IKAN

Integrating Jenkins/CloudBees with z/OS mainframe

Cost-effective and easy to implement Enterprise-wide DevOps (with CI/CD) for mainframes

Table of contents

Summary

This technical document is intended for developers, technical people, mainframe or non-mainframe experts, and software architects.

Jenkins is the leading open source automation server and provides hundreds of plugins to support building, deploying and automating any project.

CloudBees offers support and a fully tested version of Jenkins that will be more stable. CloudBees also adds a number of features to Jenkins that brings a lot from an enterprise manageability, scalability, and security standpoint.

IKAN adds a plugin that allows you to manage IBM z/OS mainframes using Jenkins or CloudBees

This document aims at explaining how, by using the IKAN Plugin, you can manage your application life-cycle, and how you can easily deploy the developed applications on the mainframe.

We will describe in detail how IKAN works and what the different tasks are for Users and Administrators.

We are confident that after having read this document, you will be convinced of the enormous advantages of putting in place our IKAN solution.

If you would still have questions, do not hesitate to contact our support team at info@ikanalm.com.

IKAN Pipelines for z/OS: The Solution for the Mainframe

In the following section, we will explain more in detail how Jenkins/CloudBees handles the lifecycle to compile and deploy your applications on the mainframe.

Today, traditional mainframe development is already often enhanced with Eclipse-based development to address the requirements of modern development. The main issue when combining mainframe and distributed development, is how to deploy the developed applications on the mainframe.

We will cover 2 points of view:

Cloudbees User

A User creates/updates packages to compile and deploy (promote) one or more programs he developed.

Cloudbees Administrator

The initial CloudBees setup for the mainframe and the project setup are done by the CloudBees Administrator.

CloudBees – User's Point of View

Once the initial setup has been done and the IKAN Pipelines for z/OS scripts have been installed by the CloudBees Administrator, a User can start using CloudBees to create his projects.

Basically, a User can create a Compile/Build or Deploy action (a "Project" in CloudBees terminology) in 4 steps:

1. Create or update a package and put it in the VCR project or in a CloudBees machine's folder
2. Put the "Checkout" Pipeline for z/OS in the VCR project
3. Log on to CloudBees and display the Dashboard Overview
4. Define the projects: Checkout, Build, Deploy, etc.
5. Select the appropriate (sub)Project (Checkout, Compile/Build or Deploy)
6. Create the Job with parameters

Once the Job is created, a series of information screens will be available to provide additional information on the requested action, allowing following up its status and the results.

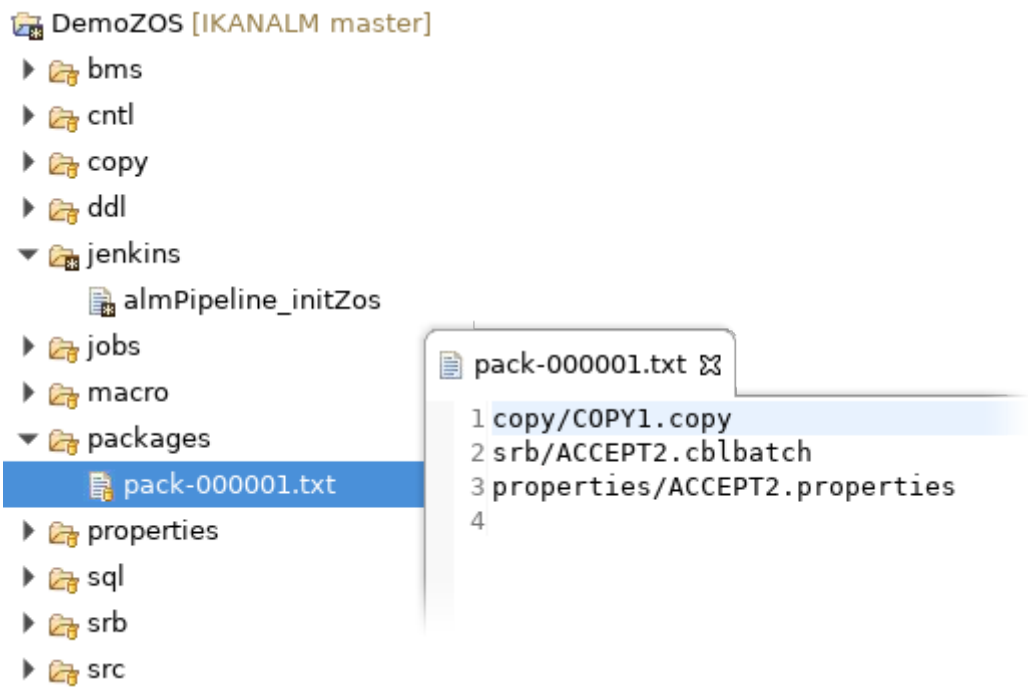
Step 1

Create or update a package in the VCR Project

A Package allows moving one or more individual files together. For z/OS this means that you can group one or more source files to compile or deploy together.

As an alternative you can use the included_files parameter in the “Checkout” project. Here you give the name(s) of the files you want to compile or deploy together

Working with packages is the preferred way of working. We suggest to create a packages folder where you add each package file as pack-[nnnnnn].txt. Other definitions are possible, but you will have to review and adapt this default definition in the pipelines.



Eclipse (Git) DemoZOS project and package file management

packageFile Usage

If you use a packageFile parameter in the script instead of the included_files parameter to select components, you need to review the path and the Package file name you want to use.

The packageFile variable is defined in each Groovy pipeline file installed as a Pipeline and uses the PACK_NUMBER variable.

almPipeline_initZos

```
def antInstallation = 'ANT-1.9.3'
def jdkInstallation = 'jdk1.8.0_202_x64'

pipeline {
    // specify agents/nodes
    agent any

        // Parameters of the Jenkins Configuration
        // almSystem_location = '/home/Jenkins/almPluginPhases'

    // define scripting tools
    tools {
        ant "${antInstallation}"
        jdk "${jdkInstallation}"
    }

    environment {
        // Parameters of the Jenkins Project
        // project_name = 'MyProject'
        // PACK_NUMBER = '000001'
        // included_files = '**/*.*'
        sourceProjectDir = "$project_name"
        targetProjectDir = "$almSystem_location/environments/build/source/$project_name/$PACK_NUMBER"
    }
    stages {
        stage('Create Package') {
            steps {
        echo pwd()
        echo "JAVA_HOME=${env.JAVA_HOME}"
                script {
                    propertyFileBuild = "${sourceProjectDir}/${env.project_name}.properties"
                    propertyFileALM = "${targetProjectDir}/alm_initial_ant.properties"
                    packageFile = "${sourceProjectDir}/packages/pack-${PACK_NUMBER}.txt"
                    if (isUnix()) {
```

"packageFile" pipeline definition

I.e. packageFile = "${sourceProjectDir}/packages/pack-${PACK_NUMBER}.txt"

The sourceProjectDir variable is the location of the exported project components. For a GIT project, this parameter may be omitted.

If you want to change the packageFile definition, you have to do that in the other Groovy pipelines. The packageFile contains the files to select, one file per line, with their folder as follows:
src/program1.cbl
copy/copy1.cpy
...

Step 2

Put the “Checkout” pipeline for z/OS in the VCR project

This “almPipeline_zosInit.groovy” pipeline can be found in the “Jenkins Plugin for z/OS” archive file. See your Administrator for more info on this.

almPipeline_initZos

```
def antInstallation = 'ant1.9.4'
def jdkInstallation = 'java-1.8.0'

pipeline {
    // specify agents/nodes
    agent any

        // Parameters of the Jenkins Configuration
        // almSystem_location = '/home/Jenkins/almPluginsBase'

    // define scripting tools
    tools {
        ant "${antInstallation}"
        jdk "${jdkInstallation}"
    }

    environment {
        // Parameters of the Jenkins Project
        // project_name = 'MyProject'
        // PACK_NUMBER = '000001'
        // included_files = '**/*.*'
        sourceProjectDir = "$project_name"
        targetProjectDir = "$almSystem_location/environments/build/source/$project_name/$PACK
    }
```

“Checkout” pipeline definition

Change the antInstallation and jdkInstallation parameters with your CloudBees Global tools definitions. Look at the packageFile definition if necessary.

Note that script commands: def folder and folder.deleteDir() will activate CloudBees security for Java files. Please, approve these commands or replace them with a machine OS command as:

Signatures already approved:

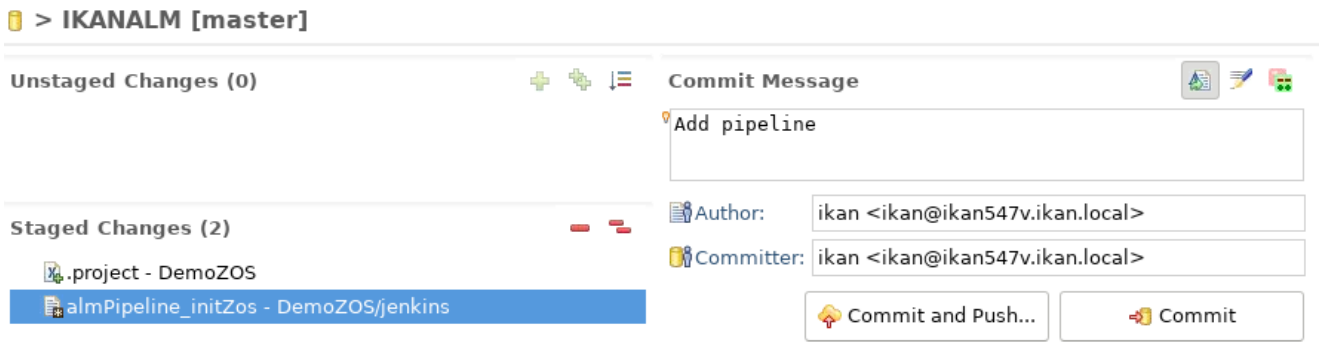
```
new java.io.File java.lang.String
staticMethod org.codehaus.groovy.runtime.DefaultGroovyMethods deleteDir
java.io.File
staticMethod org.codehaus.groovy.runtime.DefaultGroovyMethods
getProperties java.lang.Object
```

CloudBees Approval screen

```
if (isUnix()) {
       sh "rm -rf ${targetProjectDir}"
} else {
       bat "start cmd.exe /C rmdir /S /Q ${targetProjectDir}"
}
```

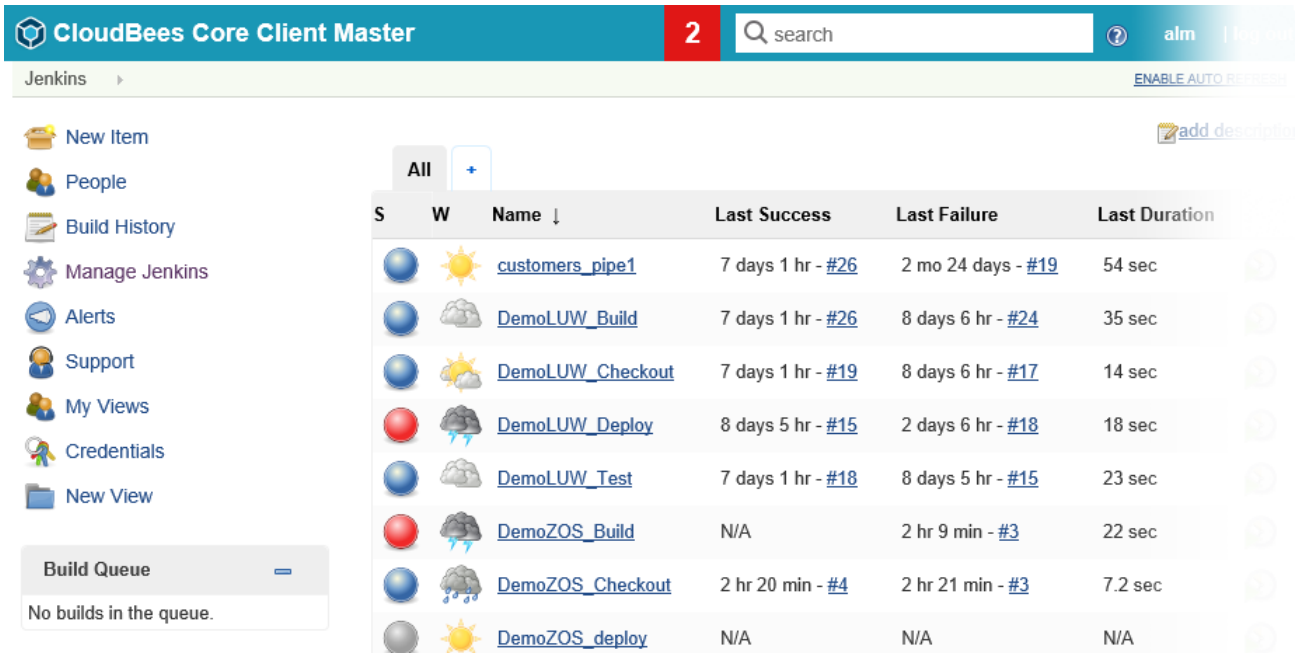
Example of what to change in the Pipeline change, if not Java approved

Finally, commit these changes to your VCR system.



Step 3

Log on and display the Dashboard Overview



Desktop Overview for the z/OS project

The following basic information will be displayed: Main menu: New Item, Manage Jenkins, etc.

Dashboard of projects:

- Name (as link for details) with it Status
- Latest successful Job: shows the latest successful Job number
- Latest failure: shows the latest failure Job number.
- Last duration: shows the last Job time
- Build: the available action icon for the Project (Level). When clicking the Build icon, a Job will be started.

The z/OS project (DemoZOS) we use here as an example is a package-based project for which the following Levels have been defined: a Checkout Project Level, a Build Project Level and a Deploy Project Level. Those Levels are showing the Lifecycle of the global project. Later, it will be possible to merge the Checkout, Build and Deploy projects as one project. We have two option to build or deploy projects: package based or release based.

For mainframe users, a project is best package-based. A package allows the User to select one or several components of a Project which should be built and deployed together, ignoring the other Project Stream components. Such a package has to stay coherent for building and deploying.

A Package (of components) will always be linked to one Project and it will always follow that Project lifecycle. Also, a package has to progress with the Project lifecycle independently of possible other packages linked to the same Project.

For distributed release-based projects, on the other hand, all components are built and deployed together.

Once the User has defined the package, he can start building/compiling and, next, promoting or deploying his programs.

Step 4

Create the Checkout Project

The first step is to run the “Checkout” project. This one will extract the project files. See the Administrator point of view on how to install a project.

To start a checkout for the project in our example, the User would click the appropriate Project button at the Checkout Level. Next, the Checkout parameters screen will be displayed.

Pipeline DemoZOS_Checkout

This build requires parameters:

project_name: DemoZOS
Real project name

PACK_NUMBER: 000001
Package number to use from VCR project

included_files: **/*.*
Selection criteria (dont change if packageFile selection

Build

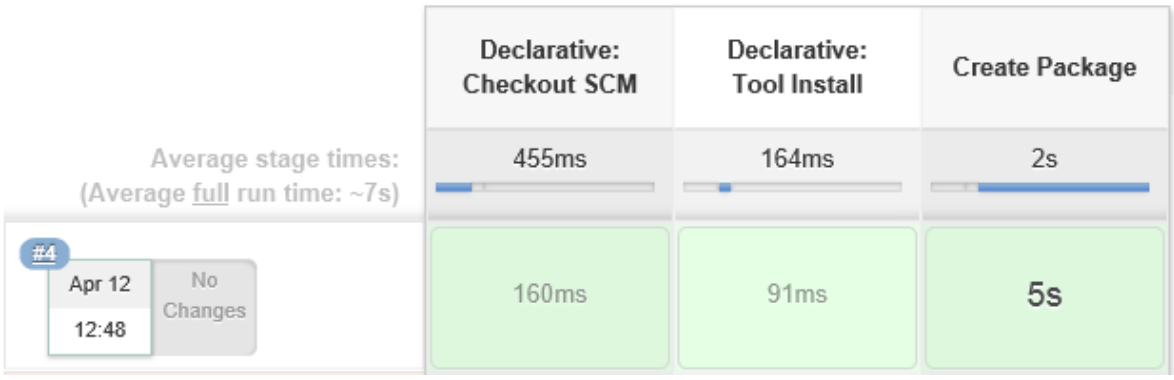
Project parameters for the pipeline Steps

Make sure the project name and the PACK_NUMBER are correct. By clicking the Build button, the Checkout Project Job will be created and the process will start.

