folder used for the deployment.
- “salesforceDeploy” phase will connect to Salesforce to deploy package(s) or unpackaged files (Look at the sfDeploy*_xml.model models for more details)

Here we follow the same process:

Create a new item, called salesForceBranchDeploy, type Pipeline.

Here you need also to add the two parameters: select: “This project is parameterized” to add two parameters, type string for Project_name and Branch_name

And a third parameter called BUILD_ARCHIVE, with as default value:

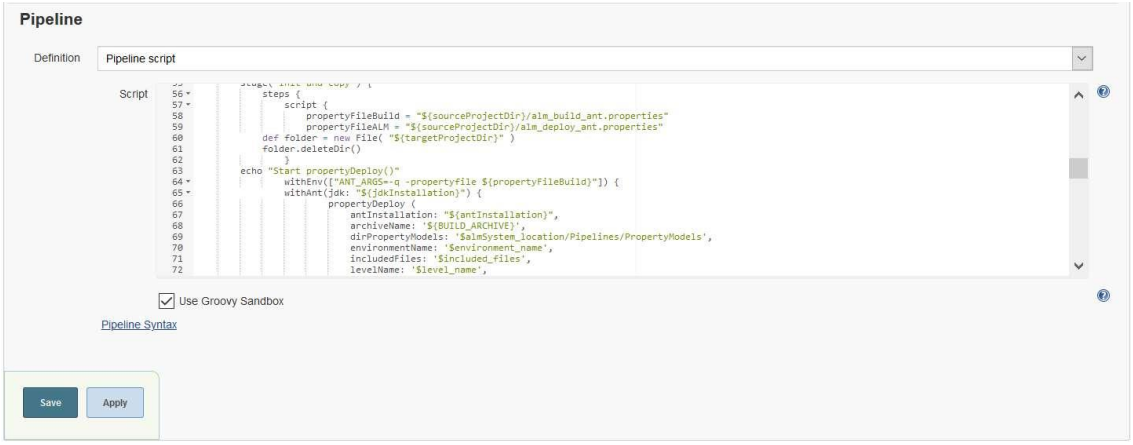
Jenkins-Saleforce_branch-Archive_NN.

This BUILD_ARCHIVE parameter is equal to the BUILD_TAG parameter. NN is the tag number you will use to deploy

You can find what parameters to add in the almPipeline_sfDeploy.groovy Groovy script. Search for “Parameters of the Jenkins Project”

```
// Parameters of the Jenkins Project
// project_name = 'MyProject'
// branch_name = 'trunk'
// BUILD_ARCHIVE = 'build [BUILD_TAG]'
```

Next install the pipeline script with the content of the almPipeline_sfDeploy.groovy file.



Make sure you have the right values for your SalesForce for:

```
withEnv(["ANT_ARGS=-q -propertyfile ${propertyFileBuild}"]) {
    withAnt(jdk: "${jdkInstallation}") {
        propertyDeploy (
            antInstallation: "${antInstallation}",
            archiveName: '${BUILD_ARCHIVE}',
            dirPropertyModels: '$almSystem_location/Pipelines/PropertyModels',
            environmentName: '$environment_name',
            includedFiles: '$included_files',
            levelName: '$level_name',
            propertyFileALM: "${propertyFileALM}",
            propertyFileBuild: "${propertyFileBuild}",
            propertyFileSource: '$sourceProjectDir/$project_name.properties',
            propertyModel: 'sfdeploy_ant',
            sourceProjectDir: '$sourceProjectDir',
            targetProjectDir: '$targetProjectDir'
```

