NewgenONE OmniDocs

Docker Containers Custom Code Deployment Guide for Azure

Version: 11.3

Newgen Software Technologies Ltd.

Table of contents

1 Preface

This guide describes the containerization approach for Newgen Enterprise products, which thus details out the container-based deployment architecture and container-based DevOps pipeline for Newgen flagship products OmniDocs on Microsoft Azure.

1.1 Revision history

Revision Date	Description
July 2024	Initial publication

1.2 Intended audience

This guide is intended for Cloud Administrators, System Administrators, developers, and all other users who are seeking information on the deployment of hotfix for container based OmniDocs. The reader must be comfortable understanding the computer terminology.

1.3 Documentation feedback

To provide feedback or any improvement suggestions on technical documentation, you can write an email to docs.feedback@newgensoft.com.

To help capture your feedback effectively, requesting you to share the following information in your email.

- Document Name
- Version
- Chapter, Topic, or Section
- Feedback or Suggestions

1.4 Third-party product information

This guide contains third-party product information about configuring Microsoft Azure CICD Pipeline for Container Deployment on AKS Azure Kubernetes Cluster. Newgen Software Technologies Ltd does not claim any ownership on such third-party content. This information is shared in this guide only for convenience of our users and could be an excerpt from the Azure documentation. For latest information on configuring the Azure Kubernetes Cluster and Azure DevOps refer to the Azure documentation.

2 CI/CD pipeline

The CICD pipeline manages the hotfix deployments with Kubernetes orchestration on cloud platforms. Here, the separation of the Build Pipeline and Release Pipeline is done into two parts. The Build Pipeline is done by the Jenkins server that can be installed on-premises or in a cloud VM. The Release pipeline is managed by Azure DevOps cloud service. In this architecture, there are three stages Dev, UAT, and Production and on each stage, deployment is quite different. More stages can be added depending on the requirements.

2.1 CI/CD Pipeline for Custom Code

This section described the CI/CD Pipeline for Custom Code.

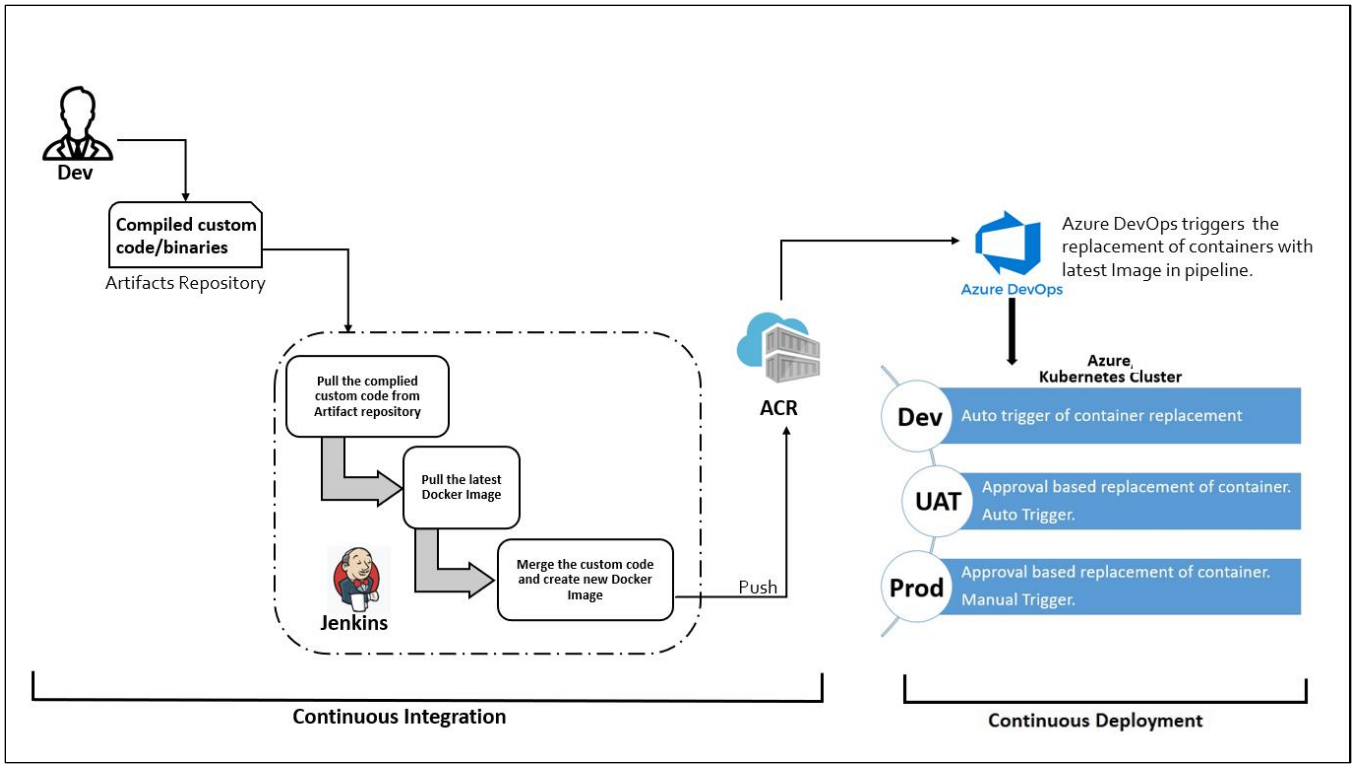

Figure 2.1

- The Custom Code deployment pipeline can also be configured and maintained in Jenkins.
- For initiating custom code deployment, the Implementation team pushes the compiled custom code to the artifacts repository. An artifacts repository can be Azure FileShare, SVN, FTP, JFrog, and so on.

- After that Jenkins pulls the compiled custom code from the artifacts repository and pull the latest Docker image. Then, it creates a new Docker after merging the custom code changes and pushes the newly created Docker images to Image Registry.
- In this architecture, Cloud or Infra team have full access to initiate the build pipeline configured on the Jenkins server. The Implementation team has no have access to this Jenkins server. However, a common artifacts repository is shared for both teams.
- As soon as any Docker image is pushed to the Azure Container Registry, Azure DevOps triggers the deployment to the Dev environment.
- UAT and Production deployments are approval based and they are called on-demand. To deploy the UAT environment, trigger the UAT deployment. Once the deployment is triggered, an approval mail is sent. After receiving the approval, the UAT deployment starts automatically.
- The production deployment is also approval based but it is multi-level approval, to deploy to a production environment the approval of all stakeholders is required, and most importantly once all the approvals from stakeholders are received, deployment to the production environment is not triggered automatically. A manual intervention mail is sent. To deploy to production with a checklist, all the checklist points get verified that they are covered or not. If not, then the deployment to the production gets rejected.

3 Implementation of Custom Code Deployment Pipeline

The custom code deployment pipeline is separated into two parts: **Build Pipeline** and **Release Pipeline**. Build Pipeline is configured on the Jenkins Server and Release Pipeline is configured on the AzureDevOps.

For configuration on Release Pipeline, refer to the *Configuration of Azure DevOps Release Pipeline for AKS* document.

3.1 Approach Guide for Build Pipeline

Perform the below steps to build pipeline:

1. There's a pre-defined folder structure for custom code. Only in that folder structure, the Implementation team pushes their custom code. However, all types of possible custom codes in OmniDocs are covered.
2. It covers 3 different types of custom codes in OmniDocs.
   For example,
   - OmniDocs_CustomCode
   - omniDocs_Hook
   - WebAPI_CustomCode
3. This folder structure is available in the artifacts repository. An artifacts repository can be like Azure FileShare, SVN, FTP, and so on. SVN is used in this implementation.