Pipeline DemoZOS_Checkout

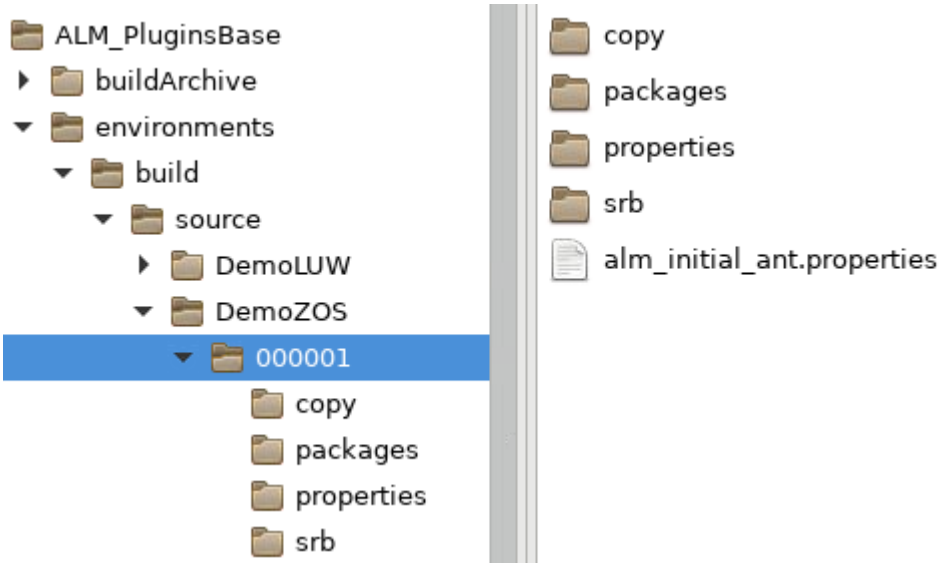
Demo ZOS Checkout before build

Stage View



Checkout the Project and create the Package selection by the z/OS pipeline Steps

In CloudBees, the selected files have been copied to the “targetProjectDir’ target location.

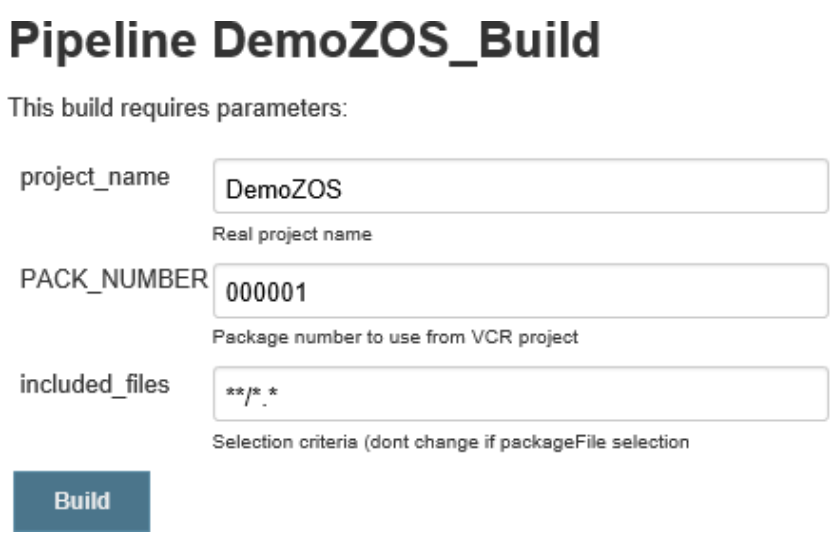


Package Files from the Project selected by the z/OS pipeline Steps

Step 5

Create the Build Project

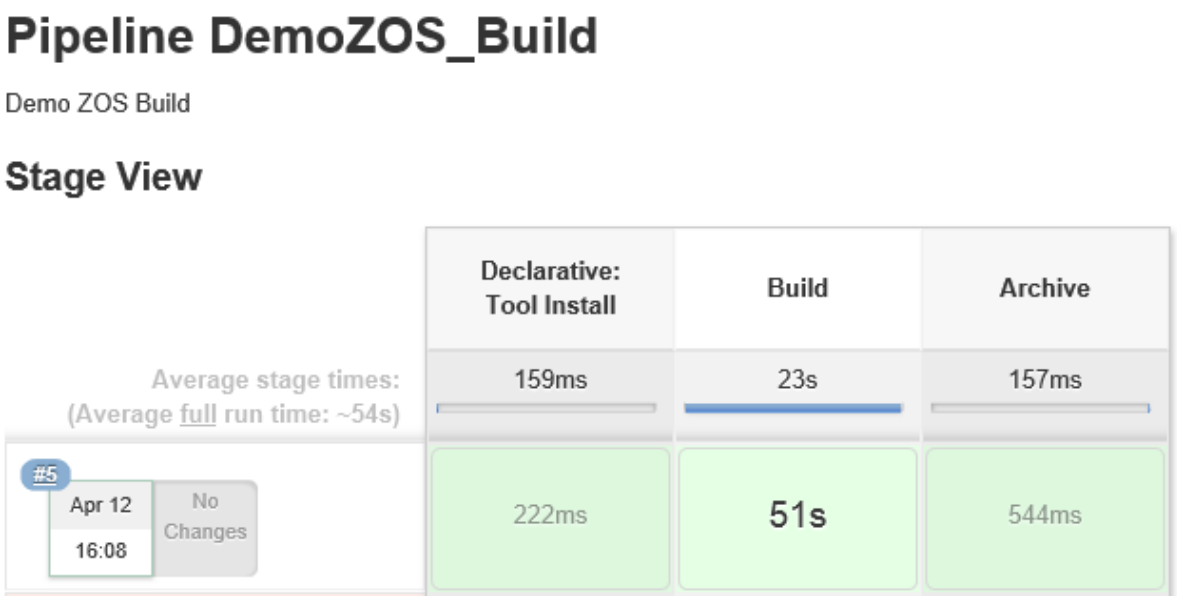
To start a compile, click the appropriate Project button at the Build Level. Next, the Build screen will be displayed. See the Administrator point of view for installing this project. To know how this project is installed, see you Administrator.



Project parameters for the z/OS pipeline Steps

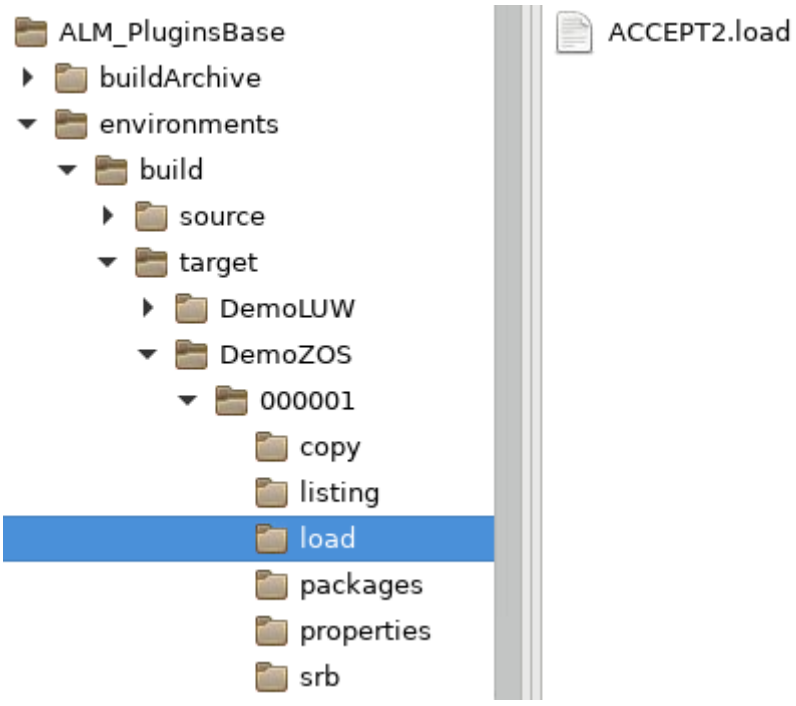
On this screen, the User can enter the parameters linked to the Project Job and, if configured that way, change some parameter values. Make sure the project name and the PACK_NUMBER are correct.

By clicking the Build button, the Build Project Job will be created and the process will start.



Build Package Files from the Project with the z/OS pipeline Steps

The selected files have been copied to the next target location (defined in the 'zosBuild' pipeline) and the resulting files of this compilation are put in the target folder for archiving.



Package Files with files generated by the z/OS pipeline Steps of compilation

In fact, that is all a User needs to do to compile all the programs in his package: go to the Desktop, click the appropriate action button and create a Level Request to execute a Build/Compile or a Deploy.

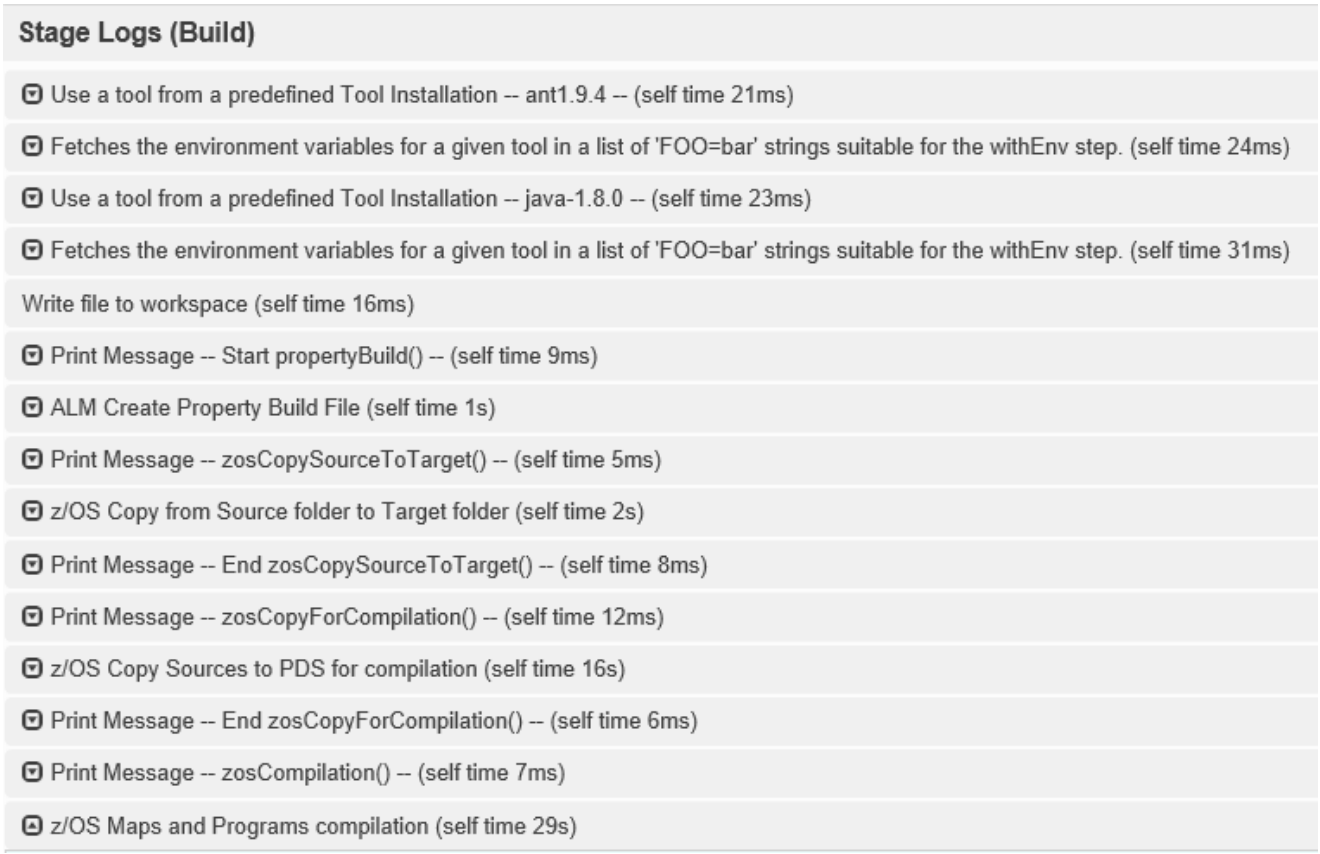
Additional information provided by CloudBees

Besides creating a Project Job, CloudBees also provides additional information to a User:

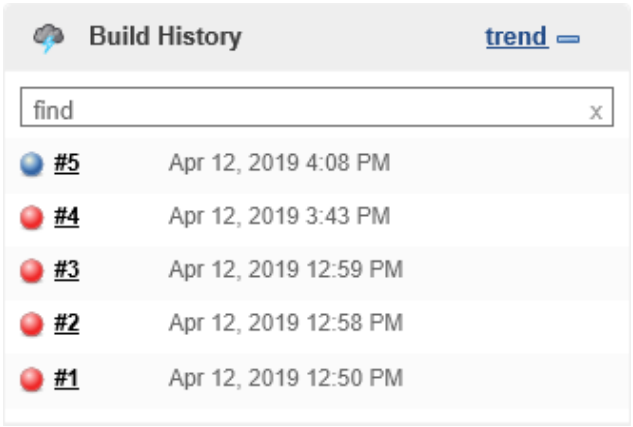
- An overview of each Job.
- For each Job: detailed information through the interface.

On the following pages we will show some sample CloudBees screens with more detailed information, such as the detailed Project Job information, the Build Log.

Finally, we will also show what happens behind the scene on the mainframe.



Project Build Job: Build Detailed Overview



Project Build Job: History

```
[com.ikanalm.plugins.jenkins.zoscompilation] $ /opt/javatools/ant1.9.4/bin/ant -file zosCompilation.xm
l -DpropertyFileALM=/home/ikan/Jenkins/ALM_PluginsBase/environments/build/target/DemoZOS/000001/alm_bui
ld_ant.properties -Dscript.syslin=/home/ikan/Jenkins/ALM_PluginsBase/PhaseScripts/linkEditSyslin.xml -D
dir.zosResources=/home/ikan/Jenkins/ALM_PluginsBase/Machine/PhaseResources -DpropertyFileSource=/home/i
kan/Jenkins/ALM_PluginsBase/environments/build/source/DemoZOS/000001/DemoZOS.properties -DstopOnErrorCo
mpile=true -Dpropsfile.languages=/home/ikan/Jenkins/ALM_PluginsBase/Machine/PhaseResources/BUILD/langua
gesZOS.properties -Danalyze.source=false -Dproject.objtypes=ALL -Ddir.jeslogs=/home/ikan/Jenkins/ALM_Pl
uginsBase/environments/build/target/DemoZOS/000001/logs -Dpropsfile.objtypes=/home/ikan/Jenkins/ALM_Plu
ginsBase/Machine/PhaseResources/globalObjtypes.properties -Dpropsfile.environment=/home/ikan/Jenkins/AL
M_PluginsBase/Machine/PhaseResources/BUILD/environment_build.properties -Dpropsfile.parmsFTP=/home/ika
n/Jenkins/ALM_PluginsBase/Machine/PhaseResources/parmsFTPZOS.properties -DallocPDS=true -Ddir.zosModels
=/home/ikan/Jenkins/ALM_PluginsBase/Machine/PhaseModels/BUILD -Dsave.jeslogs=false -Ddir.licensetool=/h
ome/ikan/Jenkins/ALM_PluginsBase/PhaseTools/config
     [echo] ALM License found!
     [echo]  WARNING: No /home/ikan/Jenkins/ALM_PluginsBase/environments/build/source/DemoZOS/000001/De
moZOS.properties file found for this project.
     [echo]         Default project properties are used.
     [echo] Level: BUILD Config: ZOS - Action: Deliver Build START
     [echo] JOB execution  15.09.05 JOB00067 ---- FRIDAY,    12 APR 2019 ----
     [echo] File ACCEPT2 was successful on ZOS.
     [echo]  *** COMPILE ZOS for ACCEPT2 submitted to z/OS with success ***
     [echo] Generated Member files are copied from ZOS.
     [echo] DSNs IKANALM.DEMOS.P0000001... are deleted on ZOS.

BUILD SUCCESSFUL
Total time: 29 seconds
```

