Jenkins pipeline

Build and Deploy

Contents

[**1.** **Jenkins Build Pipeline** 2](#_Toc535942780)

[1. Checkout stage 2](#_Toc535942781)

[2. Build Stage 2](#_Toc535942782)

[3. Security Checks Stage 4](#_Toc535942783)

[4. Deployment Stage 6](#_Toc535942784)

[5. Functional Checks Stage 6](#_Toc535942785)

[6. Upload artifacts to archiva Stage 7](#_Toc535942786)

[**2.** **Jenkins Deploy Pipeline** 8](#_Toc535942787)

[1. Jenkins – Active Choice Parameter Configuration 8](#_Toc535942788)

[2. Download artifacts from archiva 11](#_Toc535942789)

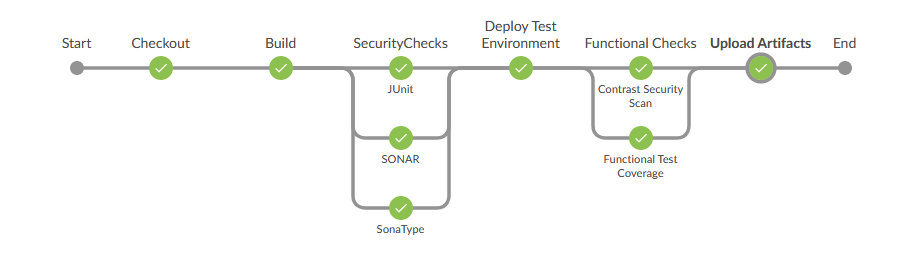
[3. Deploy Approval 12](#_Toc535942790)

[4. deployment 13](#_Toc535942791)

[5. Test complete – Smoke suite execution 13](#_Toc535942792)

[6. Cacti monitoring 13](#_Toc535942793)

# **Jenkins Build Pipeline**



## Checkout stage

1.1 Jenkins Pipeline - Checkout

1. stage('Checkout') {
2. node {
3. sh '''cd /home/wasadmin/release/jenkinsworkspace/Final\_Brokerage\_Build\_8\_2\_1
4. rm -rf Final\_Brokerage\_Build\_8\_2\_1'''
5. checkout([$**class**: 'SubversionSCM', additionalCredentials: [], excludedCommitMessages: '', excludedRegions: '', excludedRevprop: '', excludedUsers: '', filterChangelog: **false**, ignoreDirPropChanges: **false**, includedRegions: '', locations: [[credentialsId: 'ed5b09d9-3219-46ad-9ab7-02c359239e0a', depthOption: 'infinity', ignoreExternalsOption: **true**, local: '.', remote: 'http://10.11.100.160:85/folio/Brokerage/branches/FDX\_BRN\_DEV\_8\_2\_1']], quietOperation: **false**, workspaceUpdater: [$**class**: 'CheckoutUpdater']])
6. sh '''chmod -R 777 /home/wasadmin/release/jenkinsworkspace/Final\_Brokerage\_Build\_8\_2\_1'''
7. echo "Checkout complete"
8. }
9. }

1.2 Details - Checkout

* During Checkout Phase of Build Pipeline, the system will checkout the repository mentioned (in this case ‘'http://10.11.100.160:85/folio/Brokerage/branches/FDX\_BRN\_DEV\_8\_2\_1’)
* The credentials are stored in Jenkins -> “Credentials”.

## Build Stage

2.1 Jenkins Pipeline - BUILD

1. stage('Build') {
2. node {
3. sh '''cd /home/wasadmin/release/scripts/Final\_Brokerage\_Build\_8\_2\_1
4. sh dy\_build\_maven.sh'''
5. jacoco classPattern: '\*\*/target/classes', exclusionPattern: '\*\*/\*Test\*.class', execPattern: '\*\*/target/jacoco.exec', inclusionPattern: '\*\*/\*.class'
6. echo "Build complete"
7. }
8. }

