webMethods Automated Build and Deployment Guide with Jenkins

Author: Srikanth Prathipati
Version 0.1 | December 2015
# Table of contents

1.0 Introduction 3

1.1 Overview 3
1.2 Audience 3

2.0 Glossary 3

3.0 Assumptions/Prerequisites 4

4.0 Components Involved 4

4.1 VCS 4
4.2 Designer Workstation 4
4.3 Continuous Integration Tool 5
4.4 Asset build Environment 5
4.5 Deployer 5

5.0 Process 6

5.1 Asset Development 7
5.2 Asset Deployment 8

6.0 Asset Catalogue 9

7.0 VCS Structure 10

7.1 Recommended Structure for SVN 10
7.2 Recommended Structure for TFS 10
7.3 Recommended Structure for GIT 12

8.0 Build Server - Setup & Configuration 12

8.1 Directory Structure 12
8.2 Permission Matrix 13

9.0 CI Tool - Setup & Configuration 14

9.1 Installation & Port Configurations 14
9.2 Plug-in Installation 14
9.3 Configure Jenkins 16

10.0 ABE - Setup & Configuration 18

11.0 Build & Deployment Workflow 22

11.1 Check out with Asset Build Environment Vs Check out with CI Tool 22
11.2 Checkout using CI Tool 23
11.3 Build & Deployment 24

12.0 Change History 26
1.0 Introduction

Continuous Integration (CI) is a software engineering practice in which changes are immediately tested and reported when they are added to a larger code base. Their goal is to provide rapid feedback so that if a defect is introduced into the code base, it can be identified and corrected as soon as possible. - techtarget.com

CI is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, deployment and tests allowing teams to detect problems early.

Why CI?

- CI helps to reduce overheads across the development & deployment process
- Deployments become much less error-prone and much more repeatable
- Anyone in the team can deploy
- Developers can continue their time developing code
- Deploying to somewhere new is not a headache
- Releases can be more frequent

1.1 Overview

The sections in this document contain instructions for automating build and deployment of ESB, MWS (CAF, Tasks, skins, shells, users, roles, etc.), BPM, UM, TN assets using Jenkins or any other continuous integration tool as part of a continuous integration (CI) process. This document only covers about Repository based deployment but not run time deployment from server.

1.2 Audience

This document is intended for Development Team, Deployment Team, Infrastructure team who shall actively participate in the build-release-deploy activities and CI Process.

2.0 Glossary

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CI</td>
<td>Continuous Integration</td>
</tr>
<tr>
<td>IS</td>
<td>Integration Server</td>
</tr>
<tr>
<td>ESB</td>
<td>Enterprise Service Bus</td>
</tr>
<tr>
<td>UM</td>
<td>Universal Messaging</td>
</tr>
<tr>
<td>MWS</td>
<td>My webMethods Server</td>
</tr>
</tbody>
</table>
### 3.0 Assumptions/Prerequisites

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCS is the source for the build creation.</td>
</tr>
<tr>
<td>2</td>
<td>All the dependent assets need to be checked into VCS (e.g. IS config folder assets for users, global variables etc.)</td>
</tr>
<tr>
<td>3</td>
<td>Deployer and ABE need to be installed in the same environment (often called build server) where the entire framework is configured</td>
</tr>
<tr>
<td>4</td>
<td>CI Tool (e.g. Jenkins/Maven) should have proper access permissions to required assets (Matrix given in the following sections)</td>
</tr>
</tbody>
</table>

### 4.0 Components Involved

#### 4.1 VCS
Most widely used SCM tools are SVN, TFS and GIT. GIT SCM has some limitations with webMethods.

#### 4.2 Designer Workstation
Designer workstation is webMethods component installed on developers desktop. This is an eclipse plugin useful for developing ESB, BPM and CAF assets. This also integrates with version...
control systems. A local Integration Server on user’s desktop to be installed to perform the development.

### 4.3 Continuous Integration Tool

**Continuous Integration - Tools**

There are several tools available to incorporate Continuous Integration into the build.

- Supported by Customer as part of their SDLC Tools
- Part of the Atlassian Software Development and Collaboration Suite
- Highly customisable tasks and provides a well-documented API

In this document, Jenkins is considered as CI Tool. Jenkins is an open source continuous integration tool written in Java. Jenkins provides continuous integration services for software development. It is a server-based system running in a servlet container such as Apache Tomcat. It supports SVN and can execute Apache Ant and Apache Maven based projects as well as arbitrary shell scripts and Windows batch commands.

### 4.4 Asset build Environment

ABE comes with a set of ANT scripts that are used in building the composites in the repository based approach. This is packaged along with the product suite.