Build compilation Log

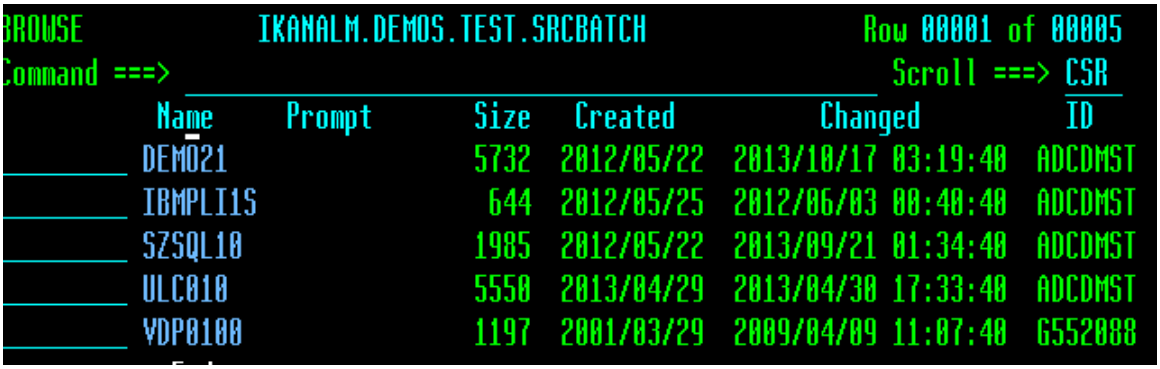
```
zosCompilation()
[Pipeline] withEnv
[Pipeline] {
[Pipeline] withAnt
[Pipeline] {
[Pipeline] zosCompilation
[com.ikanalm.plugins.jenkins.zoscompilation] $ /opt/javatools/ant1.9.4/bin/ant -file zosCompilation.xml
-DpropertyFileALM=/home/ikan/Jenkins/ALM_PluginsBase/environments/build/target/DemoZOS/000001/alm_build_ant.properties
-Dscript.syslin=/home/ikan/Jenkins/ALM_PluginsBase/PhaseScripts/linkEditSyslin.xml
-Ddir.zosResources=/home/ikan/Jenkins/ALM_PluginsBase/Machine/PhaseResources
-DpropertyFileSource=/home/ikan/Jenkins/ALM_PluginsBase/environments/build/source/DemoZOS/000001/DemoZOS.properties
-DstopOnErrorCompile=true
-Dpropsfile.languages=/home/ikan/Jenkins/ALM_PluginsBase/Machine/PhaseResources/BUILD/languagesZOS.properties
-Danalyze.source=false -Dproject.objtypes=ALL
-Ddir.jeslogs=/home/ikan/Jenkins/ALM_PluginsBase/environments/build/target/DemoZOS/000001/logs
-Dpropsfile.objtypes=/home/ikan/Jenkins/ALM_PluginsBase/Machine/PhaseResources/globalObjtypes.properties
-Dpropsfile.environment=/home/ikan/Jenkins/ALM_PluginsBase/Machine/PhaseResources/BUILD/environment_build.properties
-Dpropsfile.parmsFTP=/home/ikan/Jenkins/ALM_PluginsBase/Machine/PhaseResources/parmsFTPZOS.properties -DallocPDS=true
-Ddir.zosModels=/home/ikan/Jenkins/ALM_PluginsBase/Machine/PhaseModels/BUILD -Dsave.jeslogs=false
-Ddir.licensetool=/home/ikan/Jenkins/ALM_PluginsBase/PhaseTools/config
     [echo] ALM License found!
     [echo]  WARNING: No /home/ikan/Jenkins/ALM_PluginsBase/environments/build/source/DemoZOS/000001/DemoZOS.properties file
found for this project.
     [echo]         Default project properties are used.
     [echo] Level: BUILD Config: ZOS - Action: Deliver Build START
     [echo] JOB execution  15.09.05 JOB00067 ---- FRIDAY,    12 APR 2019 ----
     [echo] File ACCEPT2 was successful on ZOS.
     [echo]  *** COMPILE ZOS for ACCEPT2 submitted to z/OS with success ***
     [echo] Generated Member files are copied from ZOS.
     [echo] DSNs IKANALM.DEMOS.P0000001... are deleted on ZOS.

BUILD SUCCESSFUL
Total time: 29 seconds
[Pipeline] }
[Pipeline] // withAnt
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] echo
End zosCompilation()
```

Build compilation Console Output Trace

What happens behind the scene?

The following z/OS screens show the corresponding actions on the mainframe. The first z/OS Phase copies the copy-book(s) and the programs (and, if existing, also the special components for compiling) to the z/OS environment.

The following screen shows the files that have been collected from the VCR and that are transferred via FTP to the main-frame in a PDS with IKAN ALM.DEMOS.TEST.SRCBATCH as PDS name.



DEMO21 project

After this FTP Phase, the second phase, Z/OS program compile, is executed.

This phase creates the JCL (See Appendix V: Sample of z/OS compilation JCL), transfers that JCL via FTP to the mainframe and submits the job.

The results will also be available in the IKAN ALM Phase log. That log will list all executed events, step by step, and will tell if the compile has been executed successfully or not.

The following screen shows the JCL jobs that have been submitted and that are executed.

```
 Display  Filter  View  Print  Options  Help
-------------------------------------------------------------------------------
SDSF STATUS DISPLAY ALL CLASSES                 LINE 1-20 (43)
COMMAND INPUT ===>                                SCROLL ===> CSR
PREFIX=*  DEST=(ALL)  OWNER=ADCDMST  SYSNAME=
NP   JOBNAME  JobID    Owner    Prty Queue     C  Pos  Max-RC     SAff  ASys Status          PrtDest
_    ADCDMSTC JOB00135 ADCDMST     5 EXECUTION  A                      SYS1                 LOCAL
_    ADCDMST  TSU00134 ADCDMST    15 EXECUTION                   SYS1  SYS1                 LOCAL
     ADCDMSTC JOB00081 ADCDMST     1 PRINT      A    15 JCL ERROR                          LOCAL
     ADCDMSTC JOB00082 ADCDMST     1 PRINT      A    16 JCL ERROR                          LOCAL
```

SDSF Status Display

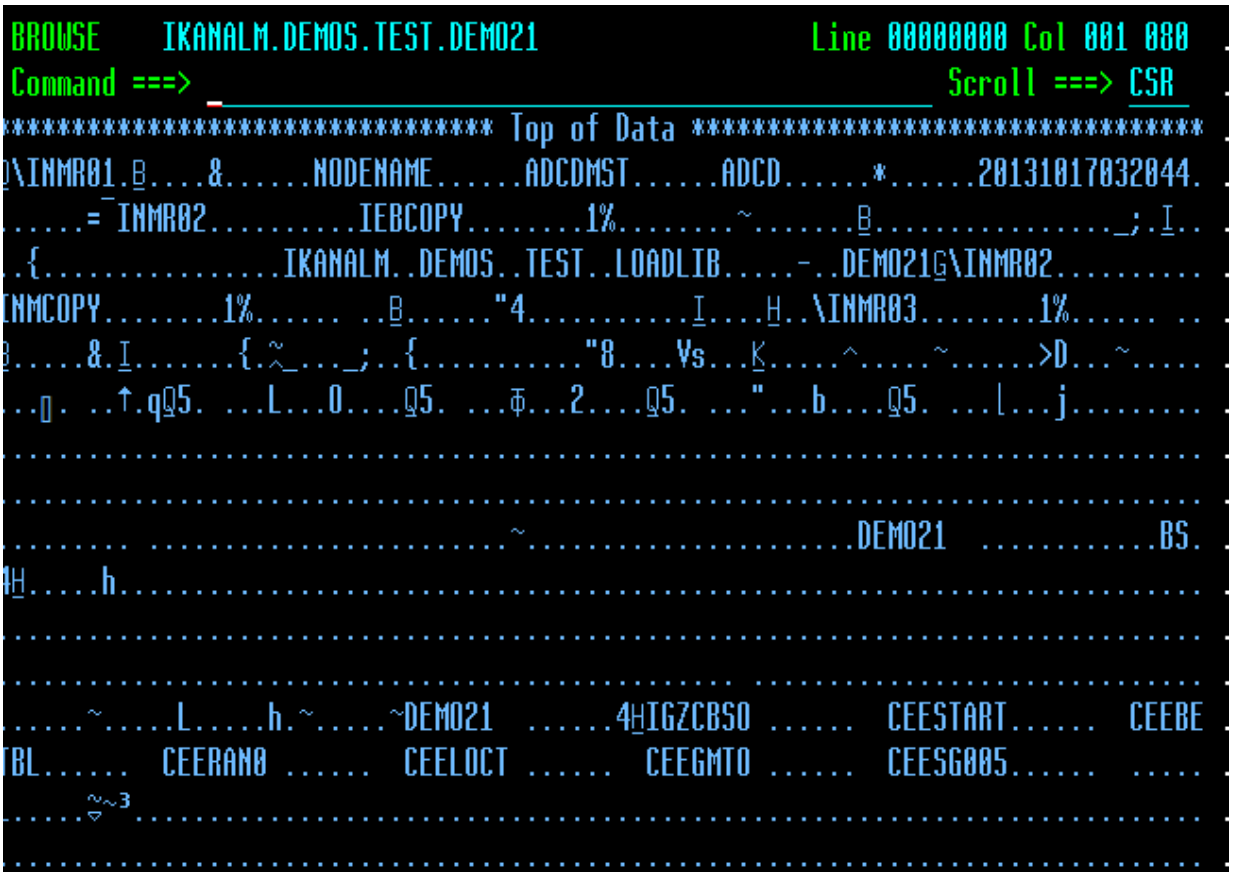
The following screen shows the result of the execution on the mainframe:

```
SDSF OUTPUT DISPLAY ADCDMSTC JOB00135  DSID     2 LINE 0       COLUMNS 02- 133
COMMAND INPUT ===> _                                          SCROLL ===> CSR
******************************** TOP OF DATA ********************************************************************************
                    J E S 2  J O B  L O G  --  S Y S T E M  S Y S 1  --  N O D E  N 1

21.20.26 JOB00135 ---- WEDNESDAY, 16 OCT 2013 ----
21.20.26 JOB00135  IRR010I  USERID ADCDMST  IS ASSIGNED TO THIS JOB.
21.20.28 JOB00135  ICH70001I ADCDMST  LAST ACCESS AT 21:20:24 ON WEDNESDAY, OCTOBER 16, 2013
21.20.28 JOB00135  $HASP373 ADCDMSTC STARTED - INIT 1    - CLASS A - SYS SYS1
21.20.28 JOB00135  IEF403I ADCDMSTC - STARTED - TIME=21.20.28
21.20.29 JOB00135  -                                          --TIMINGS (MINS.)--          -----PAGING COUNTS----
21.20.29 JOB00135  -STEPNAME PROCSTEP    RC   EXCP   CONN    TCB    SRB  CLOCK   SERV  WORKLOAD  PAGE  SWAP   VIO SWAPS
21.20.29 JOB00135  -COPYSRC             00     93      0    .00    .00     .0   1441  BATCH        0     0   117     0
21.20.41 JOB00135  -COBOL               04   1630      0    .15    .00     .1  73427  BATCH        0     0  1109     0
21.20.41 JOB00135  -ALLOCLCT            00     15      0    .00    .00     .0    144  BATCH        0     0     2     0
21.20.41 JOB00135  -CREATLCT            00     61      0    .00    .00     .0    729  BATCH        0     0     3     0
21.20.41 JOB00135  -COPYLCT             04     58      0    .00    .00     .0   1894  BATCH        0     0     0     0
21.20.43 JOB00135  -LKEDT               04    263      0    .01    .00     .0   5718  BATCH        0     0     9     0
21.20.44 JOB00135  -CLEARSEQ            00     11      0    .00    .00     .0    107  BATCH        0     0     0     0
21.20.47 JOB00135  -XMITLOAD            00    655      0    .03    .00     .0  16543  BATCH        0     0     0     0
21.20.48 JOB00135  -PRNTCOMP            00    396      0    .01    .00     .0   7209  BATCH        0     0     0     0
21.20.48 JOB00135  $HASP375 ADCDMSTC ESTIMATED  LINES EXCEEDED
21.20.48 JOB00135  -PRNTLINK            00     61      0    .00    .00     .0    775  BATCH        0     0     0     0
21.20.49 JOB00135  -FRMTLKD             00     45      0    .00    .00     .0   1174  BATCH        0     0     0     0
```

SDSF Output Display

When the Job is completed with success, the listing and the load module are transferred via FTP to the target environment.

The following screen shows the sequential file generated by the Xmit step for transfering the load module to the target environment.



Sequential Xmit file

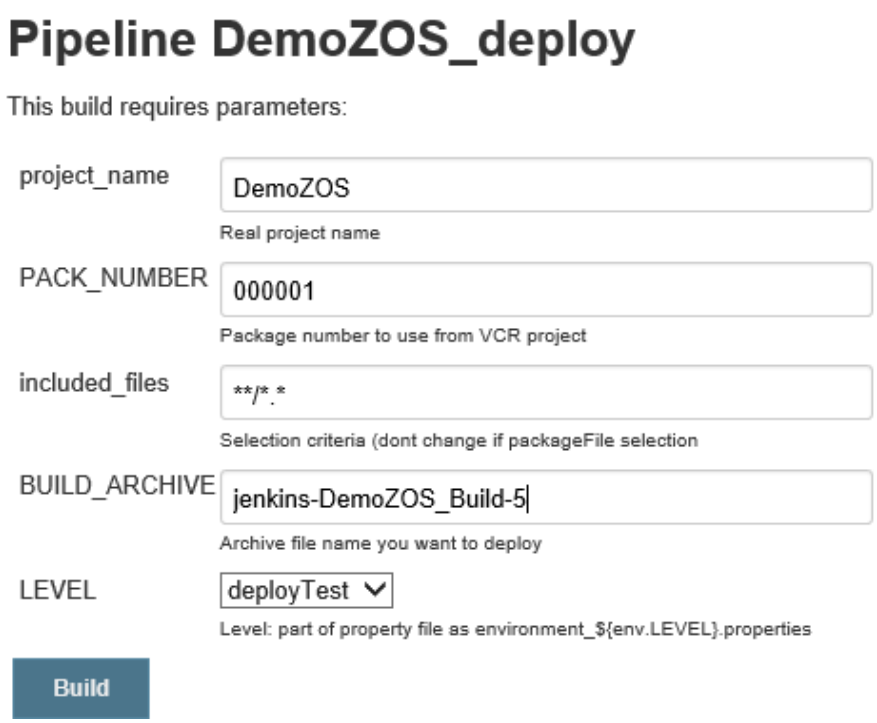
As a result, CloudBees has the listing and the load module in his archive. At this point, CloudBees is in full control of the remainder of the steps in the lifecycle.

Build Archive installed in the machine by the z/OS pipeline Step

For archiving standard folders may be used or you can use something like Artifactory.

From the CloudBees archive, the load module can be deployed or promoted to a test or production level by simple starting a Project Deploy Job that executes a receive step.

The following screen starts the Deploy.