2.2 Explanation - Build

* Invoke dy\_build\_maven.sh script – for performing the build.(Explained in below section 2.3)
* Run Jacoco for performing code coverage.
  1. Code – env\_variables.sh

1. #Environment Variables
2. echo "----- Setting Environment Variables -----"
3. echo "Build Version" **$BUILD\_VERSION**
4. BUILD\_VERSION=8\_2\_1; export BUILD\_VERSION
5. VERSION\_NUMBER=8.2.1; export VERSION\_NUMBER
6. JENKINS\_JOB=Final\_Brokerage\_Build\_8\_2\_1; export JENKINS\_JOB
7. JENKINS\_WORKSPAGE=/home/wasadmin/release/jenkinsworkspace/**$JENKINS\_JOB**; export JENKINS\_WORKSPAGE
8. SETTINGS\_FILE=**$JENKINS\_WORKSPAGE**/fdBrokerage/conf/settings.xml; export SETTINGS\_FILE
9. RELEASE\_DIRECTORY=/home/wasadmin/release/projects/fdx; export RELEASE\_DIRECTORY
10. EJB\_DEPLOY=/opt/IBM/WebSphere/AppServer/bin; export EJB\_DEPLOY
11. COMMON\_LIB=**$JENKINS\_WORKSPAGE**/fdxLib/common; export COMMON\_LIB
12. UTIL\_JAR=**$JENKINS\_WORKSPAGE**/fdUtil/target; export UTIL\_JAR
13. EXTERNAL\_LIB=/home/wasadmin/release/external/lib; export EXTERNAL\_LIB
14. EAR\_DIR=**$JENKINS\_WORKSPAGE**/fdBrk/target; export EAR\_DIR
15. SONAR\_RESULT\_FILE=/home/wasadmin/release/scripts/Final\_Brokerage\_Build\_8\_2\_1/sonar-result.txt; export SONAR\_RESULT\_FILE
17. JAVA\_HOME=/opt/IBM/WebSphere/AppServer/java\_1.7\_64; export JAVA\_HOME
18. MVN\_HOME=/home/wasadmin/release/apache-maven-3.5.2; export MVN\_HOME
19. PATH=**$PATH**:**$JAVA\_HOME**/bin/:**$MVN\_HOME**/bin/; export PATH

22. #PATCH DEPLOYMENT Variables
23. EJB\_PRE\_EAR="preejb\_fdEJB-\*.jar"; export EJB\_PRE\_EAR
24. EAR\_FILE="fdEJB-\*.jar"; export EAR\_FILE
26. FDX\_BUILD\_HOME=/home/wasadmin/release; export FDX\_BUILD\_HOME
27. FDX\_RELEASE\_HOME=**$FDX\_BUILD\_HOME**/projects/fdx; export FDX\_RELEASE\_HOME
28. CORE\_SOURCE\_DIR=**$FDX\_BUILD\_HOME**/projects/fdx/**$RELEASE**/source; export CORE\_SOURCE\_DIR
29. CORE\_BUILD\_DIR=**$FDX\_BUILD\_HOME**/projects/fdx/**$RELEASE**/build; export CORE\_BUILD\_DIR
30. MAVEN\_REP=/fdx/FDX\_MVN\_REPO\_8\_2\_1/com/fd/brokerage; export MAVEN\_REP
32. JENKINS\_JOB\_URL=http://10.20.40.166:8080/job/Final\_Brokerage\_Build\_8\_2\_1/lastSuccessfulBuild/buildNumber; export JENKINS\_JOB\_URL
    1. Explanation- env\_variables.sh

* Need to set the environment variables.
* These variables would be used in all other shell scripts.