4. Only this folder structure defined in the artifacts repository is accessible from the Implementation team.
5. Jenkins Build Pipeline has the **5 jobs** that are as follows:
   - Pull the compiled custom code from the artifacts repository.
   - Pull the latest Docker Image from the container registry in which custom code needs to deploy.
   - Merge custom code changes.
   - Build a new Docker image.
   - Push the newly created Docker image to the container registry.
     For example,

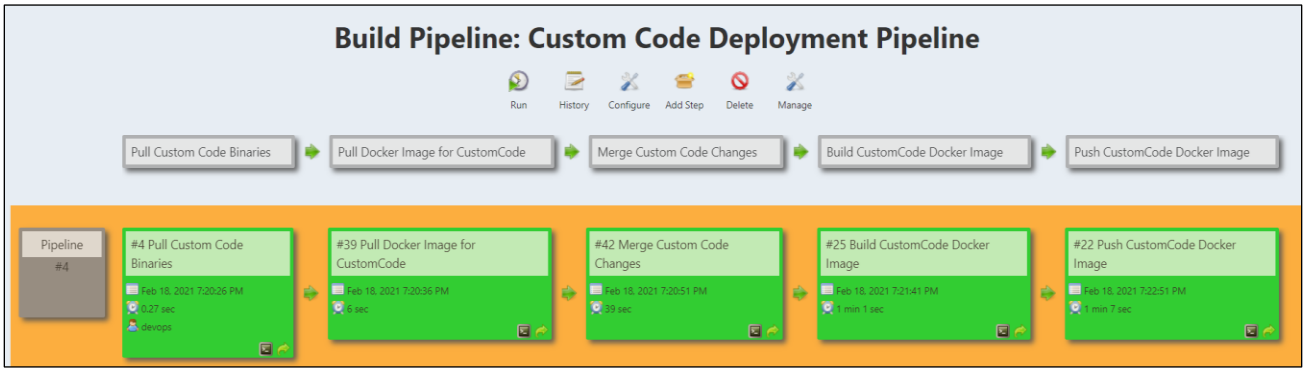

Figure 3.1

6. After pulling the latest custom code from the artifacts repository, Jenkins reads the *UserInput.properties* file.
7. This properties file contains all the user inputs that are required for condition-based custom code deployment.
8. This property file has multiple sections.
   - **#Container Registry Info**
   - This section contains the container registry information. Here provide **Azure Container Registry login server** and **Azure Container Registry username** where the container registry is created in. **Azure Container Registry password** is used as encrypted environment variables in Jenkins jobs.
     For example,

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#Container Registery Info
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
ContainerRegistryPath=newgencicdpiepline.azurecr.io
ContainerRegistryUser=newgencicdpiepline
```

Figure 3.2

   - **#Custom code changes to be deployed**

In this section, select the components to deploy the custom code. You can select or deselect the components by setting the component's value as Y or N respectively.
For example,

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#Custom code changes to be deployed
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
OmniDocs_CustomCode=Y
omniDocs_Hook=Y
WebAPI_CustomCode=Y
```

Figure 3.3

- **#Custom code can be deployed to the following Docker Images**
  This section just contains the information about the components and their destination Docker images. One component can be deployed to one or more Docker containers. So, you can decide in which container you want to deploy OmniDocs custom code changes.
  For example,

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#Custom code can be deployed to the following Docker Images
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#OmniDocs_CustomCode                OmniDocs_WEB
#omniDocs_Hook                      OmniDocs_EJB
#WebAPI_CustomCode                  OmniDocs_WEB
```

Figure 3.4

- **#Docker Image to be updated**
  In this section, select the Docker image(s) in which you want to deploy custom code changes.
  For example,

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#Docker Image to be updated
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
OmniDocs_WEB=Y
OmniDocs_EJB=Y
```

Figure 3.5

- **#Docker Image Info**

  This section contains the information about the source Docker images in which custom code changes can be merged or deployed.

  For example,

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#Docker Image Info
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

OmniDocs_WEB_ImageName=omnidocs11.0web
OmniDocs_WEB_Imagetag=base

OmniDocs_EJB_ImageName=omnidocs11.0ejb
OmniDocs_EJB_Imagetag=base
```

Figure 3.6

- **#New Docker Image Info with Custom Code changes**

  This section contains the information about new Docker images that gets created after merging the custom code changes.

  For example,

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#New Docker Image Info with Custom Code changes
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Custom_OmniDocs_WEB_ImageName=omnidocs11.0web
Custom_OmniDocs_WEB_Imagetag=custom

Custom_OmniDocs_EJB_ImageName=omnidocs11.0ejb
Custom_OmniDocs_EJB_Imagetag=custom
```

Figure 3.7

- **#Other user Inputs**

  This section contains other information that can be used in the Jenkins pipeline.

  For example,

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#Other user Inputs
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
JAVA_HOME=C:\Program Files\Java\jdk1.8.0_91
```

Figure 3.8

9. Based on the input provided in the *UserInput.properties* file, Jenkins pulls the Docker images, merges the custom code changes, builds the new Docker images, and pushes Docker images to the container registry.
10. In the case of few components, deploy .JAR file to the EJB components Docker containers like OmniDocs EJB container and this container is running on the underlying AppServer JBoss EAP. In the case of JBoss EAP, deploy any dependent library then must make an entry in the *module.xml* file. Therefore, this build pipeline also handles this case.
11. To handle the above *module.xml* case, the Implementation team provides a **module.txt** file that contains the name of the new JAR file to be deploy it as a dependent library. Rest Build pipeline manages the updates of the *module.xml* file inside the containers.
    For example:

```
<!--    Module.txt  -->
<resource-root path="NewJarFile.jar"/>
```

Figure 3.9

3.2 Configuration of Jenkins for Build Pipeline

This section describes the configuration of Jenkins for Build Pipeline.

3.2.1 Prerequisites

Following are the prerequisites:

- **Operating System**: Windows Server 2019 (Edition: Standard/Datacenter)
- **Java** 1.8 update 91 and above
- **Docker Engine** 20.10.10 or later version must be installed.
- **Azure CLI** 2.28.0 or a later version must be installed.
- **Cygwin** utility must be installed. [This utility is used to execute the Linux commands on Windows].
- **Jenkins** 2.235.0 or a later version must be installed with default plug-ins along with the following plug-ins.
  - Conditional Build Step
  - Credentials Binding
  - Subversion
  - Environment Injector

3.2.2 Configuration of Jenkins Jobs

Jenkins have the following jobs for the custom code deployment pipeline:

- Pull the compiled custom code from the artifacts repository.
- Pull the latest Docker Image from the container registry in which custom code needs to deploy.

- Merge custom code changes.
- Build a new Docker image.
- Push the newly created Docker image to the container registry.

Before creating any job, perform the following server-level configurations in the Jenkins.

1. Sign in to the **Jenkins Server**.

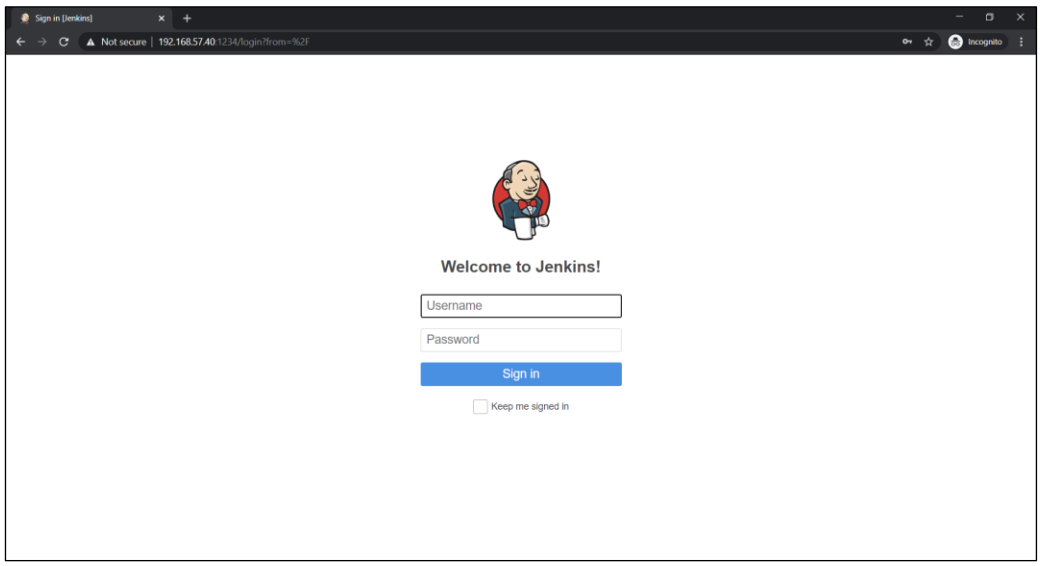

Figure 3.10

2. After the successful login, click **Manage Jenkins** link showing on the left panel.

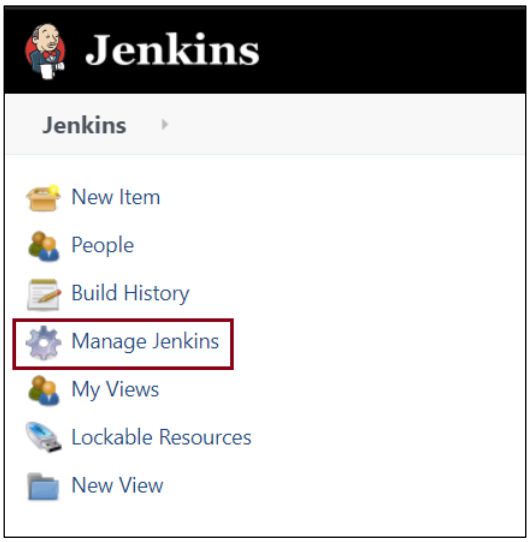

Figure 3.11

3. Click **Configure System** in the **System Configuration** section.

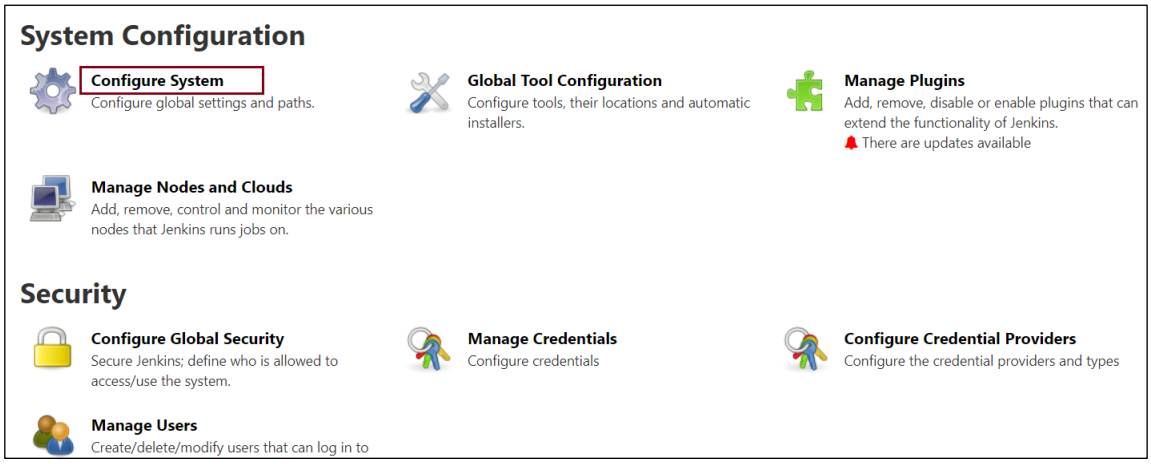

Figure 3.12

4. Under the Global properties, define an environment variable **PATH** with the following values separated with a semi-colon:
   - Docker installation path [C:\Program Files\Docker\Docker\resources\bin]
   - Cygwin installation path [C:\cygwin64\bin]
   - Azure CLI installation path [C:\Program Files (x86)\Microsoft SDKs\Azure\CLI2\wbin]
   - Windows System32 path [C:\Windows\System32]

   For example,