Appendix one: Description for each script and the parameters you need to set

Salesforcebulkretrieve

Description: Bulk Retrieve Metadata objects from Salesforce server

Parameters:

sf.active	Saleforce server is active or not.
sf.serverurl	Server Url (Useful for working against the sandbox instance on test.Salesforce.com)
sf.username	Saleforce User Name for login. (Required if no sessionId)
sf.password	Saleforce User Password for login. With a security token, paste the 25-diGit token value to the end of your password. (Required if no sessionId)
sf.sessionId	The ID of an active Salesforce session. (Required if no username and password)
sf.retrieveTarget	Root of the directory structure where metadata files are retrieved.
sf.metadataType	Name of metadata type to retrieve. (See the Metadata API Developer's Guide for more information)
sf.containingFolder	If the metadata is contained in a folder, the containingFolder should be the name of this folder.
sf.batchSize	The number of items to retrieve while doing multi-part retrieve.
sf.apiVersion	The API version to use for the retrieved metadata files.
sf.maxPoll	The number of times to poll Salesforce for the results of the task.
sf.unzip	If true, the retrieved components are unzipped else a zip file is created.
sf.trace	Prints the SOAP requests and responses.
dir.sfModels	Models Location for sf_retrieveBulkFolder_xml.model file

SalesforceDeploy

Description: Deploy Metadata Package(s) to Salesforce server

Parameters:

sf.active	Saleforce server is active or not.
sf.serverurl	Server Url (Useful for working against the sandbox instance on test.Salesforce.com)
sf.username	Saleforce User Name for login. (Required if no sessionId)
sf.password	Saleforce User Password for login. With a security token, paste the 25-diGit token value to the end of your password. (Required if no sessionId)
sf.sessionId	The ID of an active Salesforce session. (Required if no username and password)
sf.pollWaitMillis	The number of milliseconds to wait between each poll of Salesforce to retrieve the results of the deploy.
sf.checkOnly	Only check the validity of the deployed files.
sf.maxPoll	The number of times to poll Salesforce for the results of the task.
sf.purgeOnDelete	Delete components in the destructiveChanges.xml manifest file.
sf.deployRoot	Root of the directory tree of files to deploy. (Required if no zipFile value)
sf.zipFile	Path of the metadata zip file to be deployed. (Required if no deployRoot value)
sf.singlePackage	The deployRoot/zipFile contains a single package directory structure, instead of a set of packages.
sf.manifestFile	Name of the manifest file if singlePackage=true
sf.allowMissingFiles	Success if files that are specified in package.xml are not in the zip file.
sf.autoUpdatePackage	Success if files that are in the zip file are not specified in package.xml.
sf.rollbackOnError	Indicates whether any failure causes a complete rollback or not.
sf.ignoreWarnings	Indicates that a deployment should succeed even if there are warnings (true) or that one or more warnings will cause the deployment to fail and roll back (false).
testLevel.options	sf.testLevel values available (NoTestRun, RunSpecifiedTests, RunLocalTests, RunAllTestsInOrg)
sf.testLevel	Type of tests to run as part of deployment. (See testLevel.options)
sf.specifiedTests	List (separated by comma) of Apex classes containing tests run after deploy if testLevel is equal to RunSpecifiedTests.
logType.options	sf.logType values available (None, Debugonly, Db, Profiling, Callout, Detail)
sf.logType	The debug logging level for tests. (See logType.options)
sf.trace	Prints the SOAP requests and responses.
dir.sfModels	Models Location for sf_deployRoot_xml.model, sf_deployZip_xml.model files
deployProperties	Property file to use for adding the RequestId.

SalesforceDeployCancel

Description: Cancel Deploy running in Salesforce server

Parameters:

sf.active	Saleforce server is active or not.
sf.serverurl	Server Url (Useful for working against the sandbox instance on test.Salesforce.com)
sf.username	Saleforce User Name for login. (Required if no sessionId)
sf.password	Saleforce User Password for login. With a security token, paste the 25-diGit token value to the end of your password. (Required if no sessionId)
sf.sessionId	The ID of an active Salesforce session. (Required if no username and password)
sf.pollWaitMillis	The number of milliseconds to wait between each poll of Salesforce to retrieve the results of the deploy.
sf.maxPoll	The number of times to poll Salesforce for the results of the task.
sf.trace	Prints the SOAP requests and responses.
dir.sfModels	Models Location for sf_deployCancel_xml.model file
deployProperties	Property file to use for reading the RequestId.