Deploy Package Files parameters

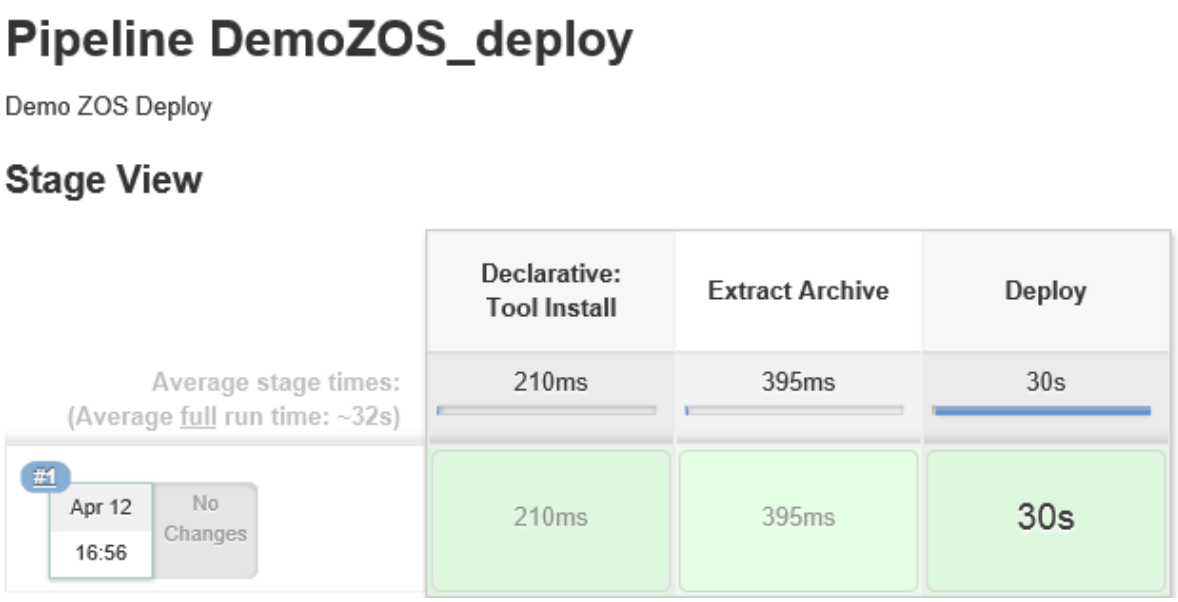
They are 2 new parameters for deploying: BUILD_ARCHIVE (build file to deploy) and LEVEL (z/OS file with parameters for the z/OS level). For the Deploy, CloudBees is using the same process as for a Compile/Build:

- A Deploy Level and Environment with its related z/OS scripts must be created. The z/OS steps defined here are:
  - The promote (FTP) of load-modules and other components to the mainframe
  - Delete obsolete files and associated components (such as load modules, listings)
  - The DB2 Bind statements, transfer (FTP) and DB2 Job execution
  - The activation of the CICS load-modules

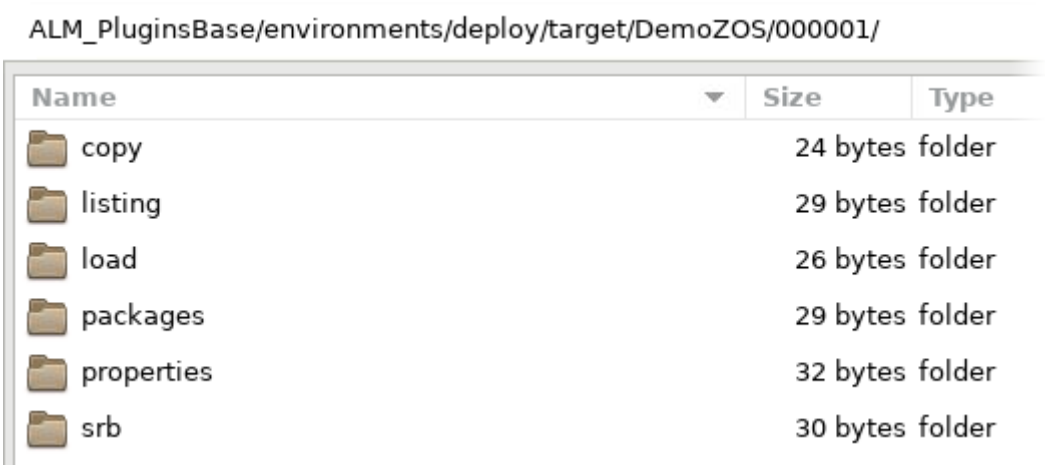
During this Deploy Job, the pipeline restores the Package files from the archive and copies them via FTP to their respective PDS(s) and special jobs will be created and executed on z/OS. The Load-modules are received through a transmitted sequential file to the PDS.

The FTP and Job results are analyzed to validate the executed deployment actions. In case of errors, messages are transmitted to the CloudBees log and the deployment is stopped.

By clicking the Build button, the Deploy Project Job will be created and the process will start.



Deploy Package Files execution with z/OS pipeline Steps



Package Files with files generated by the z/OS pipeline Steps of promotion in deploy folders

The following screen shows the JCL for receiving this transmitted sequential file.

```
SDSF EDIT    IBMUSERA (JOB00070) JCLEDIT                    Columns 00001 0007
Command ===> _                                               Scroll ===> CSR
****** ***************************** Top of Data ****************************
000001 //IBMUSERA JOB ,'IKAN JCLRECV',
000002 //         MSGLEVEL=(1,1),MSGCLASS=X,
000003 //         CLASS=A,REGION=0M,USER=IBMUSER
000004 //*         NOTIFY=IBMUSER,TYPRUN=SCAN
000005 //*XEQ ROUTEID=ADCD
000006 //*********************************************************************
000007 //**   RECEIVE PROGRAM                                               **
000008 //*********************************************************************
000009 //RECVLOAD  EXEC PGM=IKJEFT01,COND=(4,LT)
000010 //SYSPRINT DD SYSOUT=*
000011 //SYSTSPRT DD SYSOUT=*
000012 //* RECEIVE INDS('recvfile')
000013 //* DA'loadlib') SHR
000014 //SYSTSIN  DD *
000015  RECEIVE INDSNAME('IKANALM.DEMOS.P0000001.LOAD.ACCEPT2')
000016  DA('IKANALM.DEMOS.TEST.LOADLIB') SHR
000017 /*
000018 //*-------------------------------------------------------------------
000019 //*---         DELETE RECEIVED FILES
```

Receive program

The following screen shows the load modules after the FTP from the CloudBees archive to the mainframe in a PDS.

```
BROWSE    IKANALM.DEMOS.TEST.LOADLIB(ACCEPT2)       Line 00000000 Col 001 080
Command ===> _                                      Scroll ===> CSR
******************************** Top of Data ********************************
.^.....OACCEPT2 .......*IGZCBSO ...^...vCEESTART...}...qCEEBETBL...y....CEELOCT
.^.....OCEEFMAIN......  CEEROOTA......  CEEROOTD......  IGZEOPT ......  IGZETUN
.^.....}CEESG003......  CEESG004......  CEESG006......  CEESG007......  CEESG008
^3 ....................................................................................
^..5695PMB01 ......&□→
^@.^..5655G5300 .......^..569623400 .....PL/X-390  ..................^..569623400
^ h......ACCEPT2/IBMUSER/ALM000001.....RSI70816474.....RSI70816342
.......... ..\...-.................a...v...a...}...Y...ø...y...-....
 00..CEE...^.... 00.qól....x..........□□0}. .0.q□0<.............. ...≥.......0
.......Ö.......................... ... ... ...<... ...F...□...0.................
.......Y.............≤..v....≤..@.......H...U.......@.......È...H...d...{.......
.....................≤..÷....≤..T....≤..{....≤..D....≤..H
```

PDS with load modules

At this moment our programs are available for testing in the z/OS Test environment. The deployment to another z/OS Environment, be it another LPAR or Production environment, is a similar process.

CloudBees –Administrator's Point of View

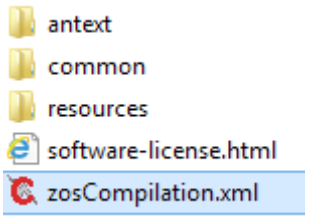
Step 1

The IKAN Pipelines for z/OS scripts

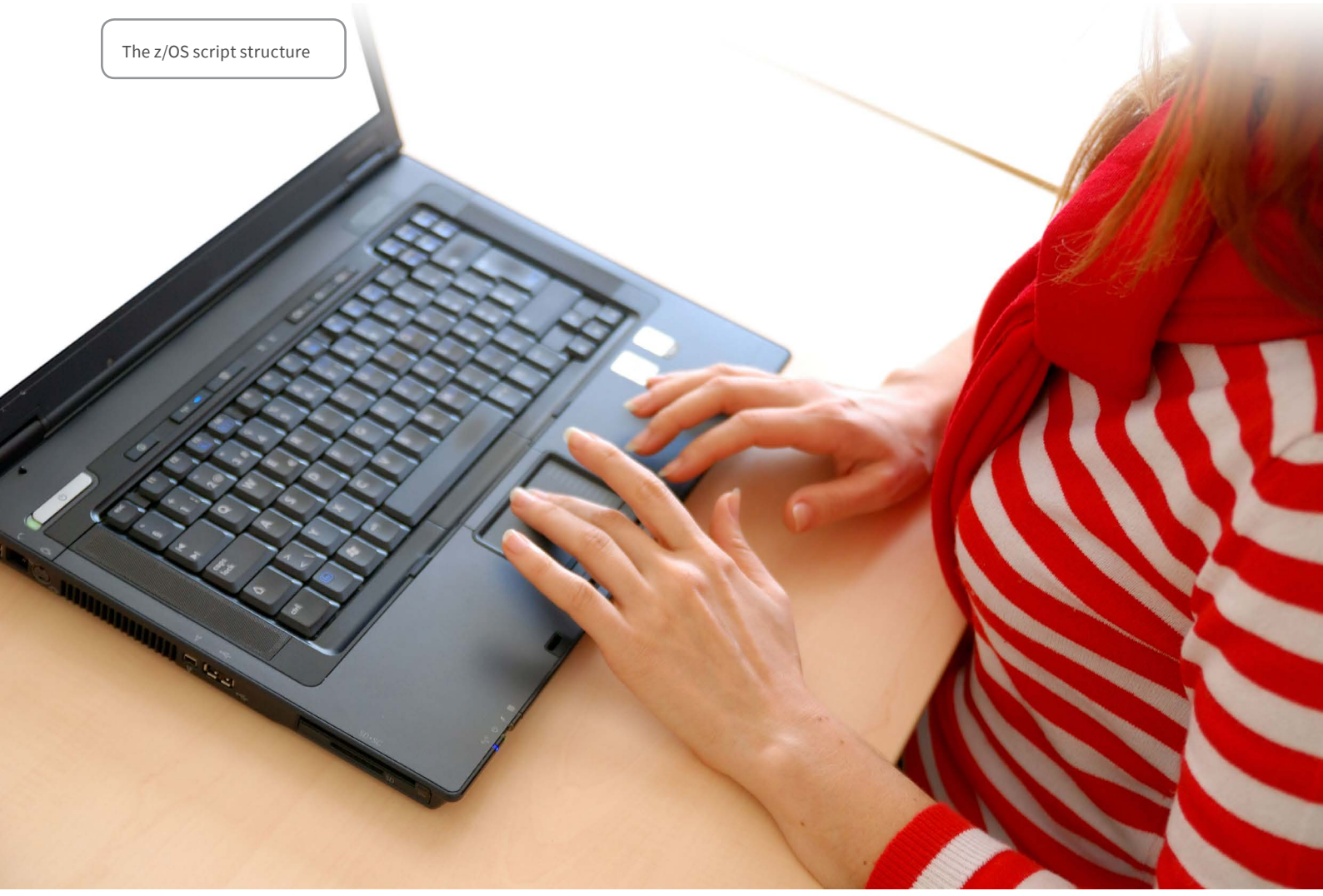
The concept

To compile or deploy programs one or more scripts are executed via CloudBees' pipelines. These scripts (steps) are defined in the IKAN Plugin for z/OS plugin, itself included with other files to the "LCM4ZOS_phases-jenkins-[version].zip" archive.

See the corresponding documentation for installing these files from the archive. In this section, we will first explain the different components of the scripts and, next, we will show how a script is represented and used in CloudBees.

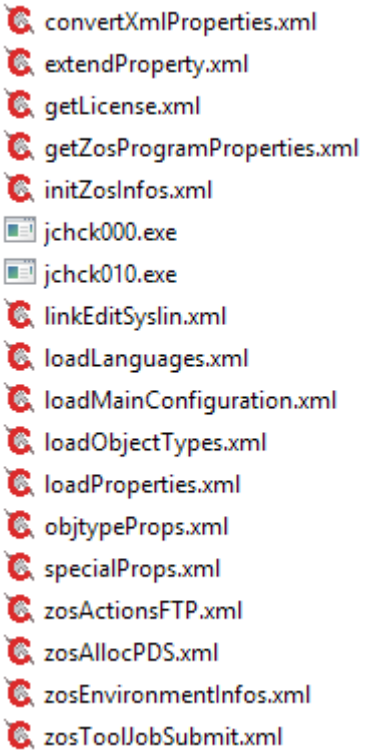


The z/OS script structure



The Common Files

Common files are files available to be used by all scripts.



List of common files

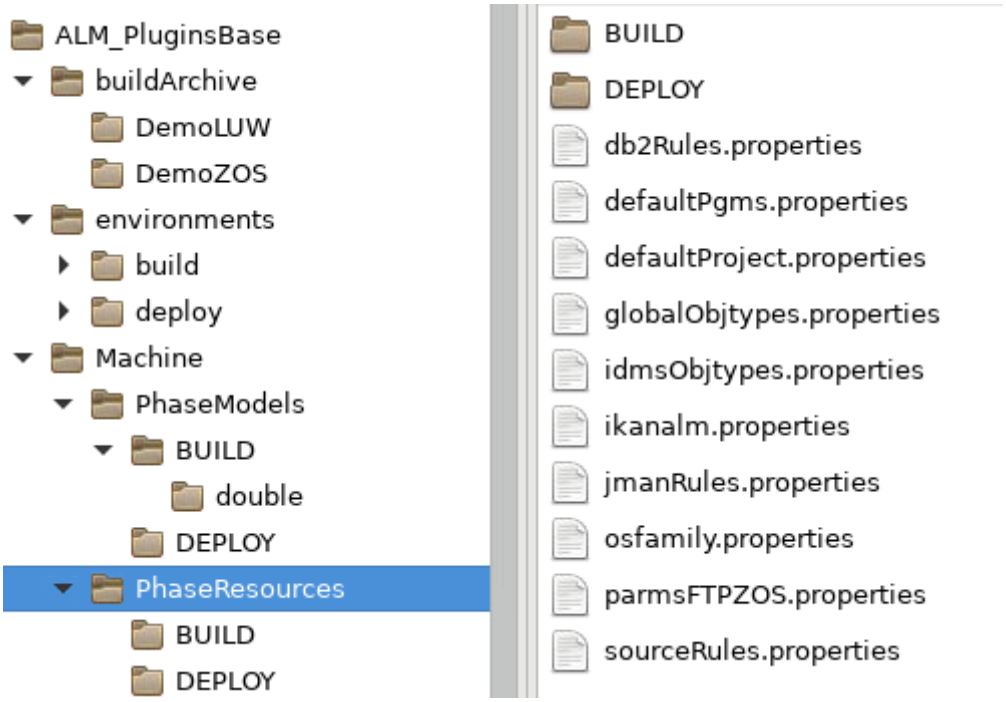
As an example, we have here a common script file that is used for linking a COBOL program. This common file will be used as a template to finally generate the correct and complete JCL step for linking a program.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Standard options for linkedit SYSLIN parameters -->
<project basedir="." default="linkSyslin" name="linkEditSyslin">
        <!-- ================================================================== -->
        <!-- Completing SYSLIN cards depending on object type and PGM options-->
        <!-- ================================================================== -->
  - <target name="linkSyslin">
        <echo file="${syslinFile}">// DD *</echo>
      - <if>
            <isset property="include.syslib1"/>
          - <then>
                <echo file="${syslinFile}" append="true"> ${include.syslib1}</echo>
            </then>
        </if>
      - <if>
            <isset property="include.syslib2"/>
          - <then>
                <echo file="${syslinFile}" append="true"> ${include.syslib2}</echo>
            </then>
        </if>
      - <if>
            <isset property="include.syslib3"/>
          - <then>
                <echo file="${syslinFile}" append="true"> ${include.syslib3}</echo>
            </then>
        </if>
        <echo file="${syslinFile}" append="true"> /* // DD DSN=&&OBJECT,DISP=(OLD,DELETE)</echo>
      - <if>
            <isset property="pgm.datacom"/>
          - <then>
                <echo file="${syslinFile}" append="true"> //* DD DSN=DATACOM.DEFAULTS(MEMBER),DISP=SHR </echo>
            </then>
        </if>
        <echo file="${syslinFile}" append="true"> // DD DSN=&&LCTFILE(${member}),DISP=(OLD,DELETE) </echo>
      - <if>
            <istrue value="${includeName}"/>
```

Link file template

The Resource Files

Resource files are used to define specific, reusable properties. Models and resource files are available for Build or Deploy steps or the can be used by both Build and deploy.



List of resource files or shareable properties

List of BUILD resource files

As an example, the COBOL language definitions from the languagesZOS.properties file:

```
# Language ==> COBOL
COBOL.lang.ref=cobol
COBOL.copy=step:copySource      |numseq:10    |model:copySource_jcl     |program:IEBGENER  |parms:
COBOL.expandSource=step:expand |numseq:15    |model:expandSource_jcl   |program:IKJEFT01  |parms:TRACE
COBOL.db2=step:preCompileDb    |numseq:20    |model:precompileDb2_jcl  |program:DSNHPC    |parms:HOST(
COBOL.datacom=step:preCompileDb|numseq:12    |model:precompileDatacomCobol_jcl,precompileDatacomExport
#COBOL.idms=step:preCompileDb  |numseq:20    |model:precompileIdms_jcl|program:IDMSC     |parms:(COBOL
#COBOL.ims=step:preCompileDb   |numseq:20    |model:precompileIms_jcl  |program:DSNIPC    |parms:(COBOL
# service parameters are completed with (OBJTYPE).useroptions
COBOL.service=step:preCompile  |numseq:05    |model:precompileService_jcl|program:SRVGEN |parms:SERVICE
#COBOL.endevor=step:preCompile |numseq:10    |model:addEndevor_jcl     |program:NDVRC1    |parms:C1BM3D
COBOL.cics=step:preCompileCics |numseq:30,60 |model:precompileCics_jcl|program:DFHECP1$  |parms:COBOL
COBOL.compilation=step:compile |numseq:35,65 |model:compileCobol_jcl   |program:IGYCRCTL  |parms:LIST,
#COBOL.compilation=step:compile|numseq:35    |model:compileEndevor_jcl|program:NDVRC1    |parms:C1BM3D
#COBOL.db2=step:preCompileDb    |numseq:36    |model:compileEndevorDb2_jcl |program:      |parms:   (ESQ
#COBOL.abendaid=step:debugger   |numseq:40,70 |model:debugAbendaid_jcl |program:ABENDAID  |parms:(COBOL
#COBOL.smarttest=step:debugger |numseq:40,70 |model:debugSmarttest_jcl|program:SMARTEST  |parms:
COBOL.xpediter=step:debugger    |numseq:40,70 |model:debugXpediterCompile_jcl,debugXpediterExport_jcl
COBOL.linktable=step:copyLct    |numseq:45    |model:copyLct_jcl        |program:IEBGENER  |parms:
COBOL.linkedit=step:linkEdit    |numseq:50,80 |model:linkEdit_jcl,xmitload_jcl|program:HEWL|parms:LIST,
#COBOL.linkedit=step:linkEdit   |numseq:50,80 |model:compileEndevorEnd_jcl                |program:
COBOL.listing=step:listing      |numseq:90,95 |model:listing_jcl        |program:IEBGENER  |parms:
COBOL.endjcl=step:endJcl        |numseq:99    |model:compileEnd_jcl     |program:|parms:   |rcmax:4
```

COBOL parameters list

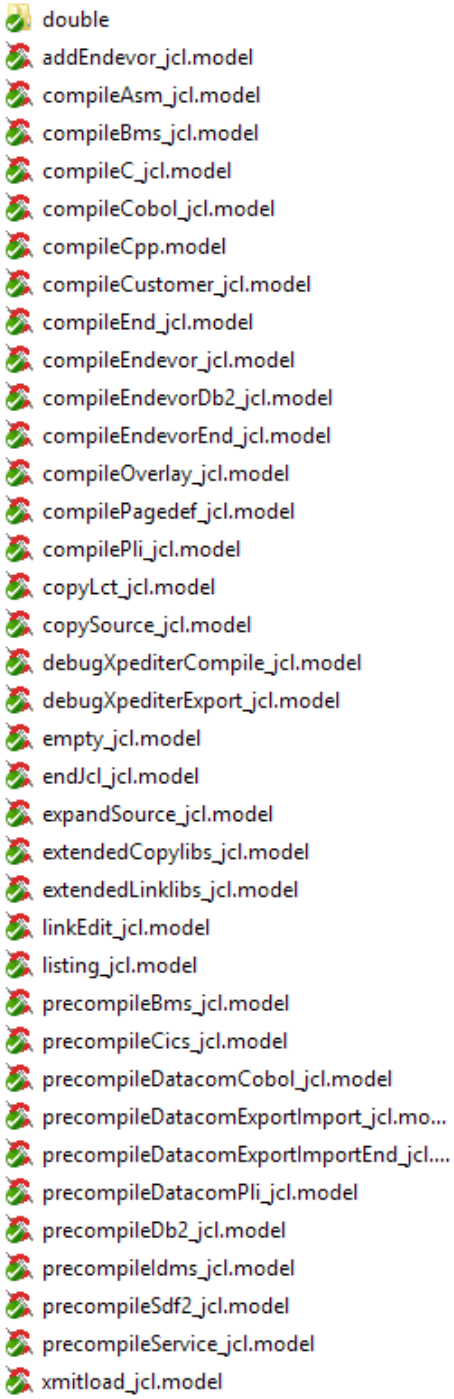
The example below shows the generated properties to use for a COBOL program.

```
pgm.amode=31
pgm.cics=false
pgm.compilation=true
pgm.compile.double=false
pgm.compile.parms=DATA(31),RENT
pgm.compile.parms.batch=DATA(31),RENT
pgm.compile.parms.cics=DATA(31),RENT
pgm.compilerType=IBM
pgm.debugger=false
pgm.execution=true
pgm.filename=ACCEPT2
pgm.joblang=JCL
pgm.jobtype=JOB
pgm.language=COBOL
pgm.lctname=ACCEPT2
pgm.link.parms=
pgm.link.parms.batch=
pgm.link.parms.cics=
pgm.linkedit=true
pgm.loadname=ACCEPT2
pgm.name=ACCEPT2
pgm.noname=true
pgm.objtype=COBB
pgm.os=ZOS
pgm.project=DemoZOS
pgm.reus=RENT
pgm.rmode=ANY
pgm.suffix=cblbatch
pgm.type=program
```

Generated parameters

The Model files

Model files are used as templates for JCL cards. As an example, we added a model for a JCL to compile a COBOL program, with a IBM compiler.



List of models for JCL steps

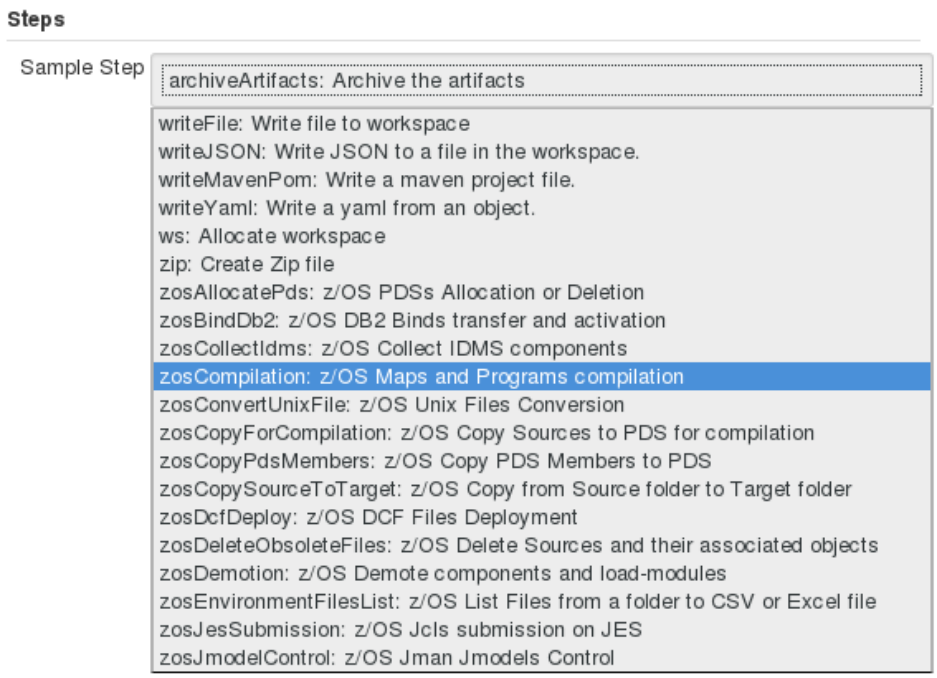
The compileCobol_jcl.model

```
//** ./models/compileCobol_jcl : START
//*****************************************************************
// SET PARMCOB='${compilation.lang.parms}'
// SET PARMCOB0='${pgm.compile.parms.batch}'
//*****************************************************************
//**     COMPILE THE ELEMENT ${ref.acronym}.${member}
//*****************************************************************
//${pgm.language}  EXEC PGM=${compilation.lang.program},COND=(${compilation.lang.rcmax},LT),
//    PARM=(&PARMCOB,
//    &PARMCOB0),MAXRC=${compilation.lang.rcmax}
//STEPLIB  DD  DISP=SHR,
//             DSN=${compilation.lang.prefix}.${compilation.lang.lib}
//SYSIN    DD  DISP=(OLD,PASS),DSN=&&&SRCOMPIL
//SYSLIN   DD  DISP=(,PASS),DSN=&&OBJECT,
//             UNIT=SYSDA,SPACE=(CYL,(2,2)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSPRINT DD  DISP=(MOD,PASS),DSN=&&COMPLIST,
//            UNIT=SYSDA,SPACE=(TRK,(10,90),RLSE)
//*            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0)
//*
${copylibs}
${extendedCopylibs}
//** ./models/compileCobol_jcl : END
```

JCL model for COBOL compile step

An IKAN z/OS phase and its usage: the z/OS Compilation phase

Pipelines are created with the help of the “Pipeline Syntax” of CloudBees. When you installed the Plugin for z/OS, scripts are available to be used in a pipeline. First, select a z/OS scripts:



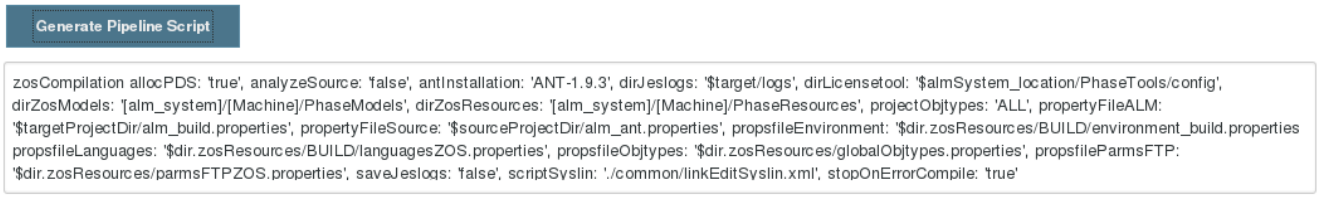
Selecting the zosCompilation script

Then, CloudBees edits the parameters to complete with eventually a default value:

Alloc PDS	true
Analyze Source	false
Ant Installation	ANT-1.9.3 ⚠ Choose Ant Installation
Dir Jeslogs	$target/logs
License Tool Dir	$almSystem_location/PhaseTools/config
Dir Zos Models	[alm_system]/[Machine]/PhaseModels
Dir Zos Resources	[alm_system]/[Machine]/PhaseResources
Project Objtypes	ALL
Main Property File	$targetProjectDir/alm_build.properties
Property File Source	$sourceProjectDir/alm_ant.properties
Propsfile Environment	$dir.zosResources/BUILD/environment_build.properties
Propsfile Languages	$dir.zosResources/BUILD/languagesZOS.properties
Propsfile Objtypes	$dir.zosResources/globalObjtypes.properties
Propsfile Parms FTP	$dir.zosResources/parmsFTPZOS.properties
Save Jeslogs	false
Script Syslin	./common/linkEditSyslin.xml
Stop On Error Compile	true

The zosCompilation script parameters

Finally, you can generate the Pipeline Script to be integrated.



Generated zosCompilation phase as a step

Complete the step with the Ant environment. Use the “almPipeline_zosBuild.groovy” pipeline as sample for completing the step properly.

The objective of the ‘zosCompilation’ Phase is to compile z/OS programs with, mainly, Assembler, COBOL and PL/1, BMS map languages, and working with CICS and Databases as DB2, Datacom, IDMS or IMS. The Phase will also control the results of the JCL submit and it will collect all files generated by the compile Jobs. Also, when applicable, the DB2 Bind Files will be generated.

This phase assumes that the files for compiling sources and the source program files have already been transferred to the mainframe in the correct PDSs. Normally, this would be done by a previous dedicated Phase.

It is the task of the Administrator to make sure that the default values for the resource file parameters are set to the company standards. He can easily do that by changing the parameter values in the CloudBees web interface.