### 4.5 Deployer

webMethods Deployer is a tool you use to deploy user-created assets that reside on source webMethods runtimes or repositories to target webMethods runtime components (runtimes).

Deployer supports two scenarios for deploying assets:
In a runtime-based deployment scenario, Deployer deploys assets from webMethods runtimes to which Deployer is connected.

In a repository-based deployment scenario, Deployer deploys assets built from sources in a VCS and stored on a repository.

### 5.0 Process

<table>
<thead>
<tr>
<th>STEP</th>
<th>DESCRIPTION</th>
<th>Responsible component</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Developer commits the assets to VCS</td>
<td>Designer Workstation / WmVCS / Any VCS Client</td>
</tr>
<tr>
<td>STEP</td>
<td>DESCRIPTION</td>
<td>Responsible component</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>2</td>
<td>CI Tool (in this document Jenkins) will checkout the assets from VCS to file system of the build server</td>
<td>CI Tool (Jenkins)</td>
</tr>
<tr>
<td>3</td>
<td>CI Tool will invoke the ABE ANT targets to parse the checked out assets and create the builds for the same. Builds will be saved on the file system of the build server</td>
<td>CI Tool (Jenkins) &amp; ABE</td>
</tr>
<tr>
<td>4</td>
<td>CI Tool will invoke the Deployer ANT target – Project automator (custom scripts) to create deployer project.</td>
<td>CI Tool (Jenkins) &amp; Deployer Command line Utility</td>
</tr>
<tr>
<td>5</td>
<td>CI Tool will invoke the Deployer ANT target (custom scripts) to perform variable substitution or deploy to the Target servers</td>
<td>CI Tool (Jenkins) &amp; Deployer Command line Utility</td>
</tr>
<tr>
<td>6</td>
<td>As part of CI process automated test can be invoked after successful deployment and deployments to higher environments can be followed</td>
<td>CI Tool</td>
</tr>
</tbody>
</table>

### 5.1 Asset Development

- Developers create the assets on the development environment and export or copy them to a file system or version control system. This is typical development model with webMethods. Below diagram shows the process.

**Typical Development model**

- Local Service Development model encourages offline development and facilitates the way to connect with different VCS systems and different versions of the code base.
5.2 Asset Deployment

Any CI Tool can be used to check out the code base from VCS. Asset Build Environment provides ANT scripts to create automated builds and using Deployer commands deployment can be automated.
6.0 Asset Catalogue

It is important to check-in the dependent assets to VCS for creating the proper build. For example, you need to check-in IS config entries for deploying ACLs, web service alias, global variables etc. Please refer to the attached document for identifying this.
7.0 VCS Structure

It is important to follow proper directory structure for checking the assets into VCS. In the SOA platform approach, different types of artefacts are stored in separate directories to ensure a plain organization of the source code structure.

7.1 Recommended Structure for SVN

SVN is most widely used SCM and is developer friendly. It is recommended to follow the below structure for this.

<table>
<thead>
<tr>
<th>Trunk</th>
<th>Folder to maintain the assets related current development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tags</td>
<td>Folder to maintain different releases of higher environments - SIT, UAT, Pre-Prod</td>
</tr>
<tr>
<td>Branches</td>
<td>Folder to maintain the stable releases - Product Release1, Release 2 etc.</td>
</tr>
</tbody>
</table>

7.2 Recommended Structure for TFS

TFS is another SCM from Microsoft which has below additional features compared to SVN.
Work item tracking, for keeping track of such things as defects, requirements, tasks, and scenarios.

Project management functions, which allow the shaping of a team project based on a user-specifiable software process, and which enable planning and tracking using Microsoft Excel and Microsoft Project.

Team build, for enabling a common process for building executable products.

Data collection and reporting, which aid in the assessment of a team project’s state, based on information gleaned from Team Foundation Server tools.

The Team Project Portal, which provides a central point of communication for a team project packaged as a Microsoft Windows SharePoint Services site.

Team Foundation Shared Services, which provide a number of common infrastructure services that invisible to end users but that are important to toolsmiths and extenders.

You can only create branches with TFS. Below is the recommended structure for TFS.

<table>
<thead>
<tr>
<th>Codebase</th>
<th>Folder to maintain the assets related current development</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntermediateReleases</td>
<td>Folder to maintain different releases of higher environments - SIT, UAT, Pre-Prod</td>
</tr>
<tr>
<td>StableReleases</td>
<td>Folder to maintain the stable releases - Product Release1, Release 2 etc.</td>
</tr>
</tbody>
</table>
7.3 Recommended Structure for GIT

Git repository is a web-based hosting service, which offers all of the distributed version control and source code management (SCM) functionality.