```
PATH= C:\Program Files\Docker\Docker\resources\bin;C:\cygwin64\bin;C:\Program Files (x86)\Microsoft SDKs\Azure\CLI2\wbin;C:\Windows\System32
```

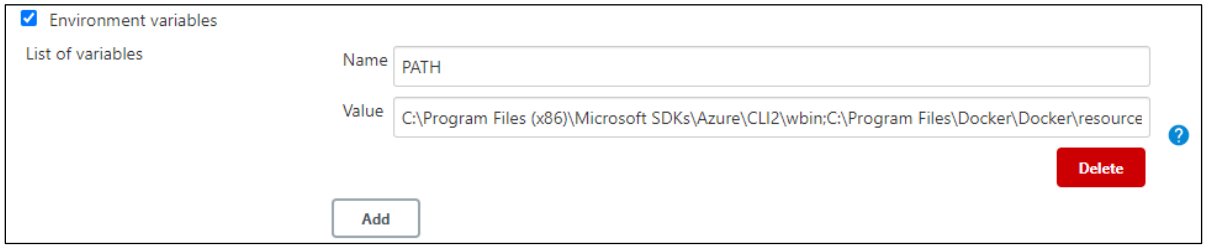

Figure 3.13

5. **Save** the changes.

3.2.2.1 Pull Custom Code Binaries

To pull custom code binaries, follow the below steps:

1. Click **New Item** link given on the left panel.

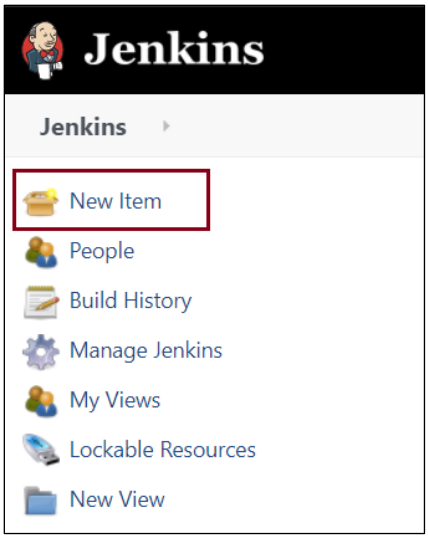

Figure 3.14

2. Specify the item name or job name and select the project type as **Freestyle project**.

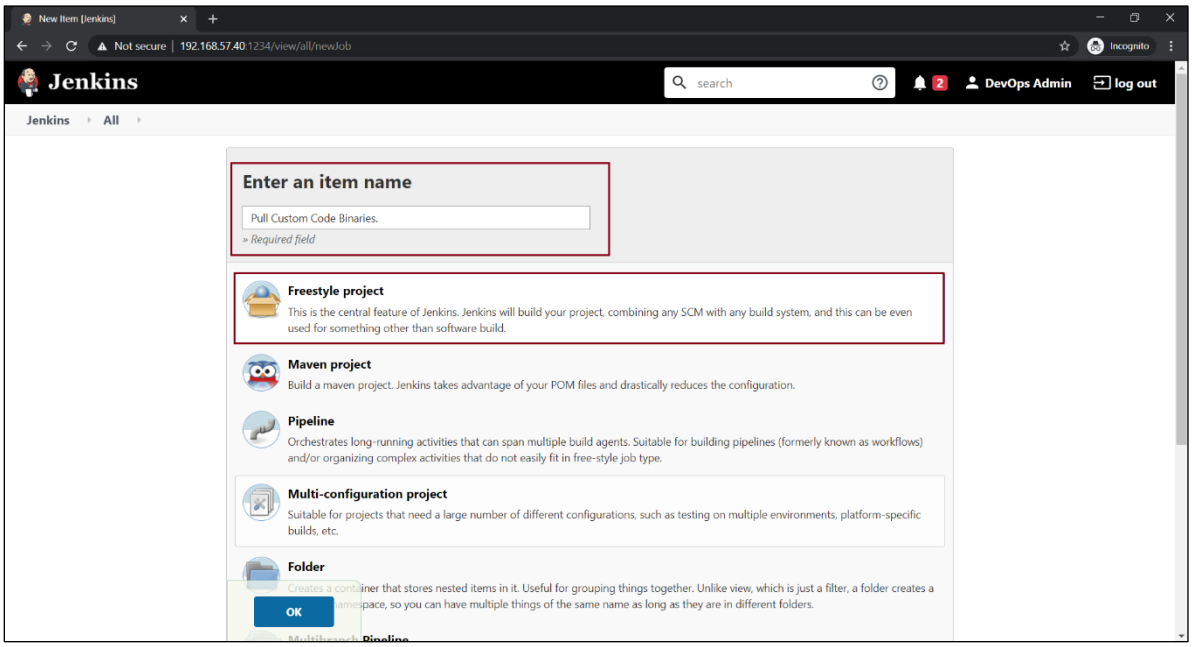

Figure 3.15

3. Specify the project description.
4. Under the **General**, click **Advanced** and specify the **Use custom workspace**.
   For example,

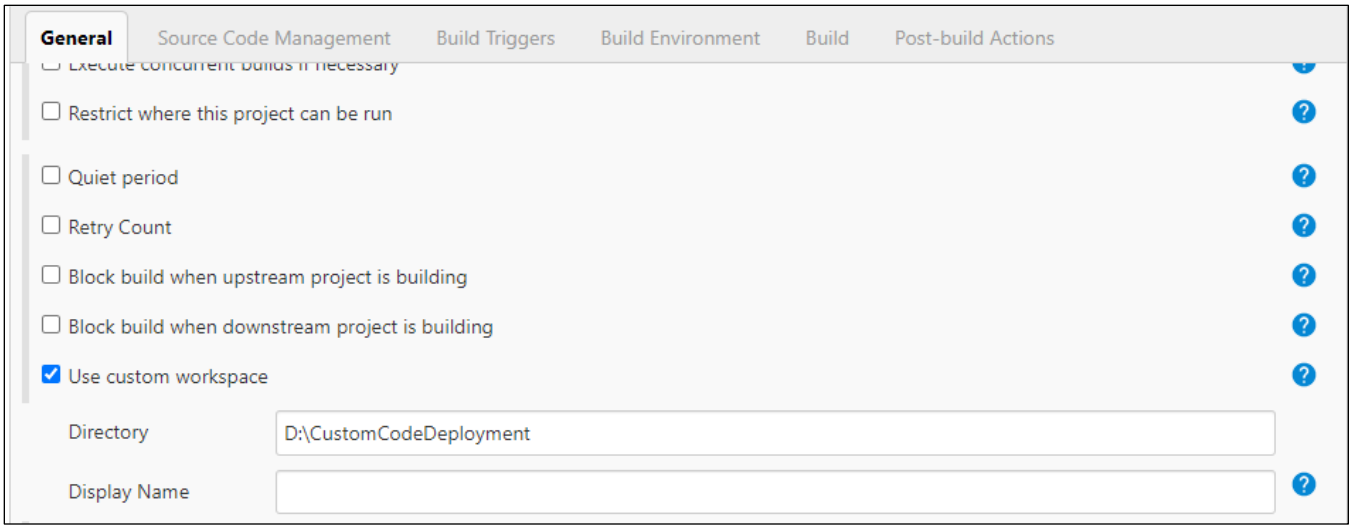

Figure 3.16

Custom workspace is the parent folder for your custom code deployment architecture.

5. Select the **Subversion** radio button under the Source Code Management.
6. Specify the **Repository URL** where your custom code folder structure resides.
7. Specify the SVN **credentials**.
8. Specify the **Local module repository** for checking out your custom code. As defined in step 4, local module repository gets appended to the custom workspace.