SalesforceDeployRecentValidation

Description Deploy Recent Validation in Salesforce server

Parameters:

sf.active	Saleforce server is active or not.
sf.serverurl	Server Url (Useful for working against the sandbox instance on test.Salesforce.com)
sf.username	Saleforce User Name for login. (Required if no sessionId)
sf.password	Saleforce User Password for login. With a security token, paste the 25-diGit token value to the end of your password. (Required if no sessionId)
sf.sessionId	The ID of an active Salesforce session. (Required if no username and password)
sf.recentValidationId	Recent Validation ID from last 4 days.
sf.rollbackOnError	Indicates whether any failure causes a complete rollback or not.
sf.pollWaitMillis	The number of milliseconds to wait between each poll of Salesforce to retrieve the results of the deploy.
sf.maxPoll	The number of times to poll Salesforce for the results of the task.
sf.trace	Prints the SOAP requests and responses.
dir.sfModels	Models Location for sf_deployRecentValidation_xml.model file

SalesforceDescribeMetadata

Description: Describe Metadata types from Salesforce server

Parameters:

sf.active	Saleforce server is active or not.
sf.serverurl	Server Url (Useful for working against the sandbox instance on test.Salesforce.com)
sf.username	Saleforce User Name for login. (Required if no sessionId)
sf.password	Saleforce User Password for login. With a security token, paste the 25-diGit token value to the end of your password. (Required if no sessionId)
sf.sessionId	The ID of an active Salesforce session. (Required if no username and password)
sf.apiVersion	The API version to use for the retrieved metadata files.
sf.resultFilePath	Path of the output file.
display.resultFile	Prints the result file in the Ant log.
dir.sfModels	Models Location for sf_describeMetadata_xml.model file

SalesforceListMetadata

Description: List Metadata components from Salesforce server

Parameters:

sf.active	Saleforce server is active or not.
sf.serverurl	Server Url (Useful for working against the sandbox instance on test.Salesforce.com)
sf.username	Saleforce User Name for login. (Required if no sessionId)
sf.password	Saleforce User Password for login. With a security token, paste the 25-diGit token value to the end of your password. (Required if no sessionId)
sf.sessionId	The ID of an active Salesforce session. (Required if no username and password)
sf.apiVersion	The API version to use for the retrieved metadata files.
sf.metadataType	Name of metadata type to retrieve. (See the Metadata API Developer's Guide for more information)

sf.folder	The folder associated with the component. Required for components that use folders.
sf.resultFilePath	Path of the output file.
sf.trace	Prints the SOAP requests and responses.
display.resultFile	Prints the result file in the Ant log.
dir.sfModels	Models Location for sf_describeMetadata_xml.model file

SalesforceRetrieve

Description: Retrieve Metadata components from Salesforce server

Parameters:

sf.active	Saleforce server is active or not.
sf.serverurl	Server Url (Useful for working against the sandbox instance on test.Salesforce.com)
sf.username	Saleforce User Name for login. (Required if no sessionId)
sf.password	Saleforce User Password for login. With a security token, paste the 25-diGit token value to the end of your password. (Required if no sessionId)
sf.sessionId	The ID of an active Salesforce session. (Required if no username and password)
sf.apiVersion	The API version to use for the retrieved metadata files.
sf.retrieveTarget	Root of the directory structure where metadata files are retrieved.
sf.packageNames	List (comma-separated) of the package's names to retrieve. (Required if unpackaged is empty)
sf.pollWaitMillis	The number of milliseconds to wait between each poll of Salesforce to retrieve the results of the deploy.
sf.maxPoll	The number of times to poll Salesforce for the results of the task.
sf.singlePackage	The deployRoot/zipFile contains a single package directory structure, instead of a set of packages.
sf.unpackaged	The name of a manifest file that specifies the components to retrieve. (Required if packageNames is empty)
sf.unzip	If true, the retrieved components are unzipped else a zip file is created.
sf.trace	Prints the SOAP requests and responses.
dir.sfModels	Models Location for sf_describeMetadata_xml.model file