This is completely different from SVN and GIT and works on local repository concept. You will always have two repositories one is local repository and another one is Remote repository.

It is not possible to pull individual assets from repository either local or remote to your workspace. The only possibility is to make the workspace as your local repository. E.g. In case of local development, for working with IS packages, it is mandatory to make the packages folder as your local repository.

Below is the recommended structure for GIT repository.

<table>
<thead>
<tr>
<th>Branch</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB</td>
<td>Branch to maintain all IS assets</td>
</tr>
<tr>
<td>MWS</td>
<td>Branch to maintain all MWS assets</td>
</tr>
<tr>
<td>BPM</td>
<td>Branch to maintain all BPM assets</td>
</tr>
<tr>
<td>TN</td>
<td>Branch to maintain all TN assets</td>
</tr>
<tr>
<td>UM</td>
<td>Branch to maintain all UM assets</td>
</tr>
</tbody>
</table>

You can create copy of branches whenever stable or intermediate releases are required.

8.0 Build Server - Setup & Configuration

8.1 Directory Structure
Similar to VCS structuring, it is better to follow the organized directory structure to support the entire process. Please below directory structure on the build server for the same.
8.2 Permission Matrix

It is important to have proper access to CI tool for configuring this framework. webMethods process (say running with wmuser) and CI Tool (Jenkins) process (say running with Jenkins user) should have the below level of read or write access.

<table>
<thead>
<tr>
<th>Users Vs Folder/Permissions</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build source Dir (Created by CI Tool)</td>
<td>Jenkins</td>
<td>jenkins</td>
</tr>
<tr>
<td>Build Output folder (created by ABE)</td>
<td>jenkins &amp; wmuser</td>
<td>jenkins</td>
</tr>
<tr>
<td>wmDeployer/bin</td>
<td>Jenkins</td>
<td></td>
</tr>
</tbody>
</table>
9.0 CI Tool - Setup & Configuration

Configure your CI Tool to use ANT runtime that comes as part of ABE installation and to use Java runtime that comes as part of webMethods installation. Follow the below steps if you are using Jenkins as your CI Tool.

9.1 Installation & Port Configurations

Download Jenkins native package from below URL.

http://jenkins-ci.org/

Install Jenkins by selecting the default options. By default Jenkins runs on the port 8080 and if you wish to change this to different port, follow the below steps.

Open Jenkins.xml located under <Jenkins Installation Dir> and change the below tag with a different port number.

<arguments>-Xrs -Xmx256m -Dhudson.lifecycle=hudson.lifecycle.WindowsServiceLifecycle -jar "%BASE%/jenkins.war" --httpPort=8080</arguments>

Save and close the file and restart the service for applying the saved changes.

9.2 Plug-in Installation

Open Jenkins in the browser (by default http://<HostName>:8080/) and navigate to Manage Jenkins.
Click on Manage Plugins to install the plugin.

**Manage Jenkins**

- **Configure System**
  Configure global settings and paths.

- **Configure Global Security**
  Secure Jenkins, define who is allowed to access/use the system.

- **Reload Configuration from Disk**
  Discard all the loaded data in memory and reload everything from file system.

- **Manage Plugins**
  Add, remove, disable or enable plugins that can extend the functionality.

- **System Information**
  Displays various environmental information to assist trouble-shooting.

**TFS:** Search with the Filter Team Foundation to search the available plugins if your VCS is TFS.
SVN: In case of subversion, select the below plug-in.

<table>
<thead>
<tr>
<th>Plugin</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subversion Plugin</td>
<td>1.53</td>
</tr>
</tbody>
</table>

GIT: In case of GIT, select the below plug-ins.

<table>
<thead>
<tr>
<th>Plugin</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Git client</td>
<td>1.19.0</td>
</tr>
<tr>
<td>Git plugin</td>
<td>2.2.16</td>
</tr>
</tbody>
</table>

Install the required plugin and wait for the restart.

9.3 Configure Jenkins
Click on Manage Jenkins from left menu and select Configure System to save ANT and Java Runtime environments.
Navigate to JDK and ANT sections, click on add JDK and add ANT buttons. Provide the location of Java home (under webMethods Installation folder) and ANT home (under webMethods Installation folder) and uncheck the install automatically button and finally click on save to apply the changes.

Navigate to Team Foundation Server Section and provide the TFS Command line client details, in case of TFS.
10.0 ABE - Setup & Configuration

Asset build environment comes up with set of built in properties and built in Ant targets. For complete automation of build creation and deployment we need to add custom properties and custom ANT targets. Following steps will describe about the same.