```
echo "zosCompilation()"
withEnv(["ANT_ARGS=-q -propertyfile ${propertyFileALM}"]) {
    withAnt(jdk: "${jdkInstallation}") {
        zosCompilation (
            allocPDS: 'true',
            analyzeSource: 'false',
            antInstallation: "${antInstallation}",
            dirJeslogs: '$targetProjectDir/logs',
            dirLicensetool: '$almSystem_location/PhaseTools/config',
            dirZosModels: '$almSystem_location/Machine/PhaseModels/BUILD',
            dirZosResources: '$almSystem_location/Machine/PhaseResources',
            projectObjtypes: 'ALL',
            propertyFileALM: "${propertyFileALM}",
            propertyFileSource: '$sourceProjectDir/$project_name.properties',
            propsfileEnvironment: '$almSystem_location/Machine/PhaseResources/BUILD/environment_build.properties',
            propsfileLanguages: '$almSystem_location/Machine/PhaseResources/BUILD/languagesZOS.properties',
            propsfileObjtypes: '$almSystem_location/Machine/PhaseResources/globalObjtypes.properties',
            propsfileParmsFTP: '$almSystem_location/Machine/PhaseResources/parmsFTPZOS.properties',
            saveJeslogs: 'false',
            scriptSyslin: '$almSystem_location/PhaseScripts/linkEditSyslin.xml',
            stopOnErrorCompile: 'true'
        )
    }
}
echo "End zosCompilation()"
```

Full zosCompilation step

The execution of the 'zosCompilation' Phase will use the 'zosCompilation' script to finally generate a complete JCL, taking into account all JCL steps to be executed.

The figure below shows the generated JOB card and the STEP card to compile the COBOL program. The complete generated JCL can be found in Appendix V: Sample of z/OS compilation JCL.

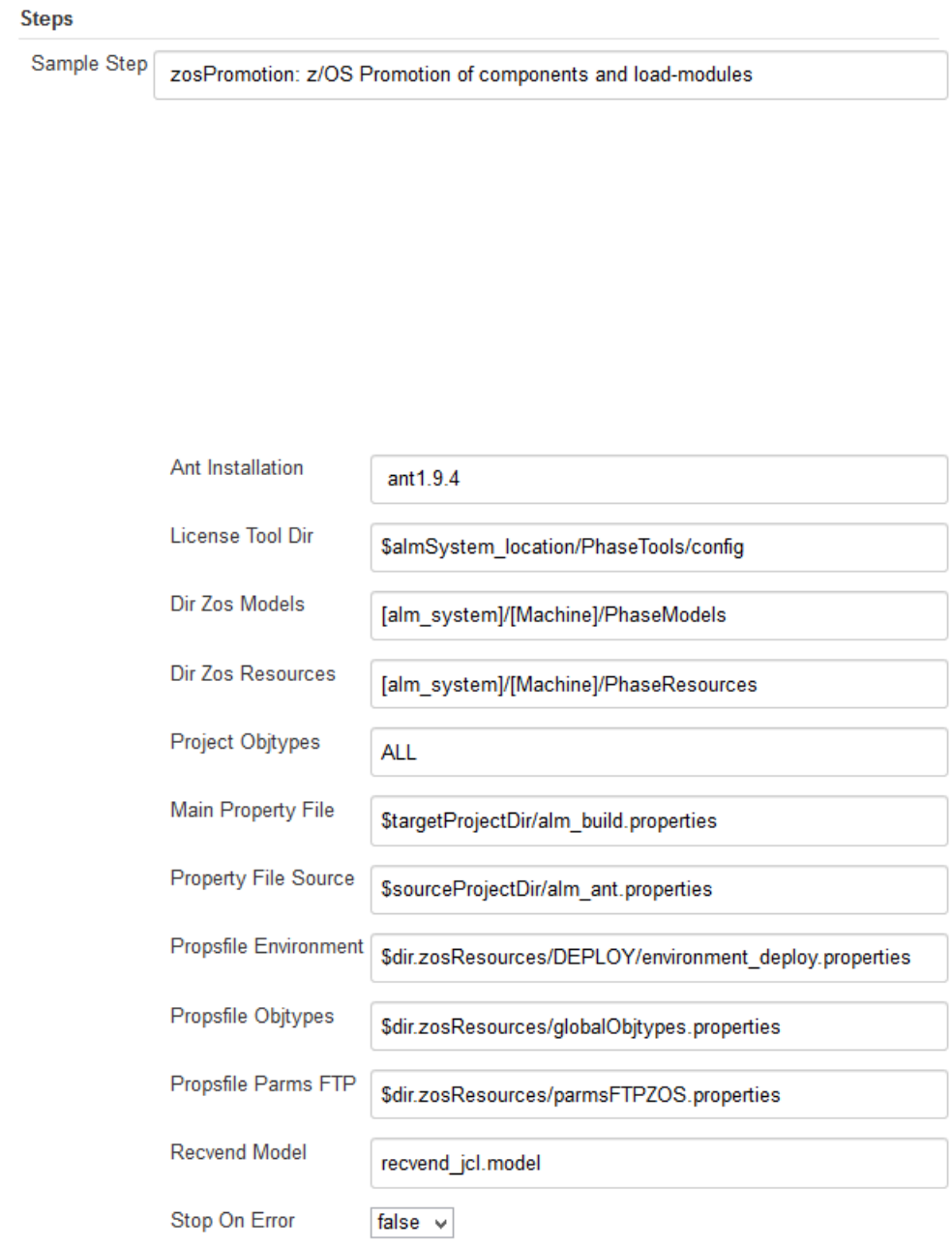
```
...
//*****************************************************************
//**    COMPILE COBOL2, store object in objlib if compile=ok
//**    compile listing is stored in IKAN ALM.DEMOS.TEST.LSTALIB
//*****************************************************************
//         SET PARMCOB='LIST,LIB,NOSEQ,NOCMPR2,MAP'
//         SET PARMCOB0='DATA(31)'
//*****************************************************************
//**    COMPILE THE ELEMENT                                      **
//*****************************************************************
//COBOL2  EXEC PGM=IGYCRCTL,COND=(4,LT),
//   PARM='&PARMCOB0,&PARMCOB'
//STEPLIB  DD  DISP=SHR,DSN=SYS1.COB2COMP
//SYSIN    DD  DISP=(OLD,PASS),DSN=&&&SRCOMPIL       (ULC010)
//SYSLIN   DD  DISP=(,PASS),DSN=&&OBJECT,
//             UNIT=SYSDA,SPACE=(CYL,(2,2)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=0)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSPRINT DD  DISP=(,PASS),DSN=&&COMPLIST,
//            UNIT=SYSDA,SPACE=(TRK,(10,10),RLSE)
//*            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0)
//*
//SYSLIB   DD  DISP=SHR,DSN=IKAN ALM.DEMOS.TEST.COPYLIB
//         DD  DISP=SHR,DSN=IKAN ALM.DEMOS.INTG.COPYLIB
//         DD  DISP=SHR,DSN=IKAN ALM.DEMOS.QUAL.COPYLIB
//         DD  DISP=SHR,DSN=IKAN ALM.DEMOS.PROD.COPYLIB
//*
```

Generated JCL step of compilation

The Administrator will do this for all the scripts required for the mainframe Build (compile) and deploy processes (probably for several z/OS projects).

An IKAN z/OS script and its usage: the z/OS Deployment script

Next, we will show you an example of a Deployment script. The main step of the z/OS deployment with CloudBees is the ‘zosPromotion of components and load-modules’ script.



zosPromotion parameters

Generated zosPromotion script as a step

Complete the step with the Ant environment.

```
echo "Start Promotion"
withEnv(["ANT_ARGS=-q -propertyfile ${propertyFileALM}"]) {
    withAnt(jdk: "${jdkInstallation}") {
        zosPromotion (
            antInstallation: "${antInstallation}",
            dirLicensetool: '$almSystem_location/PhaseTools/config',
            dirZosModels: '$almSystem_location/Machine/PhaseModels/DEPLOY',
            dirZosResources: '$almSystem_location/Machine/PhaseResources',
            projectObjtypes: 'ALL',
            propertyFileALM: "${propertyFileALM}",
            propertyFileSource: '$sourceProjectDir/$project_name.properties',
            propsfileEnvironment: '$almSystem_location/Machine/PhaseResources/DEPLOY/environment_${LEVEL}.properties',
            propsfileObjtypes: '$almSystem_location/Machine/PhaseResources/globalObjtypes.properties',
            propsfileParmsFTP: '$almSystem_location/Machine/PhaseResources/parmsFTPZOS.properties',
            recvendModel: 'recvend_jcl.model',
            stopOnError: 'false'
        )
    }
}
```

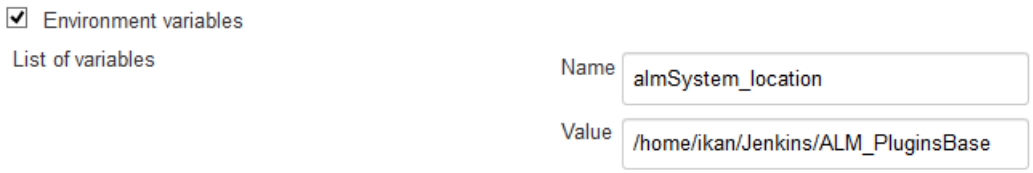
Full zosPromotion step

Typically, the parameters are very similar, but you can specify another property file for the target environment during the project setup if required.

Step 2

Create the CloudBees “Checkout” project

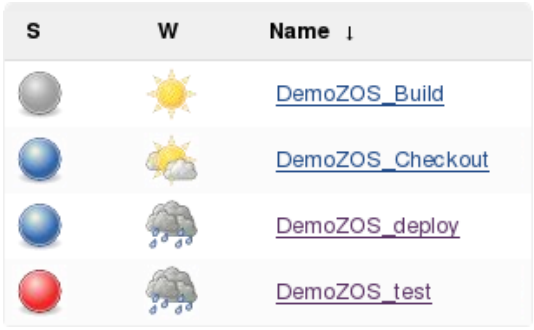
First define the Configuration settings for a global variable.



Global variable

Once the scripts have been installed in the CloudBees Plugins, the Administrator creates complete life cycles for Mainframe projects

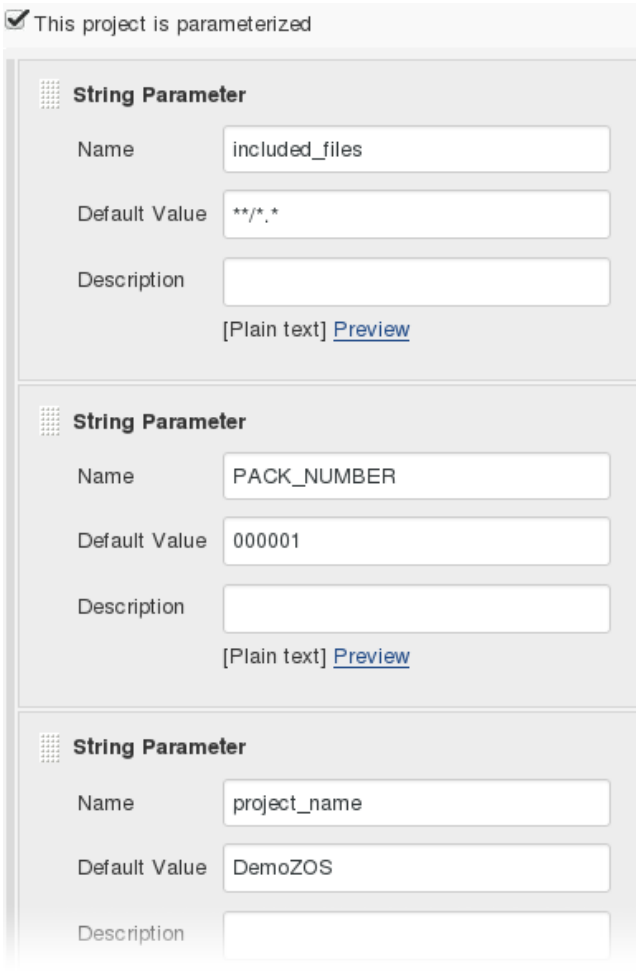
In this example we will show the creation of a package-based Mainframe project as 3 or 4 (sub)projects: [project]_ Checkout, [project]_Build, [project]_Deploy, [project]_Test, etc..



Project's life-cycle

Define the Project_Checkout settings as the following.

Create a 'Pipeline' project and use the 'Pipeline script from SCM' option. Add parameters: included_files, PACK_NUMBER, project_name as:



Create Checkout project parameter

included_files variable is useful for selecting files you want to build or deploy in case you don't use the packageFile parameter (PACK_NUMBER).

It can also be used in combination with the packageFile parameter. See the packageFile Usage paragraph, below.

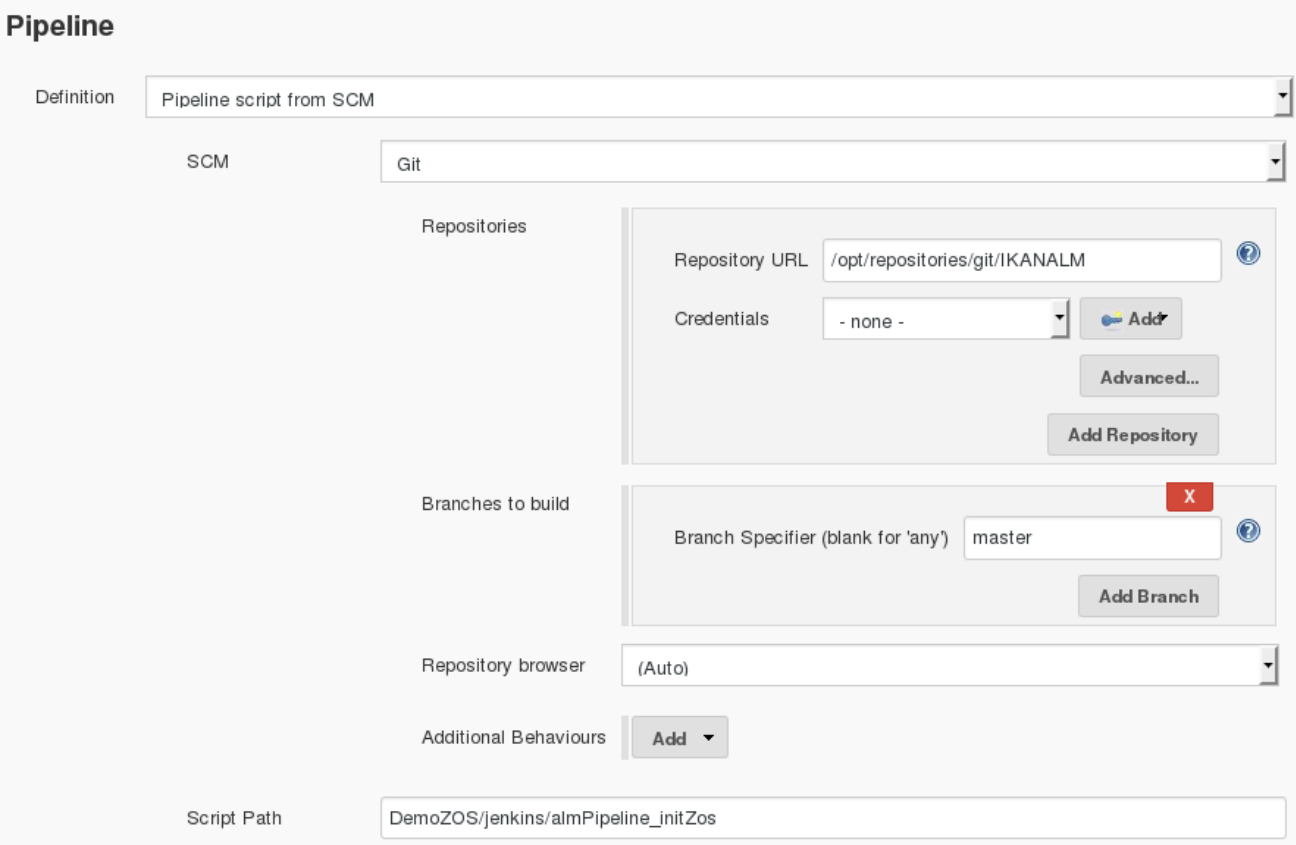
PACK_NUMBER variable is for identifying the Package you want to use for doing the build or deploy operations.

project_name variable is used to define the project name to use all over the lifecycle. It can be defined from the almPipeline_zosInit.groovy (or similar) script in the VCR project.

This file has to be installed in your VCR project at this path (i.e. [project/branch]/Jenkins/almPipeline_zosInit).

Install the 'Pipeline script from SCM' with your data.

And put the groovy script relative path and name into the VCR project as the almPipeline_initZos.groovy file.



Link VCR project and "zosInit" pipeline location

Review variables included by default for your project. Don't delete unidentified variables. This pipeline assumes to select the useful files for the designated package.

By this selection (using included_files value), the rest of the project components will not be used by pipelines in next (sub)projects. **After save, you can test your pipeline!**

Step 3

Create the CloudBees “Build” project

Create a ‘Pipeline’ project. Add same parameters as the ‘Check-Out’ project.

Install the ‘Pipeline’ script with the contents of almPipeline_zosBuild.groovy.

The following screen shows a Build/Compile Project pipeline.

Pipeline script

Script