   For example,

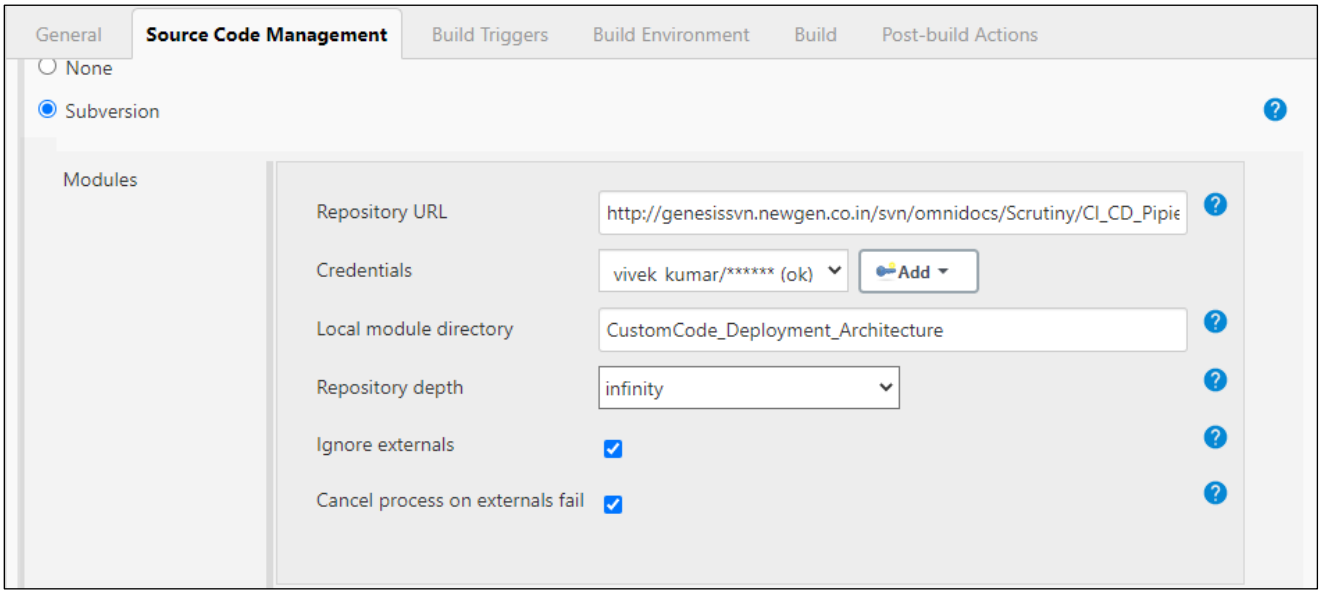

Figure 3.17

9. **Save** the changes.

3.2.2.2 Pull Docker Image for Custom Code

To pull Docker images for custom code, follow the below steps:

1. Click **New Item** link given on the left panel.
2. Specify the item name or job name and select the project type as **Freestyle project**.
3. You can specify the project description.
4. Select the checkbox **Inject passwords to the build as environment variables** given in the **Build Environment** section.
5. Specify 1 Job password: **ContainerRegistryPassword** and specify the Azure Container Registry password. For example,

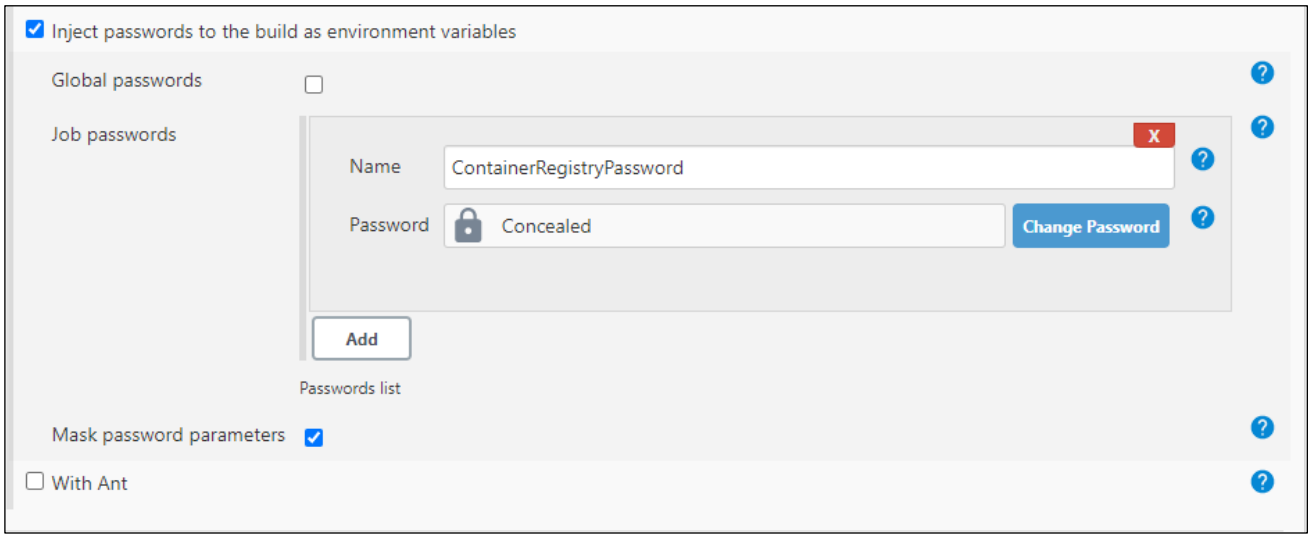

Figure 3.18

6. Add **Inject environment variables** as a build step task in the **Build** section.
7. Specify the **UserInput.properties** file path.
   For example,

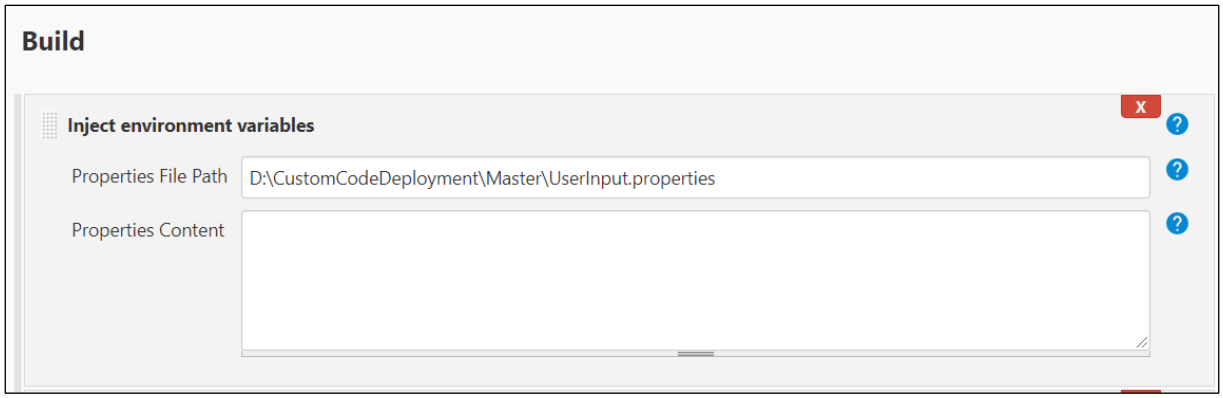

Figure 3.19

8. Add a **Conditional step (single)** as a build step task under the **Build** section.
9. Select **Execute Windows batch command** as **Run?** and **Builder** ('Run?' is a condition to decide whether a 'builder' command should run or not).
10. Specify the following command for the condition:

```
@echo off
findstr /I "OmniDocs_WEB=Y" D:\CustomCodeDeployment\Master\UserInput.properties
```

11. Specify the following commands for the builder:

```
@echo off
docker login %ContainerRegistryPath% -u %ContainerRegistryUser% -p
%ContainerRegistryPassword%
docker pull
%ContainerRegistryPath%/%OmniDocs_WEB_ImageName%:%OmniDocs_WEB_Imagetag%
```

For example,

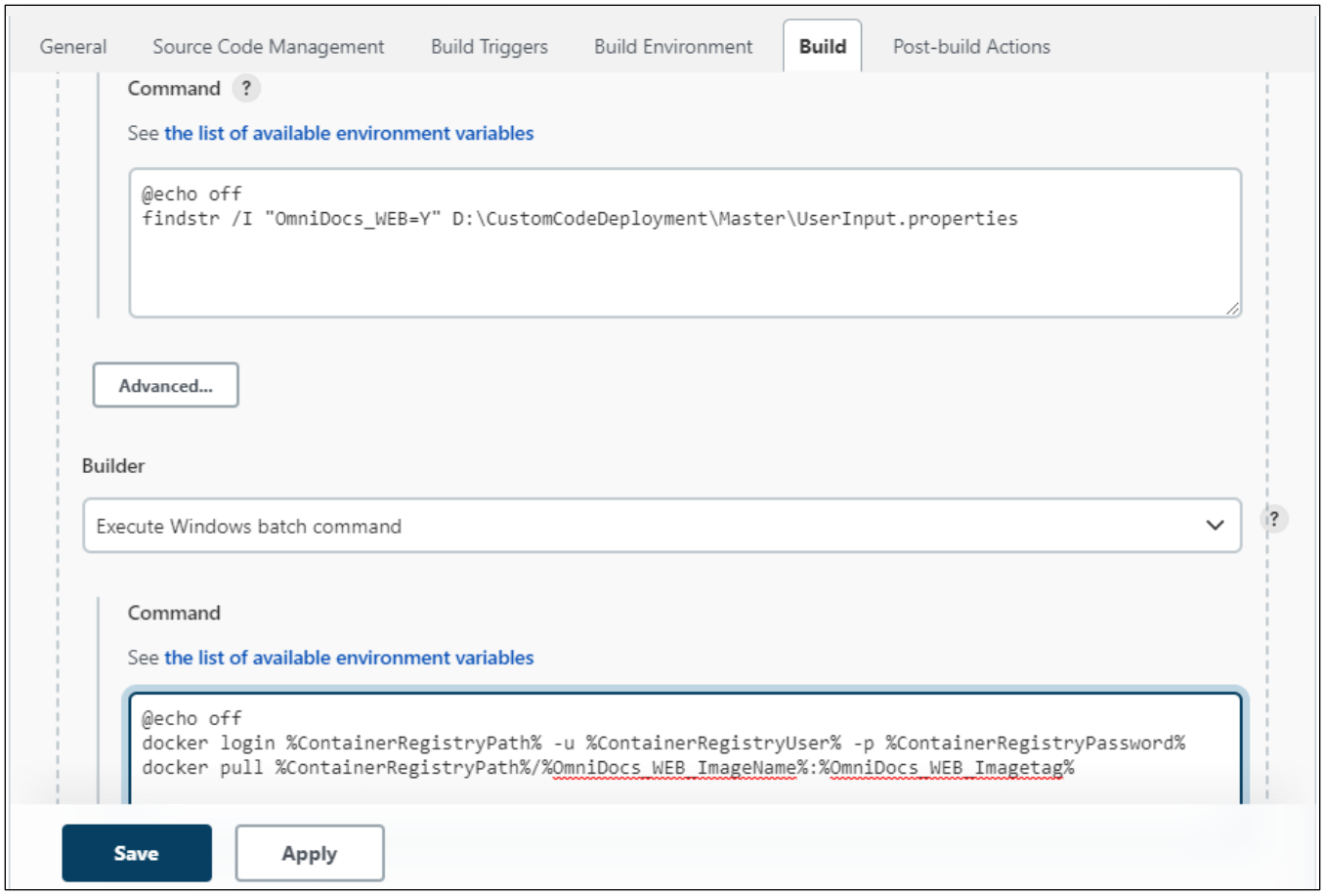

Figure 3.20

12. Click **Save** to save the changes.
    Here, the condition and builder for the **OmniDocs_WEB** Docker image is set.
    There is more 'Conditional step (single)' for other Docker images such as OmniDocs_EJB.

3.2.2.3 Merge Custom Code Changes

To merge custom code changes, follow the below steps:

1. Click **New Item** link given on the left panel.
2. Specify the item name or job name and select the project type as **Freestyle project**.
3. You can specify the project description.
4. Add **Inject environment variables** as a build step task under the **Build** section.
5. Specify the **UserInput.properties** file path.
   For example,

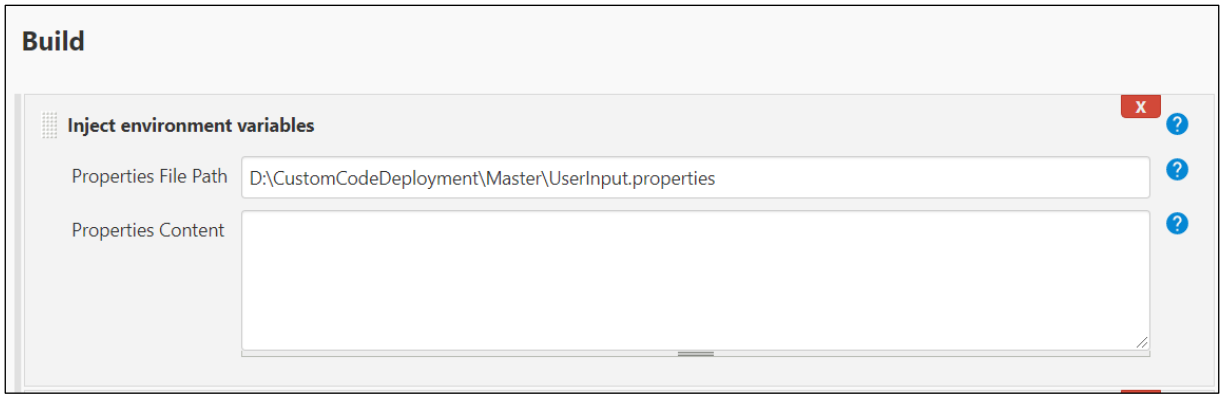

Figure 3.21

6. Add **Conditional step (multiple)** as a build step task under the **Build** section.
7. Select **Execute Windows Batch commands** as **Run?** (Run? is a condition to decide whether a builder command should run or not).
8. Click **Add step to condition** in the **Steps to run if the condition is met** section.
   For Example,

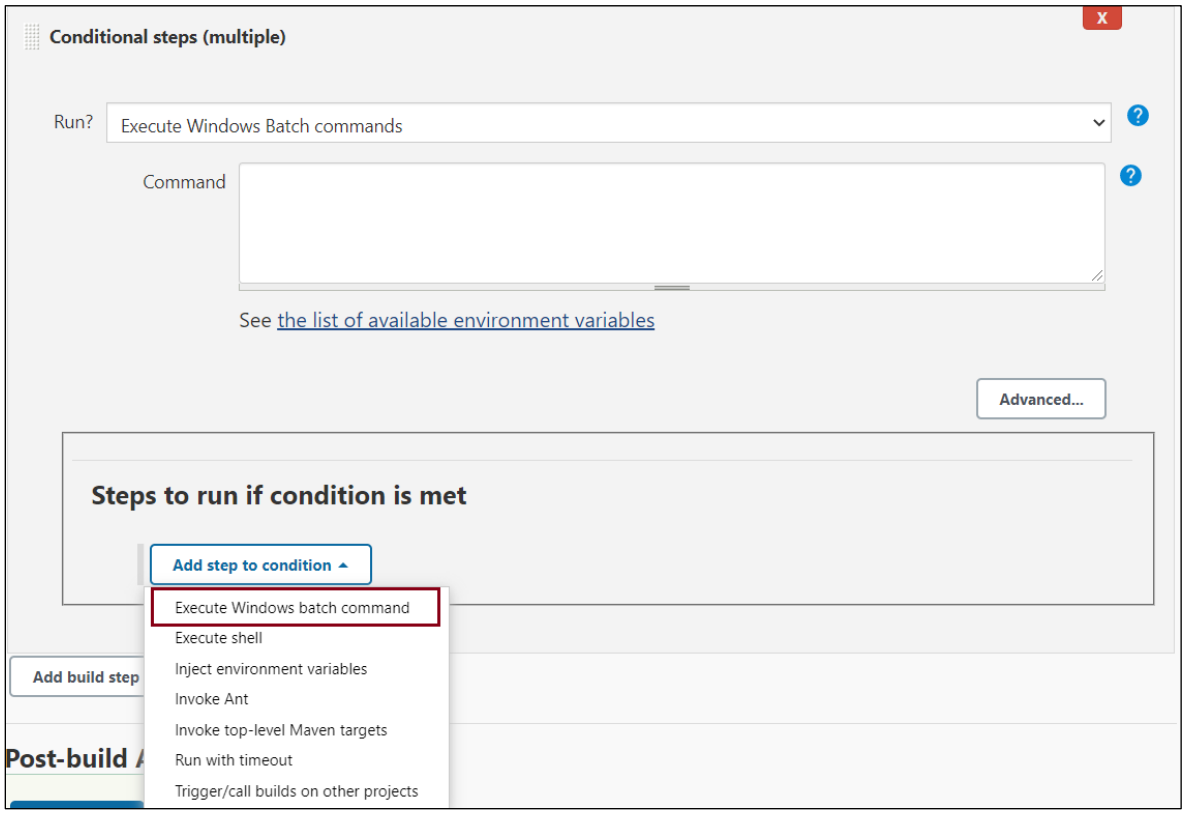

Figure 3.22

Merge OmniDocs_CustomCode custom code changes:

1. Specify the following command for the condition:

```
@echo off
findstr /I "OmniDocs_CustomCode=Y"
D:\CustomCodeDeployment\Master\UserInput.properties
```

2. Specify the following commands for the 1st builder:

```
@echo off
findstr /I "OmniDocs_WEB=Y" D:\CustomCodeDeployment\Master\UserInput.properties
if %ERRORLEVEL% equ 0 goto found
goto notfound
:found
set
srcCustomCode=D:\CustomCodeDeployment\CustomCode_Deployment_Architecture\OmniDoc
s_CustomCode
set
artifactsDir=D:\CustomCodeDeployment\Build_Docker_Images\OmniDocs_WEB\artifacts\
webapps
pushd %srcCustomCode%\webapps
xcopy *.war %artifactsDir%\ /I /Y
:notfound
exit /b 0
```

Merge OmniDocs_Hook custom code changes:

1. Specify the following command for the condition:

```
@echo off
findstr /I "omniDocs_Hook=Y" D:\CustomCodeDeployment\Master\UserInput.properties
```

2. Specify the following commands for the 1st builder:

```
@echo off
findstr /I "OmniDocs_EJB=Y" D:\CustomCodeDeployment\Master\UserInput.properties
if %ERRORLEVEL% equ 0 goto found
goto notfound
:found
for /f %%i in ('docker create %OmniDocs_EJB_ImageName%:%OmniDocs_EJB_Imagetag%')
do set RESULT=%%i
set srcFile1=/Newgen/jboss-eap-
7.4/modules/omnidocs_library/main/omnidocs_hook.jar
set srcFile2=/Newgen/jboss-eap-7.4/modules/omnidocs_library/main/module.xml
set destDir1=D:\CustomCodeDeployment\TempDir\omnidocs_hook\OmniDocs_EJB
set
destDir2=D:\CustomCodeDeployment\CustomCode_Deployment_Architecture\omniDocs_Hook
md %destDir1%
md %destDir2%
docker cp %RESULT%:%srcFile1% %destDir1%
docker cp %RESULT%:%srcFile2% %destDir2%
docker rm -f %RESULT%
```

```
set
srcCustomCode=D:\CustomCodeDeployment\CustomCode_Deployment_Architecture\omniDoc
s_Hook
set
artifactsDir=D:\CustomCodeDeployment\Build_Docker_Images\OmniDocs_EJB\artifacts\
modules\omnidocs_library\main\
set war=omnidocs_hook.jar
pushd %srcCustomCode%\omnidocs_hook.jar
"%JAVA_HOME%\bin\jar.exe" -uvf %destDir1%\%war% *
xcopy %destDir1%\%war% %artifactsDir% /I /Y

pushd  %srcCustomCode%
xcopy *.jar %artifactsDir% /I /Y

pushd D:\CustomCodeDeployment\Master
if exist "%srcCustomCode%\module.txt" (
"%JAVA_HOME%\bin\java.exe" -jar AppendModuleXML.jar %srcCustomCode%\module.xml
%srcCustomCode%\module.txt
"%JAVA_HOME%\bin\java.exe" -jar UpdateModuleXML.jar %srcCustomCode%\module.xml
xcopy %srcCustomCode%\module.xml %artifactsDir% /I /Y
)

:notfound
exit /b 0
```

Merge WebAPI_CustomCode custom code changes:

1. Specify the following command for the condition:

```
@echo off
findstr /I "WebAPI_CustomCode=Y"
D:\CustomCodeDeployment\Master\UserInput.properties
```

2. Specify the following commands for the 1st builder:

```
@echo off
findstr /I "OmniDocs_WEB=Y" D:\CustomCodeDeployment\Master\UserInput.properties
if %ERRORLEVEL% equ 0 goto found
goto notfound
:found
for /f %%i in ('docker create %OmniDocs_WEB_ImageName%:%OmniDocs_WEB_Imagetag%')
do set RESULT=%%i
set srcFile=/Newgen/jws-6.0/tomcat/webapps/omnidocs.war
set destDir=D:\CustomCodeDeployment\TempDir\WebAPI_CustomCode\OmniDocs_WEB
md %destDir%
docker cp %RESULT%:%srcFile% %destDir%
docker rm -f %RESULT%
set
srcCustomCode=D:\CustomCodeDeployment\CustomCode_Deployment_Architecture\WebAPI_
CustomCode
```