Navigate to <SAG_installation_Dir>/common/AssetBuildEnvironment/master_build and open build.properties file. This contains the below properties. Please fill the suitable values for the same.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sag.install.dir</td>
<td>SoftwareAG installation directory</td>
</tr>
<tr>
<td>build.output.dir</td>
<td>tells where to create the builds</td>
</tr>
<tr>
<td>build.source.dir</td>
<td>location of the checkout code base</td>
</tr>
<tr>
<td>build.source.project.dir</td>
<td>optional used with the combination of is.acdl.config.dir</td>
</tr>
<tr>
<td>build.version</td>
<td>wM version Ex: 9.x or 8.2.x</td>
</tr>
<tr>
<td>enable.checkout</td>
<td>make this as false as Jenkins checks out the codebase</td>
</tr>
<tr>
<td>enable.archive</td>
<td>optional make it true if you want to keep the old builds</td>
</tr>
<tr>
<td>enable.build.IS</td>
<td>creates builds for IS assets if it is true</td>
</tr>
<tr>
<td>enable.build.MWS</td>
<td>creates builds for MWS assets if it is true</td>
</tr>
<tr>
<td>enable.build.BPM</td>
<td>creates builds for BPM assets if it is true</td>
</tr>
<tr>
<td>enable.build.TN</td>
<td>creates builds for TN assets if it is true</td>
</tr>
<tr>
<td>enable.build.Optimize</td>
<td>creates builds for Optimize assets if it is true</td>
</tr>
<tr>
<td>enable.build.Broker</td>
<td>creates builds for Broker assets if it is true</td>
</tr>
<tr>
<td>enable.build.EDA</td>
<td>creates builds for EDA assets if it is true</td>
</tr>
<tr>
<td>enable.build.RULES</td>
<td>creates builds for WmRules if it is true</td>
</tr>
<tr>
<td>enable.build.UniversalMessaging</td>
<td>creates builds for UM assets if it is true</td>
</tr>
<tr>
<td>build.log.enable</td>
<td>creates build logs if it is true</td>
</tr>
<tr>
<td>build.log.fileName</td>
<td>file name to store the logs</td>
</tr>
<tr>
<td>build.logLevel</td>
<td>Logging level</td>
</tr>
<tr>
<td>build.exit.error</td>
<td>Exits if the build creation fails if this is marked as false</td>
</tr>
<tr>
<td>is.acdl.config.dir</td>
<td>fill it if the config directory is same of development environment</td>
</tr>
<tr>
<td>bpm.acdl.model.ids</td>
<td>BPM model ids</td>
</tr>
<tr>
<td>bpm.acdl.bam.model.ids</td>
<td>BAM model ids</td>
</tr>
<tr>
<td>bpm.acdl.model.version</td>
<td>BAM model version</td>
</tr>
</tbody>
</table>
Also add the below custom properties to the same file for deploying the assets to the reference server.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>env</td>
<td>Variable to indicate the environment details</td>
</tr>
<tr>
<td>deployer.home</td>
<td>Deployer directory</td>
</tr>
<tr>
<td>env.dir</td>
<td>folder which contains the automator and variable substation information</td>
</tr>
<tr>
<td>automater.file</td>
<td>Project automator file path</td>
</tr>
<tr>
<td>var.sub</td>
<td>Variable substitution file path</td>
</tr>
<tr>
<td>proj.name</td>
<td>Project Name in project automator</td>
</tr>
<tr>
<td>proj.map</td>
<td>Map name in Project automator, it is used during variable substitution</td>
</tr>
<tr>
<td>proj.dc</td>
<td>Deployment candidate name in project automator</td>
</tr>
<tr>
<td>deployer.host</td>
<td>Deployer host name</td>
</tr>
<tr>
<td>deployer.port</td>
<td>Deployer port number</td>
</tr>
<tr>
<td>deployer.user</td>
<td>Deployer user name</td>
</tr>
<tr>
<td>deployer.pwd</td>
<td>Deployer user password</td>
</tr>
</tbody>
</table>

Add the ANT target to build.xml to create (using project automator) and deploy projects.