2.5 Code – dy\_build\_maven.sh

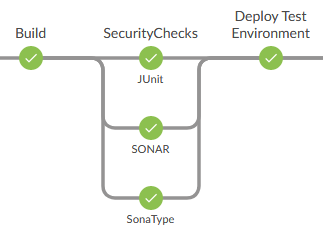


* 1. Explanation – Dy\_build\_maven.sh
* Line 5 – 16, sets various environment variables.
* Line 17, creates a temp file build\_version.txt which will hold the current version of build.
* Line 19 – 297, sets dynamic version number for all the modules at run time. This is required till we rollout to production.
  + When rolling out to production, we need to add these parameters into pom.xml file
  + Once added to pom.xml, we need to set dynamic versions during every run.
* Line 299 – 332, updates application.xml with corresponding version number for the modules.
* Line 333 – 339, updates fdxBuildDeployMaven.pm script with respective version number changes.
* Line 340 – 342, modifies fdDbProcedures->build.xml file (not required in production – this step is required in test Jenkins servers because the jenkinsworkspace path is different)
* Line 344 – 351, cleans up previously prepared artifacts.
* Line 356, Builds the application.
  + It skips the testcases as this would be executing as a separate step
* Line 376 – Performs EJBDeploy

## Security Checks Stage

* 1. Jenkins Pipeline – Security checks

1. stage('SecurityChecks') {
2. node {
3. parallel JUnit: {
4. sh '''cd /home/wasadmin/release/scripts/Final\_Brokerage\_Build\_8\_2\_1
5. sh junit.sh'''
6. echo "Junit Complete"
7. },
8. SONAR: {
9. timeout(time: 1, unit: 'HOURS') {
10. sh '''cd /home/wasadmin/release/sonar-runner-2.4/bin
11. ./sonar-runner -Dproject.settings=/home/wasadmin/release/sonar-runner-2.4/Projects/sonar-project.properties'''
12. sh 'echo "Sonar Scanning completed.................."'
13. sh '''cd /home/wasadmin/release/scripts/Final\_Brokerage\_Build\_8\_2\_1
14. ./sonar\_publisher.sh'''
15. }
16. sh 'echo "Sonar Publishing completed"'
17. },
18. SonaType: {
19. sh 'echo "Sonar Type completed.................."'
20. }
21. }
22. }
    1. Explanation – Security checks



* The section would run the Junit testcases, SONAR and SonaType in parallel by invoking respective scripts.
* 
* Junit.sh
  + Line 6 – lists the projects which has Junit test cases. And executes the test cases
* Sonar\_publisher.sh
  + Line 9-11 – defines SONAR URL corresponding to BLOCKER, CRITICAL and MAJOR bugs.
  + Line 16 – Checks ‘.sonar/report-task.txt’ file which will be automatically created once SONAR scan happens. This file would have the URL which need to be used for retrieving the results.
  + Once it finds report-task.txt file, the script polls the URL found in the file until the status changes from IN-PROGRESS to SUCCESS/FAILURE.
  + Once the status is changed to SUCCESS, it would retrieve the results and checks with the previous result (present in sonar-result.txt file).
    - If the current result(BLOCKER, CRITICAL, MAJOR) is less than or equal to the previous results, the STAGE would be marked as SUCCESS
    - ELSE the BUILD would be marked as FAILED
    - If sonar-result.txt file is not present it creates and sets the current results as the base values.

## Deployment Stage

4.1 Jenkins Pipeline - Deployment

1. stage('Deploy Test Environment') {
2. node{
3. sh ''' cd /home/wasadmin/release/scripts/Final\_Brokerage\_Build\_8\_2\_1
4. sh deploy\_artifacts.sh'''
5. sh 'echo "Deployed completed"'
6. }
7. }

4.2 Explanation – Deployment

* This would be the current deployment script used in production Jenkins.