```
27  stages {
28      stage('Build') {
29          steps {
30              script {
31                  propertyFileBuild = "${sourceProjectDir}/alm_initial_ant.properties"
32                  propertyFileALM = "${targetProjectDir}/alm_build_ant.properties"
33                  packageFile = "${sourceProjectDir}/packages/pack-${PACK_NUMBER}.txt"
34                  if (isUnix()) {
35                 sh "rm -rf ${targetProjectDir}"
36                  } else {
37                 bat "start cmd.exe /C rmdir /S /Q ${targetProjectDir}"
38                  }
39              }
40              dir("${targetProjectDir}") { writeFile file:'alm_build_ant.properties', text:'' }
41              echo "Start propertyBuild()"
42              withEnv(["ANT_ARGS=-q -propertyfile ${propertyFileBuild}"]) {
43
```

Build pipeline

Step 4

Modify the phase parameters

If necessary review the packageFile path and name.

Review the archiveFile and Artifactory definitions.

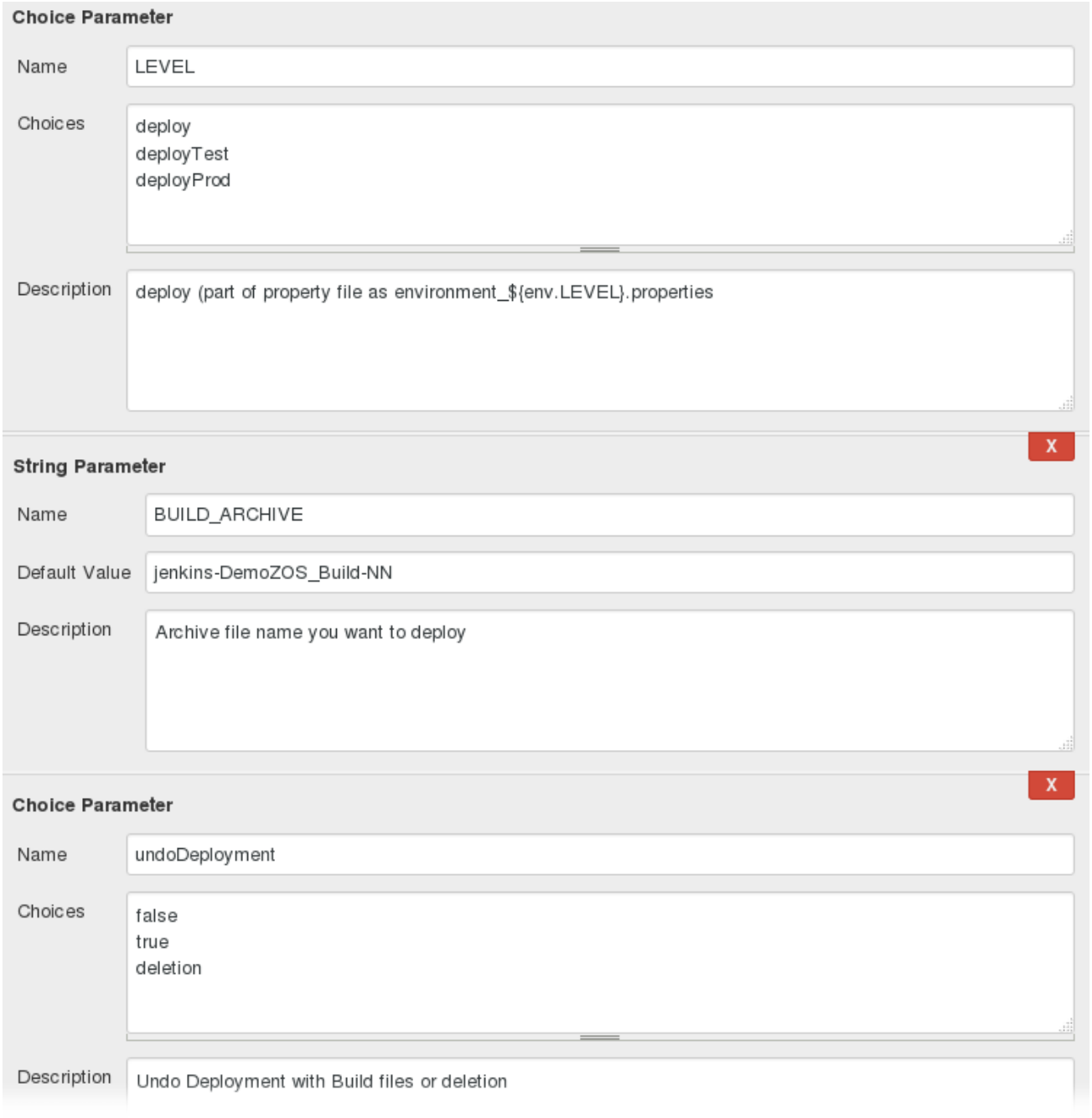
If needed, the default values of these parameters can be modified as shown on the Phase Parameters above. A**fter the save, you can test your pipeline!**

Step 5

Create the CloudBees “Deploy” project

Create a ‘Pipeline’ project.

Add the same parameters as the ‘Check-Out’ project and add ‘LEVEL’ and ‘BUILD_ARCHIVE’ and ‘undoDeployment’ parameters as:



Deploy pipeline more parameters

The LEVEL parameter helps to select the 'environment_X.properties files to use for deploying. See the PhaseResources/DEPLOY folder with its files.

The BUILD_ARCHIVE parameter is similar as the archiveName name except that the PACK_NUMBER is ignored here. So, NN will be the number coming from the Build you want to deploy.

The undoDeployment parameter helps to revert to the previous version of files defined in the package Build.

- The revert action (=true) is done if you install the zosCopyPdsMembers Phase/Step.
- The deletion (=deletion) is done if you install the zosDeleteObsoleteFiles Phase/Step.

Install the 'Pipeline' script with the contents of almPipeline_zosDeploy.groovy.

Pipeline script

Script

```
121         }
122         echo "Start Promotion"
123             withEnv(["ANT_ARGS=-q -propertyfile ${propertyFileALM}"]) {
124             withAnt(jdk: "${jdkInstallation}") {
125                     zosPromotion (
126                         antInstallation: "${antInstallation}",
127                         dirLicensetool: '$almSystem_location/PhaseTools/config',
128                         dirZosModels: '$almSystem_location/Machine/PhaseModels/DEPLOY',
129                         dirZosResources: '$almSystem_location/Machine/PhaseResources',
130                         projectObjtypes: 'ALL',
131                         propertyFileALM: "${propertyFileALM}",
132                         propertyFileSource: '$sourceProjectDir/$project_name.properties
133                         propsfileEnvironment: '$almSystem_location/Machine/PhaseResourc
134                         propsfileObjtypes: '$almSystem_location/Machine/PhaseResources/
135                         propsfileParmsFTP: '$almSystem_location/Machine/PhaseResources/
136                         recvendModel: 'recvend_jcl.model',
137                         stopOnError: 'false'
```

Deploy pipeline

If necessary review the contents. Review the archiveFile and Artifactory definitions.

Using the same method as described above, the Deploy Environment is completed with the required z/OS deployment scripts such as: 'zosPromotion', 'zosBindDb2', 'zosUpdateCics'. **After the save, you can test your pipeline!**

Now that the CloudBees Administrator has done his job, the User can start using CloudBees for building/compiling, promoting or deploying his programs.

Conclusion

CloudBees offers an alternative for pure mainframe-based development by combining an Eclipse-based development environment with a distributed version control repository. On top of that CloudBees complements the development process with continuous integration and continuous deployment (CI/CD) features.

The major asset of the IKAN Plugin for z/OS is the scripts concept. JCL can be very complicated. And difficult to write and maintain. By using z/OS scripts, you can generate and tailor any JCL step. And ypour JCL will follow your company standard and name giving.

Thanks to the script concept and the available models and resources we can also guarantee an easy and successful implementation (as an average, it will only take a few weeks). The key element is for you to define your process. Once that has been established, the implementation of IKAN Plugin for z/OS in CloudBees is fast and straightforward.

If you are already using a mainframe solution like CA-Endevor or Serena ChangeMan and you would decide to migrate to CloudBees, you will of course need to migrate your CA-Endevor or Serena ChangeMan legacy to a VCR. To do so, we have a standard migration procedure.

In a nutshell: by implementing CloudBees for z/OS, you can continue exploiting the full strengths of your mainframe and seamlessly combine them with new innovative tooling. This will help you cutting down the costs of maintaining different systems, and above all ease the work of your developers as CloudBees and its IKAN Plugin for z/OS will take care of the different steps in the lifecycle of your application including its deployment on the mainframe.

For More Information

To know more, visit http://www.ikanalm.com
Contact IKAN Development: info@ikanalm.com

Related Document

Modern Mainframe Development and Jenkins

The following appendices explain the terminology used by the different ALM mainframe software providers.

Appendix I: IKAN Pipelines for z/OS scripts Terminology

The following table explains the terms used by IKAN Pipelines for z/OS scripts and provides a brief comment for each of them. This will help users of CloudBees to have a better understanding of the terminology used.

IKAN Pipeline	Remarks
VCR	A Version Control Repository contains the components to manage. Examples of VCRs are: Git, Subversion, IBM ClearCase, Serena PVCS, Microsoft VisualSourceSafe.
Project	A tailored Lifecycle process including development, testing, quality assurance and production can be easily defined, implemented and enforced, offering a comprehensive framework across all major platforms including Windows, UNIX, Linux and IBM mainframe systems. IKAN Pipelines for z/OS scripts also support a stream-based project model allowing project managers to easily add Lifecycles to each version of a project, which makes it easy to differentiate between maintenance, “urgency fix” or release build and deploy processes.
Lifecycle	Defines the Lifecycle(s) from Development to Production Levels for Streams.
Project Stream	Each Project contains exactly one HEAD Project Stream and may contain one or more Branches. A Project Stream is a working entity within IKAN Pipelines for z/OS scripts
Level	CloudBees Project(s) define every step of the Lifecycle from Development to Production, supporting physical Environments in or out of workspace.
Environment	IKAN Pipelines for z/OS scripts use the (logical) Level concept in which (Build/Deploy) environments can be defined. Every environment represents a Machine (Server/OS) on the network where the Source and Target Locations are defined for executing scripts. This is a unique architectural IKAN Pipeline feature, representing the true multi-platform aspect of scripts.
Package	Instead of using a Release Project, IKAN Pipelines for z/OS scripts may work with Packages. The required files must be added manually from the VCR info to the Package file list. It knows the relative location of the files in the VCR project.
Build	The Build Project Request in CloudBees will usually take care of a compile procedure for components or a deployment or tests.
Script	Users can extensively customize the workflow of their projects, by using highly reusable building blocks, called scripts. By using the “Generate Pipeline syntax” feature, Scripts can be shared between different Projects, also between different CloudBees installations.
Script (Ant)	Runs the process (e.g., for build, compilation, deployment, copy, etc.) using the Source and Target locations on the CloudBees machine. A script can use property files, models and other scripts generally defined in a Phase.

Build Number	CloudBees generates a unique build number that can be used in several processes to identify the output from the (build/compile) procedures. The 'zosCompilation' Phase is also able to put this information on members in a Partitioned Dataset on z/OS.
Approval	May be added in the Pipeline with the 'input' command, manually.
Rollback	CloudBees has no rollback process. IKAN scripts assume this functionality with the 'undo-Deployment' option.
Machine	A machine runs an CloudBees Agent which will take care of building/deploying the software components. Linux/UNIX (flavors) and Windows platforms. The Agent (LUW) machine can update more than one LPAR via FTP connections. The z/OS scripts might be reused with models and PDS definitions using several FTP connections.
Release Number/ Incident Number	CloudBees may have an Issue Tracking System plugin that allows you to easily link existing issue or defect tracking systems.
Archive	CloudBees has a plugin for compressing and IKAN Pipelines for z/OS scripts save all Build results in Archive and keeps them in a dedicated location. Archives are identified with the Build Tag. Also, an Artifactory management is available in the IKAN Pipelines.
Report	CloudBees provides activity reports.
Scripts adds	Remarks
Extension/ Object-type	The extension/objtype determines the processing needed for a certain file type. This is defined by a property file and scripts. Object-types are used for z/OS activities.
Obsolete File	CloudBees has no process for scratching individual files. This action is resolved with a script which scratches the source component using the ".to_be_deleted" suffix in the VCR. Associated z/OS components are deleted in their PDS and DB2 binds are re-executed. For deleting a component from the Production environment, the components should be renamed into the VCR with the special ".to_be_deleted" suffix. Next, the project should be built and deployed, and tested in all the Levels between Development and Production, ensuring that this deletion does not jeopardize the Production. This delete task will be activated with the suffix of the component (don't delete). Next IKAN Plugin may assume the deletion during the deployment by using the suffix in the dedicated Phase.
Baseline	IKAN Pipelines for z/OS scripts does not manage a baseline in VCR.

Appendix II: CA-ENDEVOR Terminology

The following table maps the terms used by IKAN Pipelines for z/OS scripts and CA-Endevor and provides a brief comment for each of them. This will help the respective users of CloudBees or CA-Endevor to have a better understanding of the terminology used.

IKAN Pipeline	CA Endevor	Remarks
VCR	Database/Delta	CA-Endevor assumes VCR versions with the Image and Delta file(s).
Project	System and/or Sub-system	Within IKAN Pipelines for z/OS Scripts, the defined project needs attributes to tell CloudBees to which CA-Endevor System/Sub-system the Software Items should be added in Environment parameters.
Lifecycle	Map	Defines the Lifecycle from Development to Production.
Stream	Not available	CA-Endevor works with a unique Project version.
Package	Package	CA-Endevor groups components in Batch packages.
Level	Stage	Defines every step of the Lifecycle from Development to Production.
Environment	(Stage)	This is a unique architectural IKAN Pipelines feature, not known in CA-Endevor, representing the true multi-platform aspect of Scripts.
Build Request	(Add) Action	The Build Project Request in CloudBees will usually take care of populating CA- Endevor with the Software Components (ADD action).
Script	Processor group	The processor group in CA-Endevor determines the ultimate process to run within a certain type. For example, the Processor Type COBOL might have processor groups for COBOL, DB2, CICS, BATCH, IMS etc.
Script (Ant)	Processor	Runs the process (e.g., for build or compilation).
Idrdata/Build number	Footprint	Build number: is an incremental number given after each software build. IDR DATA: Identification record data field. Identification records have a fixed format and fixed content, both defined by the program management binder. Is used by IBM Endevor footprints contain the following information: site ID, environment, system, subsystem, element, type, stage, version/level, and generate date/time.
Approval	Approval	CA-Endevor allows defining several Approval Groups which are in the same hierarchy. Every group may approve on any moment.
Rollback	Backout	CA-Endevor allows reversing the result from a promotion/delivery if it is a member(s) in a Partitioned Dataset (PDS). In the case of DB2 a (manual) rebind should be executed.

Machine	Ship	CA-Endevor only supports other z/OS Logical Partitions (LPARS).
Release Number/ Incident Number	CCID	The release/incident number within CloudBees may be passed to CA-Endevor as the CCID (Change Control Identifiers) most often correspond to mechanisms such as work order requests or request-for-service numbers.
Archive	Not available	CA-Endevor keeps these components in Stage Level with CCID's.
Report	Report	CA-Endevor allows using Batch reports.
Extension	Type	The extension/type determines the processing needed for a certain type.

Appendix III: Serena ChangeMan ZMF terminology

The following table maps the terms used by IKAN Pipelines for z/OS scripts and Serena ChangeMan ZMF and provides a brief comment for each of them. This will help the respective users of CloudBees or ChangeMan to have a better understanding of the terminology used.

IKAN Pipeline	ChangeMan ZMF	Remarks
VCR	Baseline/Delta/ Package	ChangeMan assumes VCR functionalities as Check-Out, Commit (Baseline Ripple), Check-In (Freeze) from the Package Lifecycle.
Project	Application	ChangeMan has the same concepts as IKAN Pipelines for z/OS Scripts but only for IBM mainframe systems. Also the stream-based project is not available.
Lifecycle	Stage/ Promotion Levels	Defines the Lifecycle from Development to Production.
Stream	Not available	ChangeMan works with a unique Project version.
Package	Package	ChangeMan uses the Package for the Development process up to the Stage action. IKAN Pipelines for z/OS Scripts leaves development actions to the customer IDE and the VCR. CloudBees runs with Build Requests and ChangeMan works with Staging, manage Build (Compile) requests, as well as Deployments with the Approval supervision for the Package. ChangeMan, however, needs to update the Baseline & Stacked Reverse Delta supports in double with the Production and the Package.
Level	Promotion Level	Defines every Level of the Lifecycle from Development to Production.

Environment	Site (Local or Remote)	The Local or Remote Site concept in ChangeMan is covered by the IKAN Pipelines for z/OS Scripts environment concept. A CloudBees Project (a logical Level) can have one or more environments.
Script	Procedures or skeletons	The skeletons in ChangeMan determine the process to run within a certain type. For example, the Procedure CMNCOB2 might have process skeletons for COBOL, DB2, CICS, IMS, etc.. Depending on the Source options.
Build Request	(ST) Action	ChangeMan takes care of the compile procedure the same way as CloudBees.
Script (Ant)	Skeleton Procedure	Runs the process (e.g., for build or compilation or deploy).
Build Number	Package Number	ChangeMan uses the Package number for versioning files.
Approval	Approval	ChangeMan allows defining several Approval Groups which are hierarchical. Every group may approve one after the other.
Rollback	Demotion	ChangeMan allows reversing the result from a promotion/delivery if it is a member(s) in a Partitioned Dataset (PDS). In the case of DB2 a (automatic) rebind will be executed.
Machine	Site	ChangeMan only supports other z/OS Logical Partitions (LPARS). The ChangeMan site is the Local or a Remote LPAR.
Release Number/ Incident Number	Not available	ChangeMan does not use Incident numbers. In the Package description panel, a reason may be entered for all included components.
Archive	Package	ChangeMan contains components in the Package which is frozen before the deployment. It designs the version to deploy.
Report	Report	ChangeMan allows using Batch reports.
Obsolete File	Scratch/ Rename	ChangeMan assumes the Scratch and the Rename functionalities during the Promote. Will be supported through a custom script.
Extension	Library type	The extension/type determines the processing needed for a certain type.
Not available	Impact-Analysis	ChangeMan assumes Source, Copy, JCL, Proc and DS names relationships. Using CloudBees, a special development can be done for creating the I-A table from the VCR and an ALM Report can be created for consulting purposes.
Not available	Merge & Reconcile	CloudBees does not need to support this because it is a task of the VCR.

As Archive	Freeze	In relation to this ChangeMan concept, IKAN Pipelines for z/OS Scripts creates an Archive at the end of every Build containing all components to deploy. It is this Archive that used for the next.
Not available	Baseline	It is a ChangeMan concept that duplicates (or not) the Production Level used for future package developments considering it is the version 0 as Reference in Production. CloudBees doesn't assume this concept because it is the VCR task to define the versions of components. CloudBees creates or presents the Tag for a Build version.

Appendix IV: Available z/OS IKAN scripts

The following table maps the Phases used by IKAN Pipelines for z/OS scripts for compiling and deploying components to Mainframe Environments. Note for IDMS, a script collects the dictionary components and the next script installs them into another one.

Script	Action	Description
z/OS Allocate PDS	Build/ Deploy	Script for creating/deleting PDSs into an environment.
z/OS Programs Compilation	Build	For Maps (firstly) and Programs (secondly), the Script generates a compile JCL depending on the Source contents and language using included JCL models. Next, each JCL is executed by JES in parallel under FTP and the resulting Job is analyzed to know its status. Next, the generated compile Listing, Load-module and DB2 DBRM, Datacom Plan are transferred to the IKAN ALM Target Environment. Optionally, DB2 Binds may be generated from models. Note that for CA-Endevor the Repository will be updated for compiling with it. Included Delete Package PDSs. Default Languages: ASSEMBLE, BMS, COBOL, PLI, SDF2, REXX). Other Custom languages may be included easily.
z/OS copy Sources before Compilation	Build	This Script transfers, via FTP, Copybooks, Linkedit Control Cards (LCT cards), Programs & Load-modules and BMS/SDF2 Maps to PDS(s) in the Mainframe Environment. Included Allocate Package PDSs for compiling.
z/OS Copy Pds Members	Deploy	This script transfers components from PDS(s) of a z/OS environment to PDS(s) of another z/OS environment using the components list in the IKAN ALM target environment.
z/OS copy Source to Target	Build/ Deploy	A dedicated Script for copying the z/OS Components (Sources or Objects) to the IKAN ALM Target Environment.This Script only transfers selected component types.
z/OS DCF Deploy	Deploy	This script copies DCF components to other PDS in the IKAN ALM target environment.

z/OS Delete Sources and associated objects	Deploy	All Sources identified by the "to_be_deleted" suffix are deleted in PDS(s) of the Mainframe Environment by FTP. Also, the associated Listings, Load-modules, DBRMs, Plans and DB2 Binds are deleted in their PDS(s). No action in DB2 and Datacom Databases.
z/OS Demotion of components and load-modules	Deploy	z/OS components in the IKAN ALM Package archive are deleted from their PDS(s) of the Mainframe Environment. This Script have to be followed by a Backup using the z/OS Copy PDS members Script.
z/OS Control JMODEL	Build	This script controls with changes the JMAN components.
z/OS Deploy JMODEL	Deploy	This script deploys with changes the JMAN components.
z/OS Promotion of components and load-modules	Deploy	z/OS components in the IKAN ALM Package archive are transferred to their PDS(s) of the Mainframe Environment. Exception: the Load-modules which are transferred to flat files before a generated JCL using included JCL models is executed by JES under FTP for receiving them in their PDS(s).
z/OS QMF Build	Build	This script prepares the QMF components.
z/OS QMF Deploy	Deploy	This script deploys with changes the QMF components.
z/OS DB2 Binds transfers and activation	Deploy	If DB2 is used, Bind files are copied to their PDS(s) and a JCL is generated using included JCL models and executed by JES under FTP for running these Binds on the DB2 Database.
z/OS CICS Load-modules activation	Deploy	If there are CICS Maps or Programs, a JCL is generated using included JCL models, and executed by JES under FTP for running the PHASEIN commands on a CICS.
z/OS Update Datacom components Promotion	Deploy	If there are Plans, the Script generates a JCL using included JCL models and executed by JES under FTP for importing Plans on the Datacom Database.
z/OS Update Endevor components Promotion	Deploy	If the CA-Endevor Repository is active on the Mainframe, the Script generates a JCL using included JCL models and executed by JES under FTP for moving components from the Stage ID to the corresponding Level.
z/OS SQL DB2 updates Execution	Deploy	If DB2 is used, DDL and SQL statements may be applied with variable substitutions as owner, qualifier. After the transfer of DDL and SQL commands concatenated into 2 members, 2 JCLs are generated using included JCL models and executed by JES under FTP for running DDL and, next, SQL on the DB2 Database.
z/OS Update Debugger	Deploy	For instance, for the Xpediter tool,, this script copies Xpediter components from a FILEIO file to another FILEIO using the components list in the IKAN ALM target environment.

z/OS Collect IDMS components	Build	For the first build, this Script generates a JCL using included JCL models that is executed in the Mainframe Environment by JES under FTP. This one collects IDMS components and info about date and parent relations in the IDD of development. For the rebuild before deployment, the Script controls the correlation with the target IDD with execution of another generated JCL.
z/OS IDMS components Promotion	Deploy	This script transfers components to temporary files in the Mainframe Environment and she generates a JCL using included JCL models that is executed by JES under FTP for updating the target IDD.

Appendix V: Sample of z/OS compilation JCL

The following JCL is fully generated by the z/OS Compilation Phase used by IKAN Pipelines for compiling a component into the Mainframe Environment.