**ANT target Name:** createProject

**Description:** This ANT target takes project automator as an input and creates projects on deployer.

```xml
<target name="createProject">
  <echo message="Automater file name ....${automater.file}" />
  <if>
    <os family="windows"/>
    <then>
      <exec executable="cmd" dir="${deployer.home}/bin" failonerror="true">
        <arg value="/c" />
        <arg value="projectautomator.bat" />
        <arg value="${automater.file}" />
      </exec>
    </then>
    <else>
      <exec executable="/bin/bash" dir="${deployer.home}/bin" failonerror="true">
        <arg value="projectautomatorUnix.sh" />
        <arg value="${automater.file}" />
      </exec>
    </else>
  </if>
</target>
```
<delete file="${deployer.home}/replicate/inbound/${var.sub}"
<echo message="Deleting the old variable substitutions...."/>
</target>

**ANT target Name:** importVarSubs

**Description:** This ANT target takes variable substation file as an input and imports substitutions to the assets.

```xml
<target name="importVarSubs">
    <copy file="${env.dir}/${var.sub}" todir="${deployer.home}/replicate/inbound"/>
    <echo message="Copying the new variable Substitutions...."/>
    <echo message="Importing the new variable Substitutions...."/>
    <if>
        <os family="windows"/>
        <then>
            <exec executable="cmd" dir="${deployer.home}/bin" failonerror="true">
                <arg value="/c" />
                <arg value="Deployer.bat" />
                <arg value="--import" />
                <arg value="--varsub" />
                <arg value="--vsFile" />
                <arg value="${var.sub}" />
                <arg value="--map" />
                <arg value="${proj.map}" />
                <arg value="--project" />
                <arg value="${proj.name}" />
                <arg value="--validate" />
                <arg value="false" />
                <arg value="--host" />
                <arg value="${deployer.host}" />
                <arg value="--port" />
                <arg value="${deployer.port}" />
                <arg value="--user" />
                <arg value="${deployer.user}" />
                <arg value="--pwd" />
                <arg value="${deployer.pwd}" />
            </exec>
        </then>
        <else>
            <exec executable="/bin/bash" dir="${deployer.home}/bin" failonerror="true">
                <arg value="Deployer.sh" />
                <arg value="--import" />
                <arg value="--varsub" />
                <arg value="--vsFile" />
            </exec>
        </else>
    </if>
</target>
```
ANT target Name: DeployProject

Description: This ANT target deployes the project to the target server.

<target name="DeployProject">
  <echo message="Deploying the project to .... ${env}...!"/>
  <if>
    <os family="windows"/>
    <then>
      <exec executable="cmd" dir="${deployer.home}/bin" failonerror="true">
        <arg value="/c" />
        <arg value="Deployer.bat"/>
        <arg value="--deploy" />
        <arg value="--dc" />
        <arg value="${proj.dc}" />
        <arg value="--project" />
        <arg value="${proj.name}" />
        <arg value="--host" />
        <arg value="${deployer.host}" />
        <arg value="--port" />
        <arg value="${deployer.port}" />
        <arg value="--user" />
        <arg value="${deployer.user}" />
        <arg value="--pwd" />
        <arg value="${deployer.pwd}" />
        <arg value="--reportFilePath" />
        <arg value="${build.output.dir}"/>
      </exec>
    </then>
  </if>
</target>
11.0 Build & Deployment Workflow

11.1 Check out with Asset Build Environment Vs Check out with CI Tool

Asset Build Environment comes up with build-source-checkout.xml which contains the VCS details and check out directory. This still works but when check out is triggered every time ABE deletes and starts checking out the same set even there is no change checked into VCS. Imagine if VCS is having around 100 IS packages. This causes a lot of time consumption.

CI Tools comes up with different types of VCS plug in’s which can be configured with a job to perform VCS updates when new changes are checked into VCS. Following section will help to configure the same.
11.2 Checkout using CI Tool
Let us consider Jenkins as CI tool in this case.

a. Click on **New Item** from the left menu and select **Build a free-style software project** option. Provide the name and click on ok.

b. Navigate to **Source Code Management section** in configure link of the Job. Select the VCS type and map the credentials.

**SVN:**
11.3 Build & Deployment

Navigate to Build section. Click on Add build step, and select Invoke Ant option.
Select the ANT Version (the one which we have configured in Step 5.2), in the targets section provide “build” as the target method and finally in Build file section enter the “build.xml” under master build folder of Asset Build Environment. Click on Save to apply the changes.

To perform deployment without variable substitution navigate to build section and invoke ant targets that “createProject” and “DeployProject”.

To perform deployment with variable substitution navigate to build section and invoke ant targets that we have added in section 5.3 in the below order. “createProject”, “importVarSubs” and then “DeployProject”.
12.0 Change History

<table>
<thead>
<tr>
<th>Author</th>
<th>Comments</th>
<th>Date</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Srikanth Prathipati</td>
<td>Initial draft</td>
<td>12/12/2015</td>
<td>0.1</td>
</tr>
</tbody>
</table>