## Functional Checks Stage

5.1 Jenkins CODE – FUNCTIONAL Checks

1. stage('Functional Checks') {
2. node {
3. parallel 'Functional Test Coverage': {
4. triggerRemoteJob abortTriggeredJob: **true**, enhancedLogging: **true**, job: 'TestComplete', maxConn: 1, remoteJenkinsName: 'TestComplete133',propagate: **true**
5. sh 'echo "Functional Test completed"'
6. }, 'Contrast Security Scan': {
7. sh 'echo "Contrast Scan completed"'
8. }
9. }
10. }

5.2 EXPLANATION – Functional Checks

* Functional Test coverage is performed by TestComplete.
  + We need to perform below steps for configuring TestComplete

<https://support.smartbear.com/testcomplete/docs/working-with/integration/jenkins/preparing.html>

<https://support.smartbear.com/testcomplete/docs/working-with/integration/jenkins/requirements-and-limitations.html>

* Contrast Security Scan – was a mock step

## Upload artifacts to archiva Stage

6.1 Jenkins Code – Upload Artifacts to Archiva

1. stage('Upload Artifacts') {
2. node {
3. sh '''cd /home/wasadmin/release/scripts/Final\_Brokerage\_Build\_8\_2\_1
4. sh upload\_artifacts.sh'''
5. sh 'echo "Upload completed"'
6. }
7. }

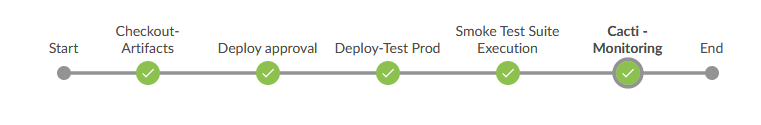
6.2 Explanation



* Need to add the repository information in setting.xml and pom.xml
* Line4 – does maven deploy, which would perform uploading of artifacts to archiva.

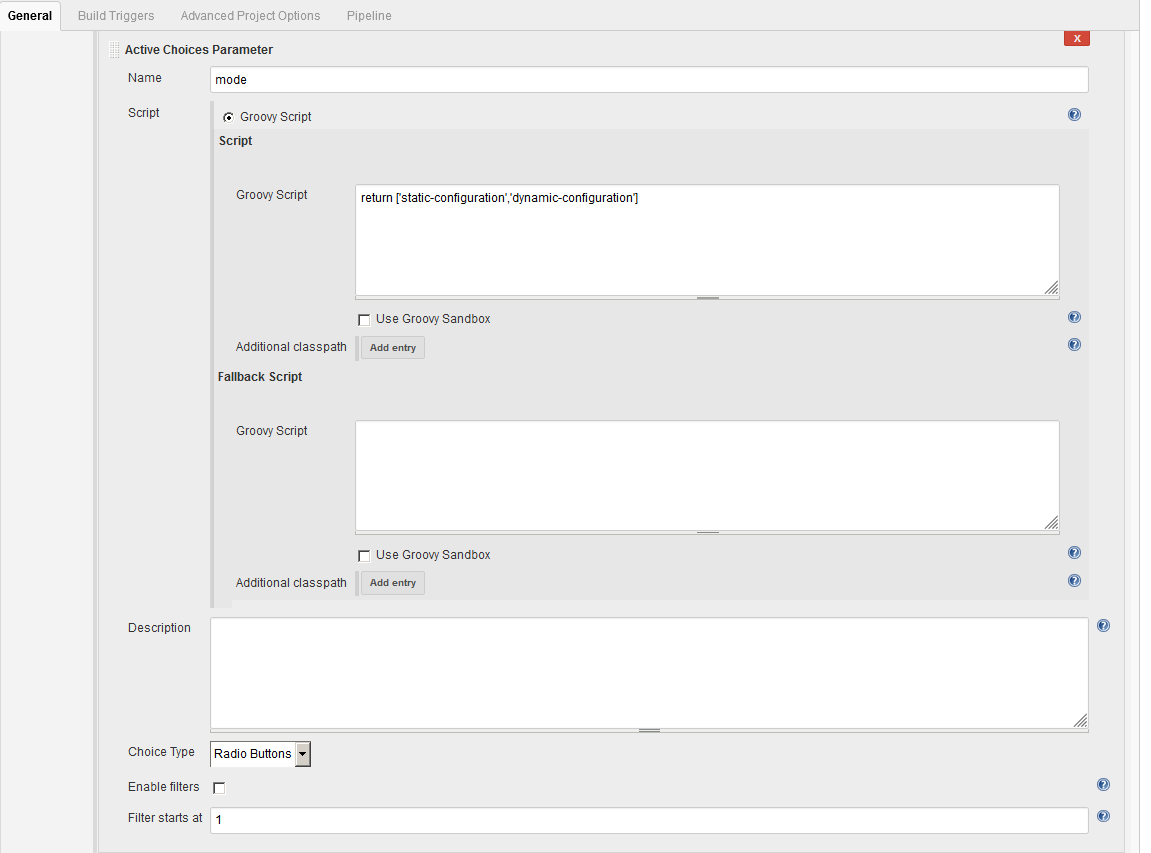
|  |  |
| --- | --- |
| ump-snapshot-local | Holds the latest develop version of artifacts e.g. the latest UMP war file created from the successful commit pipeline than ran when a merge request was approved and the changes were merged to the latest integration branch |
| ump-staging-local | Holds release candidates. These are the artifacts typically tested in QA, UAT or QL |
| ump-release-local | Holds the artifacts released to Production |