```
//ADCDMSTC JOB (5145,00000,2233,T),'IKAN',
//         MSGLEVEL=(1,1),MSGCLASS=X,
//         CLASS=A,REGION=8M
//*
//*XEQ ROUTEID=ADCD
//*****************************************************************
//**   COPYING THE PROGRAM IN SOURCE WORK FILE                   **
//*****************************************************************
//     SET SRCOMPIL=SOURCE
//COPYSRC  EXEC PGM=IEBGENER
//SYSTSIN   DD DUMMY
//SYSTSPRT  DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSUT1    DD  DISP=(SHR),
//              DSN=IKANALM.DEMOS.TEST.SRCBATCH(DEMO21)
//SYSUT2    DD  DISP=(,PASS),DSN=&&&SRCOMPIL,
//             UNIT=SYSDA,SPACE=(CYL,(10,10)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=0)
//SYSIN     DD  DUMMY
//*****************************************************************
//**    COMPILE COBOL2, store object in objlib if compile=ok
//**    compile listing is stored in IKANALM.DEMOS.TEST.LSTALIB
//*****************************************************************
//         SET PARMCOB='LIST,LIB,NOSEQ,NOCMPR2,MAP'
//         SET PARMCOB0='DATA(31)'
//*****************************************************************
//**    COMPILE THE ELEMENT                                       **
//*****************************************************************
//COBOL  EXEC PGM=IGYCRCTL,COND=(4,LT),
//   PARM='&PARMCOB0,&PARMCOB'
//STEPLIB  DD  DISP=SHR,DSN=SYS1.COB2COMP
```

```
//SYSIN     DD  DISP=(OLD,PASS),DSN=&&&SRCOMPIL
//SYSLIN    DD  DISP=(,PASS),DSN=&&OBJECT,
//              UNIT=SYSDA,SPACE=(CYL,(2,2)),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=0)
//SYSUT1    DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT2    DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT3    DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT4    DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT5    DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT6    DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSUT7    DD  UNIT=SYSDA,SPACE=(CYL,(5,3))
//SYSPRINT DD  DISP=(,PASS),DSN=&&COMPLIST,
//             UNIT=SYSDA,SPACE=(TRK,(10,10),RLSE)
//*             DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0)
//*
//SYSLIB    DD  DISP=SHR,DSN=IKANALM.DEMOS.TEST.COPYLIB
//          DD  DISP=SHR,DSN=IKANALM.DEMOS.INTG.COPYLIB
//          DD  DISP=SHR,DSN=IKANALM.DEMOS.QUAL.COPYLIB
//          DD  DISP=SHR,DSN=IKANALM.DEMOS.PROD.COPYLIB
//*
//*
//*
//*
//*
//*
//*****************************************************************
//**   COPYING THE LCT MEMBER IN A WORK FILE IF EXIST              **
//*****************************************************************
//ALLOCLCT EXEC PGM=IEFBR14
//SYSPRINT DD  SYSOUT=*
//LCTFILE  DD  DISP=(NEW,PASS,DELETE),DSN=&&LCTFILE,
//             UNIT=SYSDA,SPACE=(TRK,(1,1,1)),
//             DCB=(DSORG=PO,RECFM=FB,LRECL=80,BLKSIZE=0)
//CREATLCT EXEC PGM=IEBGENER
//SYSPRINT  DD SYSOUT=*
//SYSUT1    DD  *
/*
//SYSUT2    DD  DISP=(MOD,PASS),DSN=&&LCTFILE(DEMO21)
//SYSIN     DD  DUMMY
//COPYLCT EXEC PGM=IEBCOPY
//SYSPRINT  DD SYSOUT=*
//INDD00    DD DISP=SHR,DSN=IKANALM.DEMOS.TEST.LCTLIB
//INDD01    DD DISP=SHR,DSN=IKANALM.DEMOS.INTG.LCTLIB
//INDD02    DD DISP=SHR,DSN=IKANALM.DEMOS.QUAL.LCTLIB
//INDD03    DD DISP=SHR,DSN=IKANALM.DEMOS.PROD.LCTLIB
//*
//OUTDD1    DD  DISP=(MOD,PASS),DSN=&&LCTFILE
//SYSIN     DD  *
  COPY OUTDD=OUTDD1
       INDD=INDD00,INDD01,INDD02,INDD03
```

```
  SELECT MEMBER=DEMO21
/*
//*****************************************************************
//**   LINKEDIT PROGRAM                                          **
//*****************************************************************
//      SET PARMLNK='LIST,MAP,XREF,NCAL'
//      SET LINKOPT='RENT,AMODE(31),RMODE(ANY),'
//LKEDT    EXEC PGM=HEWL,COND=(4,LT),
//         PARM='&PARMLNK,&LINKOPT'
//SYSLMOD  DD DISP=SHR,DSN=IKANALM.DEMOS.TEST.LOADLIB(DEMO211)
//SYSDEFSD DD DUMMY
//SYSPRINT DD  DISP=(,PASS),DSN=&&LINKLIST,
//            UNIT=VIO,SPACE=(TRK,(10,10)),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=0)
//*        DD DISP=SHR,DSN=IKANALM.DEMOS.TEST.LOADLIB(DEMO211)
//SYSLIB   DD  DISP=SHR,DSN=IKANALM.DEMOS.TEST.LOADLIB
//         DD  DISP=SHR,DSN=IKANALM.DEMOS.INTG.LOADLIB
//         DD  DISP=SHR,DSN=IKANALM.DEMOS.QUAL.LOADLIB
//         DD  DISP=SHR,DSN=IKANALM.DEMOS.PROD.LOADLIB
//*
//         DD DISP=SHR,DSN=DFH320.CICS.SDFHLOAD
//         DD DISP=SHR,DSN=DSN810.SDSNLOAD
//         DD DISP=SHR,DSN=CEE.SCEELKED
//*         DD DISP=SHR,DSN=METASUIT.GEN813.LOADLIB
//*         DD DISP=SHR,DSN=SYS1.COB2LIB
//         DD DISP=SHR,DSN=SYS1.LINKLIB
//SYSLIN   DD  *
/*
//         DD  DSN=&&OBJECT,DISP=(OLD,DELETE)
//         DD  DSN=&&LCTFILE(DEMO21),DISP=(OLD,DELETE)
//         DD  *
  IDENTIFY DEMO211('DEMO21/ADCDMST/000003')
  NAME DEMO211(R)
/*
//*****************************************************************
//**   TRANSMIT PROGRAM                                          **
//*****************************************************************
//CLEARSEQ  EXEC PGM=IEFBR14
//DD01   DD DISP=(MOD,DELETE,DELETE),
//          DSN=IKANALM.DEMOS.TEST.DEMO211,
//          UNIT=SYSDA,SPACE=(TRK,(1)),
//          LRECL=80,BLKSIZE=3120,RECFM=FB
//*
//XMITLOAD  EXEC PGM=IKJEFT01,COND=(4,LT)
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 XMIT (ADCD.*) -
 DSNAME('IKANALM.DEMOS.TEST.LOADLIB') MEM(DEMO211)-
 OUTDSNAME('IKANALM.DEMOS.TEST.DEMO211') NOLOG NONOTIFY
```

```
/*
//PRTCMPA  IF (COBOL.RUN EQ TRUE) THEN
//*****************************************************************
//**    PRINT THE COMPILE LISTING                                **
//*****************************************************************
//PRNTCOMP EXEC PGM=IEBGENER
//SYSTSIN    DD DUMMY
//SYSTSPRT   DD SYSOUT=*
//SYSPRINT   DD SYSOUT=*
//SYSUT1     DD  DISP=(OLD,PASS),DSN=&&COMPLIST
//SYSUT2     DD  SYSOUT=*
//SYSIN      DD  DUMMY
//PRTCMPZ  ENDIF
//*
//PRTLNKA  IF (LKEDT.RUN EQ TRUE) THEN
//*****************************************************************
//**    PRINT THE LINKEDIT LISTING                               **
//*****************************************************************
//PRNTLINK EXEC PGM=IEBGENER
//SYSTSIN    DD DUMMY
//SYSTSPRT   DD SYSOUT=*
//SYSPRINT   DD SYSOUT=*
//SYSUT1     DD  DISP=(OLD,PASS),DSN=&&LINKLIST
//SYSUT2     DD  SYSOUT=*
//SYSIN      DD  DUMMY
//*****************************************************************
//**     FORMAT THE LINKEDIT LISTING                             **
//*****************************************************************
//IFSFTLKD IF (NOT ABEND) THEN
//FRMTLKD  EXEC PGM=SORT
//SORTSNAP DD  SYSOUT=*
//SORTWK01 DD  DISP=(,PASS),UNIT=SYSDA,SPACE=(CYL,(10,10),RLSE)
//SORTWK02 DD  DISP=(,PASS),UNIT=SYSDA,SPACE=(CYL,(10,10),RLSE)
//SORTWK03 DD  DISP=(,PASS),UNIT=SYSDA,SPACE=(CYL,(10,10),RLSE)
//SORTWK04 DD  DISP=(,PASS),UNIT=SYSDA,SPACE=(CYL,(10,10),RLSE)
//SORTIN   DD  DISP=(OLD,DELETE),DSN=&&LINKLIST
//SORTOUT  DD  DISP=(NEW,PASS,DELETE),DSN=&&LISTLKD,
//             UNIT=SYSDA,SPACE=(CYL,(5,5)),
//             DCB=(DSORG=PS,RECFM=FBA,LRECL=133,BLKSIZE=0)
//SYSIN    DD  *
  SORT FIELDS=COPY
  OUTREC FIELDS=(1,121,12X)
/*
//SYSOUT   DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSMDUMP DD  SYSOUT=*
//SYSABEND DD  SYSOUT=*
//IFEFTLKD  ENDIF
//*****************************************************************
//**     COPY THE LISTINGS                                       **
```

```
//*****************************************************************
//IFSLST1  IF (NOT ABEND) THEN
//LIST    EXEC PGM=IEBGENER
//SYSPRINT  DD SYSOUT=*
//SYSUT2    DD DISP=SHR,
//             DSN=IKANALM.DEMOS.TEST.LSTALIB(DEMO21)
//SYSIN     DD DUMMY
//SYSUT1  DD  DISP=(NEW,DELETE,DELETE),DSN=&&NULLSEQ,
//             UNIT=SYSDA,SPACE=(TRK,(1,1)),
//             DCB=(DSORG=PS,RECFM=FBA,LRECL=133,BLKSIZE=0)
//*          DD DISP=(OLD,DELETE),DSN=&&PCMPLIST
//           DD DISP=(OLD,DELETE),DSN=&&COMPLIST
//           DD DISP=(OLD,DELETE),DSN=&&LISTLKD
//IFELST1  ENDIF
//*
//*
//IFSFAIL  IF (RC GT 4 OR ABEND) THEN
//FAILURE  EXEC PGM=IEBGENER,MAXRC=0
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  *
  JOB 1626 FAILED
/*
//SYSUT2   DD  SYSOUT=*
//SYSIN    DD  DUMMY
//IFEFAIL  ENDIF
//PRTLKDZ  ENDIF
```

IKAN Development N.V.
Kardinaal Mercierplein 2
2800 Mechelen
Tel. +32 15 797306
info@ikan.be
www.ikan.be

IKAN