```
set
artifactsDir=D:\CustomCodeDeployment\Build_Docker_Images\OmniDocs_WEB\artifacts\
webapps\
set war=omnidocs.war

pushd %destDir%
"%JAVA_HOME%\bin\jar.exe" -xvf %destDir%\%war% WEB-INF\lib\webapi.jar

pushd %srcCustomCode%\webapi.jar
"%JAVA_HOME%\bin\jar.exe" -uvf %destDir%\WEB-INF\lib\webapi.jar *

pushd %destDir%
"%JAVA_HOME%\bin\jar.exe" -uvf %destDir%\%war% WEB-INF\lib\webapi.jar

xcopy %destDir%\%war% %artifactsDir% /I /Y

:notfound
exit /b 0
```

3.2.2.4 Build Custom Code Docker Image

To build the custom code docker image, follow the below steps:

1. Click **New Item** link given on the left panel.
2. Specify the item name or job name and select the project type as **Freestyle project**.
3. You can specify the project description.
4. Add **Inject environment variables** as a build step task in the **Build** section.
5. Specify the **UserInput.properties** file path.
   For example,

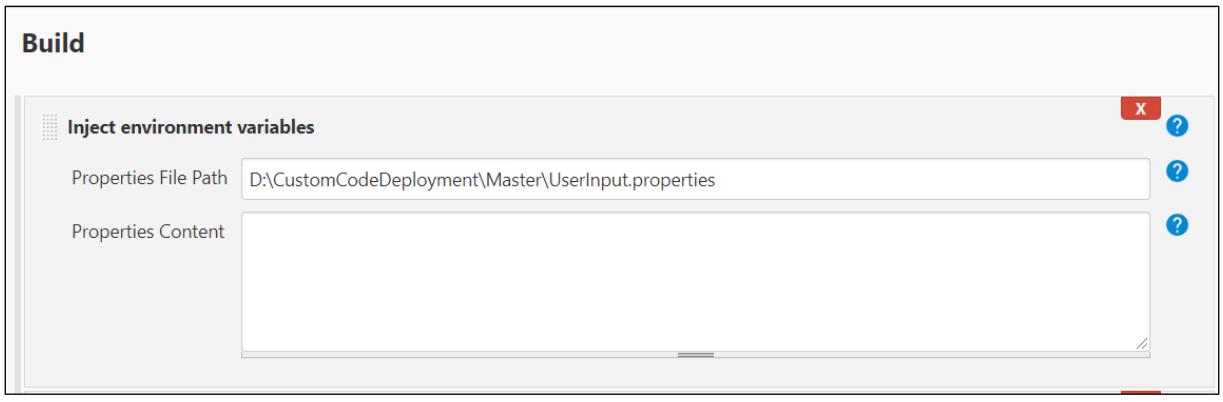

Figure 3.23

6. Add **Conditional step (single)** as a build step task under the **Build** section.
7. Select **Execute Windows batch command** as **Run?** and **Builder** (Run? is a condition to decide whether a builder command should run or not).

8. Specify the following command for the condition:

```
@echo off
findstr /I "OmniDocs_WEB=Y" D:\CustomCodeDeployment\Master\UserInput.properties
```

9. Specify the following commands for the builder:

```
@echo off
set ImageFilePath="D:\CustomCodeDeployment\Build_Docker_Images\OmniDocs_WEB"
set ImageName=%Custom_OmniDocs_WEB_ImageName%
set ImageTag=%Custom_OmniDocs_WEB_Imagetag%

pushd %ImageFilePath%
copy /y Dockerfile_Original Dockerfile

if exist Dockerfile (
   sed -i s+ContainerRegistryPath+%ContainerRegistryPath%+g Dockerfile
   sed -i s+IMAGE_NAME+%OmniDocs_WEB_ImageName%+g Dockerfile
   sed -i s+IMAGE_TAG+%OmniDocs_WEB_Imagetag%+g Dockerfile
) else (
    echo File doesn't exist
)

pushd %ImageFilePath%
docker build . -t %ImageName%:%ImageTag%
```

For example,

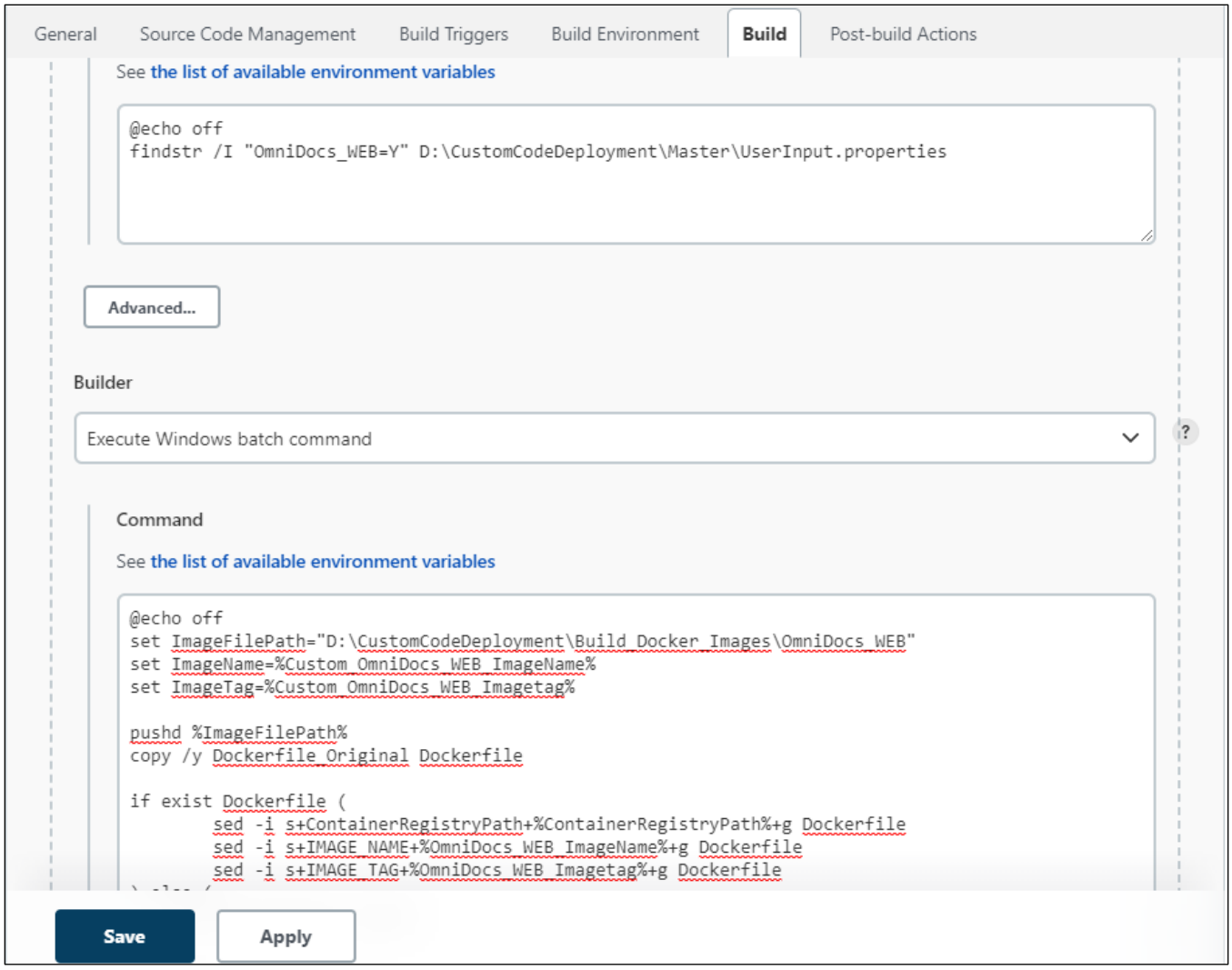

Figure 3.24

10. Click **Save** to Save the changes.
    Here, the condition and builder for the **OmniDocs_WEB** Docker image is set.
    There are more Conditional steps (single) for other Docker images such as OmniDocs_EJB.

3.2.2.5 Push Custom Code Docker Image

To push custom code Docker image, follow the below steps:

1. Click **New Item** link showing on the left panel.
2. Specify the item name or job name and select the project type as **Freestyle project**.
3. Specify the project description.
4. Select the checkbox **Inject passwords to the build as environment variables** given in the **Build Environment** section.

5. Specify 1 Job password: **ContainerRegistryPassword** and specify the Azure Container Registry password.
   For example,

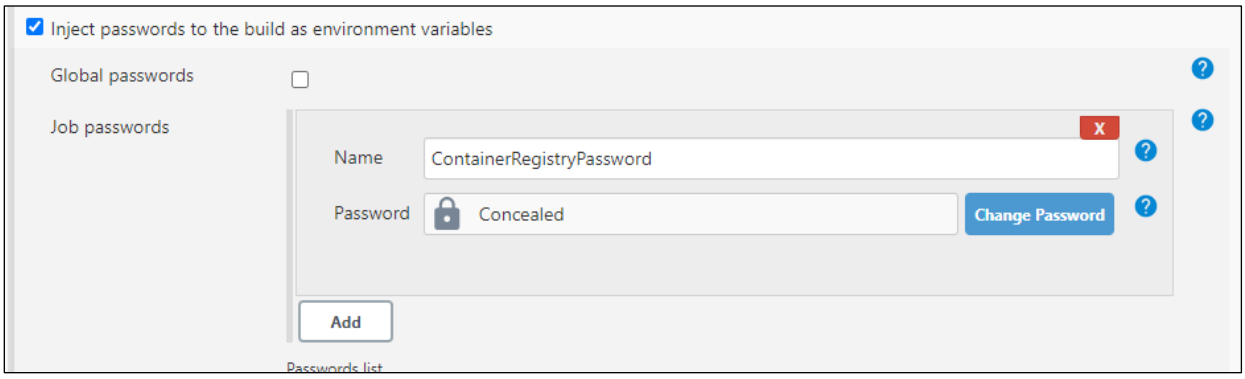

Figure 3.25

6. Add **Inject environment variables** as a build step task in the **Build** section.
7. Specify the **UserInput.properties** file path.
   For example,

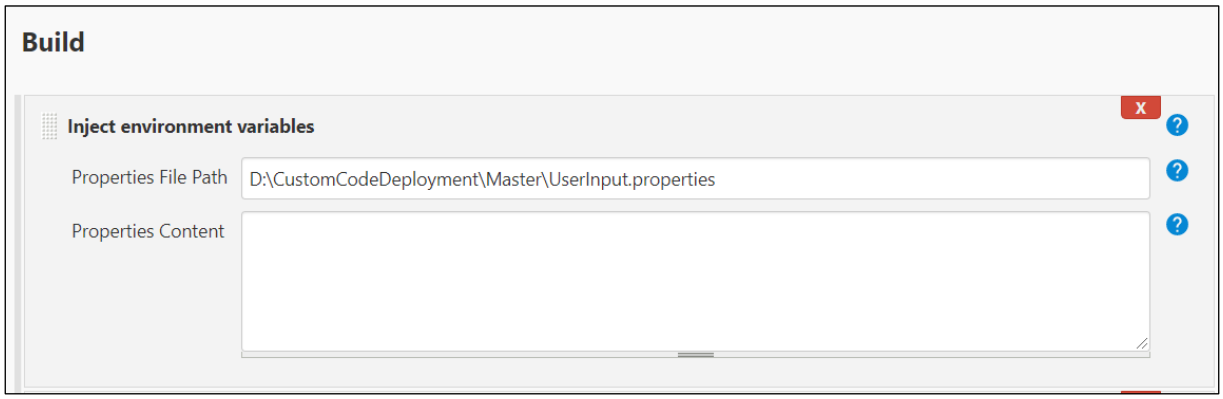

Figure 3.26

8. Add **Conditional step (single)** as a build step task under the **Build** section.
9. Select **Execute Windows batch command** as **Run?** and **Builder** (Run? is a condition to decide whether a builder command should run or not).
10. Specify the following command for the condition:

```
@echo off
findstr /I "OmniDocs_WEB=Y" D:\CustomCodeDeployment\Master\UserInput.properties
```

11. Specify the following command for the builder:

```
@echo off
set ContainerRegistryPath=%ContainerRegistryPath%
```

```
set ContainerRegistryUser=%ContainerRegistryUser%
set ContainerRegistryPassword=%ContainerRegistryPassword%
set ImageName=%Custom_OmniDocs_WEB_ImageName%
set ImageTag=%Custom_OmniDocs_WEB_Imagetag%
set BuildNumber=%ImageTag%-build-%BUILD_NUMBER%

docker login %ContainerRegistryPath% -u %ContainerRegistryUser% -p
%ContainerRegistryPassword%

docker tag %ImageName%:%ImageTag% %ContainerRegistryPath%/%ImageName%:%ImageTag%
docker push %ContainerRegistryPath%/%ImageName%:%ImageTag%

docker tag %ContainerRegistryPath%/%ImageName%:%ImageTag%
%ContainerRegistryPath%/%ImageName%:%BuildNumber%
docker push %ContainerRegistryPath%/%ImageName%:%BuildNumber%
```

For example,

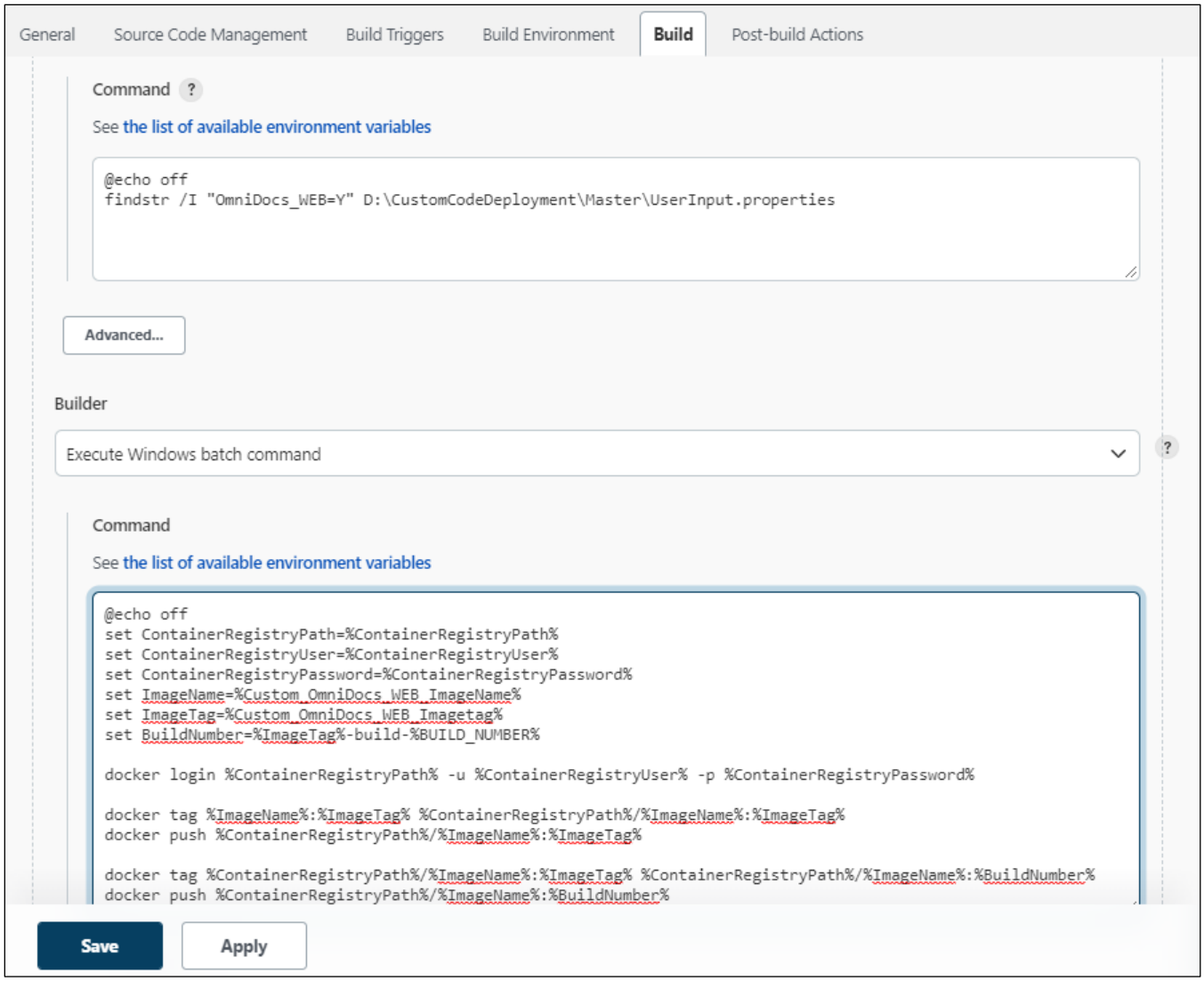

Figure 3.27

12. Click **Save** to save the changes.

    Here, the condition and builder for the **OmniDocs_WEB** Docker image is set.

    There are more Conditional steps (single) for other Docker images such as OmniDocs_EJB.