# **Jenkins Deploy Pipeline**

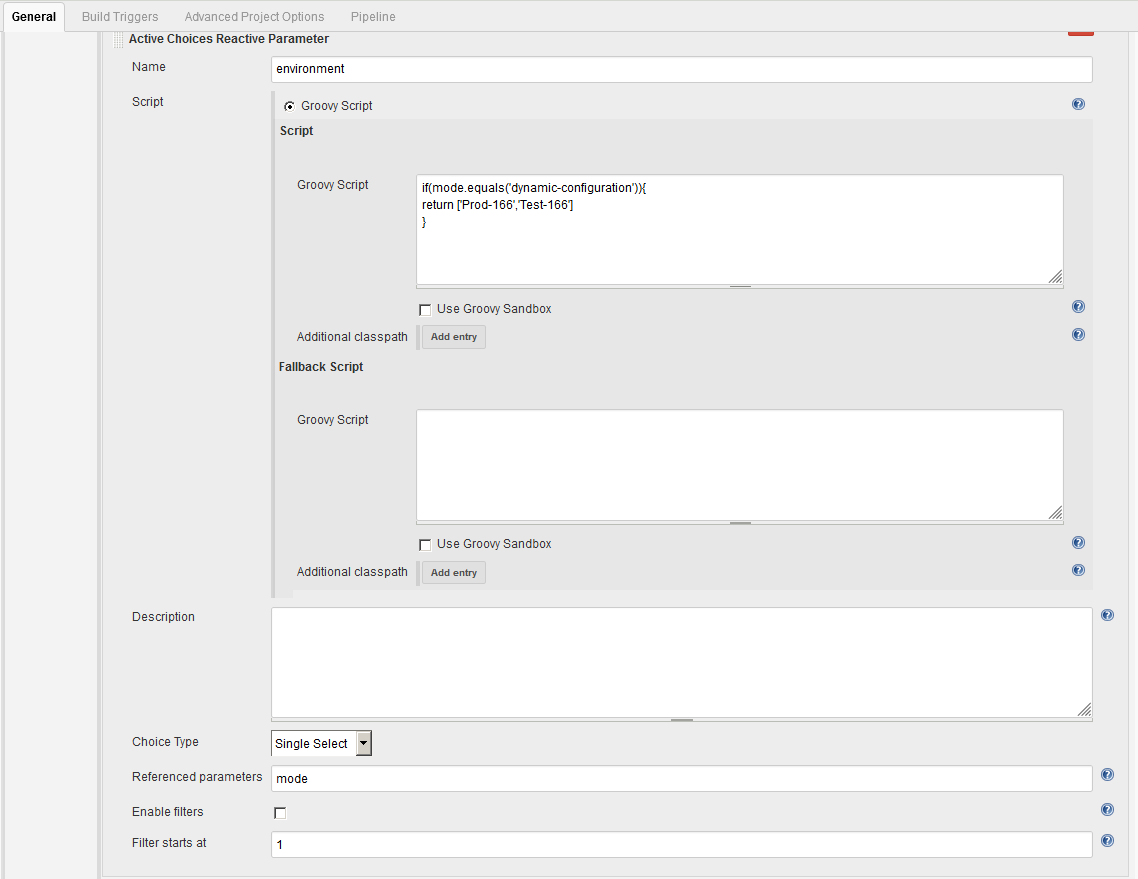


## Jenkins – Active Choice Parameter Configuration

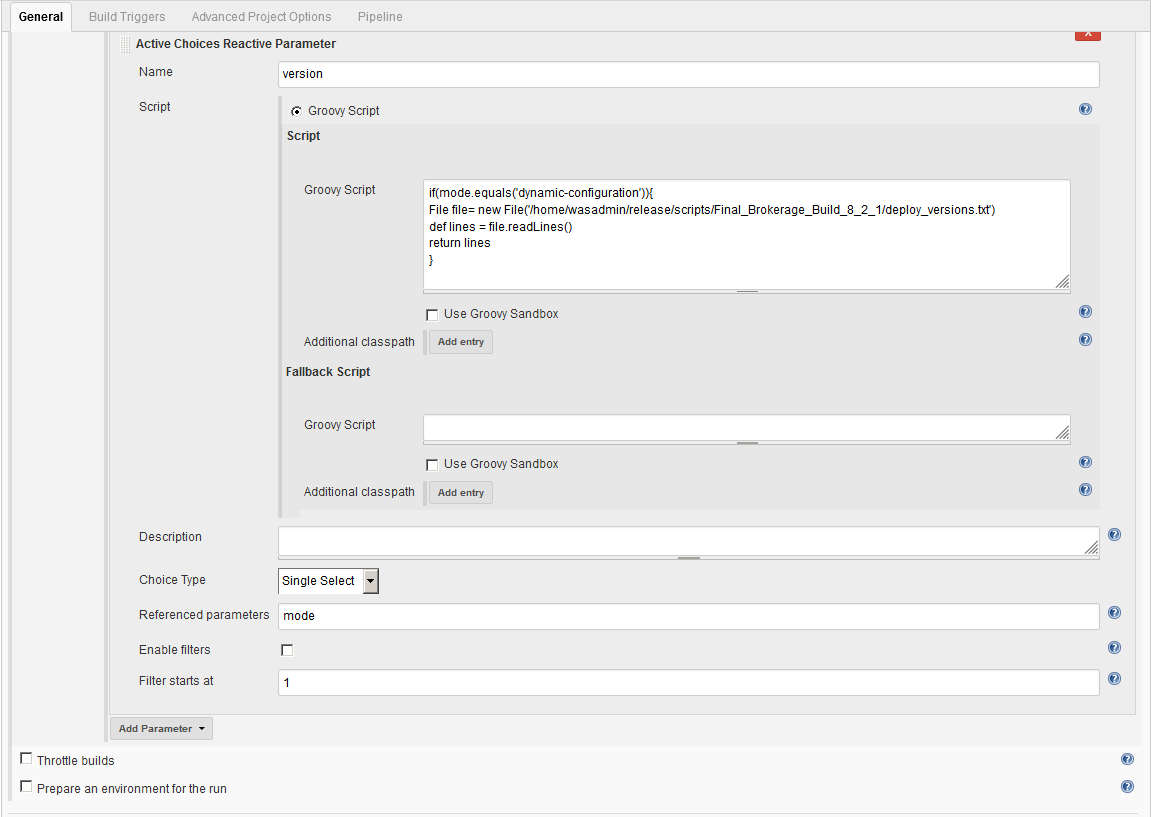
Mode configuration



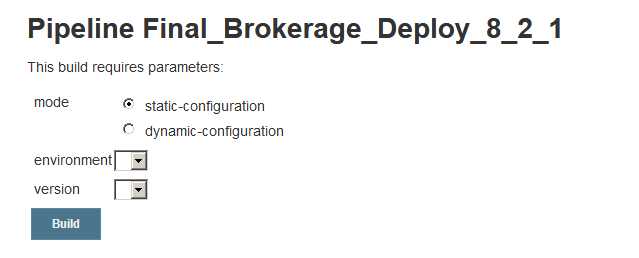
Environment Configuration



Version Configuration

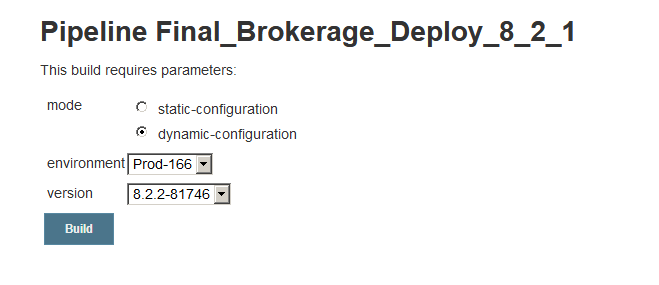


Static configuration



* Static configuration uses a config-deploy.txt
* Later it would use timer.sh to schedule the deployment during the respective time. 

Dynamic configuration



* Once the build is success after all the Build stages, it would add the respective version into deploy\_version.txt file 
* Version drop-down box would be populated with the values of deploy\_version.txt file.
* Deploy\_version.txt file would have the main revision. Version corresponding to each module, would be a part of deploy\_module\_version.txt. This would map <main version> with individual module version.

## Download artifacts from archiva

Jenkins Pipeline – Download Artifacts

1. stage('Checkout-Artifacts') {
2. node{
3. sh ''' cd /home/wasadmin/release/scripts/Final\_Brokerage\_Build\_8\_2\_1
4. sh download\_artifacts.sh'''
5. echo "Checkout complete"
6. }
7. }

Explanation – Download Artifacts

Executes download\_artifacts.sh file

Script – download\_artifacts.sh

mvn -fae --settings conf/settings.xml --global-settings conf/settings.xml clean dependency:copy-dependencies



Explanation – Download Artifacts

* Line 11 - 55 – Need to update the versions that need to be downloaded.
* Line 312 – Change the output directory

## Deploy Approval

Jenkins Pipeline – Deploy approval

1. stage('Deploy approval'){
2. def file = **new** File('/home/wasadmin/release/scripts/Final\_Brokerage\_Build\_8\_2\_1/approvalenv.txt')
3. def lines = file.readLines()
4. echo "--${params.environment}--"
5. def found
6. lines.find{line->
7. found = params.Environment == line ? 1 : 0
8. echo "found : --${found}--"
9. echo "line : --${line}--"
10. echo "params : --${params.environment}--"
11. **if**(params.Environment != line){
12. input(message: "Deploy to QUAT?", ok: 'Deploy', submitter: 'testuser')
13. echo "Finished this stage"
14. }
15. **else**{
16. echo "Approval Not Required"
17. }
18. }
19. echo "Approval complete"
20. }

Explanation – Deploy approval



* approvalenv.txt file has the environments which requires approval.
* While Building, if we have selected Dynamic Mode – Based on the environment we selected, the system would ask for *‘approval’*
* Once approved only the system would allow for deployment.

## deployment

Jenkins Pipeline – Deployment

1. stage('Deploy-Test Prod') {
2. node{
3. sh ''' cd /home/wasadmin/release/scripts/Final\_Brokerage\_Build\_8\_2\_1
4. sh deploy\_artifacts.sh'''
5. sh 'echo "deployment completed" '
6. }
7. }

Explanation – Deployment

* It would call the deployment scripts, which is used in production.

## Test complete – Smoke suite execution

Same as Build -> Test Complete Integration

## Cacti monitoring

Jenkins Pipeline – Cacti Monitoring

1. stage('Cacti - Monitoring') {
2. node{
3. echo "http://10.20.40.165/cacti/graph\_view.php?action=tree&node=tbranch-1&hgd=&hyper=true"
4. }
5. }