



Professional
Services

BW 6 Best Practices Guide

Project Name	BW 6 Best Practices
Release	V1.5
Date	March 24, 2016
Primary Author	Heejoon Park
Contributing Authors	Pong-Ning Ching, Anna Papp, Kamal Kumar
Document Owner	TIBCO Software
Document Location	
Purpose	This document provides the BW 6 best practices guide on select topics including exception management and handling, end-to-end deployment process (CI/CD support), run-time deployment architecture and tuning among others.

TIBCO Software empowers executives, developers, and business users with Fast Data solutions that make the right data available in real time for faster answers, better decisions, and smarter action. Over the past 15 years, thousands of businesses across the globe have relied on TIBCO technology to integrate their applications and ecosystems, analyze their data, and create real-time solutions. Learn how TIBCO turns data—big or small—into differentiation at www.tibco.com.

www.tibco.com



Global Headquarters
3303 Hillview Avenue
Palo Alto, CA 94304

Tel: +1 650-846-1000
+1 800-420-8450
Fax: +1 650-846-1005

Revision History

Version	Date	Author	Comments
1.0	11/16/2015	Heejoon Park	Initial Draft
1.2	11/19/2015	Heejoon Park	Contents added to Ch 2, 3, and 4
1.3 (Draft)	11/20/2015	Heejoon Park	Contents added to Ch 1, 5, 6, and Appendix A
1.4	24/03/2016	Kamal Kumar	Cleaned up, added Architecture Diagrams

Approvals

This document requires the following approvals. Signed approval forms are filed in the project files.

Name	Signature	Title	Company	Date of Issue	Version

Distribution

This document has been distributed to:

Name	Title	Company	Date of Issue	Version

Copyright Notice

COPYRIGHT© 2016 TIBCO Software Inc. This document is unpublished and the foregoing notice is affixed to protect TIBCO Software Inc. in the event of inadvertent publication. All rights reserved. No part of this document may be reproduced in any form, including photocopying or transmission electronically to any computer, without prior written consent of TIBCO Software Inc. The information contained in this document is confidential and proprietary to TIBCO Software Inc. and may not be used or disclosed except as expressly authorized in writing by TIBCO Software Inc. Copyright protection includes material generated from our software programs displayed on the screen, such as icons, screen displays, and the like.

Trademarks

All brand and product names are trademarks or registered trademarks of their respective holders and are hereby acknowledged. Technologies described herein are either covered by existing patents or patent applications are in progress.

Confidentiality

The information in this document is subject to change without notice. This document contains information that is confidential and proprietary to TIBCO Software Inc. and its affiliates and may not be copied, published, or disclosed to others, or used for any purposes other than review, without written authorization of an officer of TIBCO Software Inc. Submission of this document does not represent a commitment to implement any portion of this specification in the products of the submitters.

Content Warranty

The information in this document is subject to change without notice. THIS DOCUMENT IS PROVIDED "AS IS" AND TIBCO MAKES NO WARRANTY, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO ALL WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. TIBCO Software Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Export

This document and related technical data, are subject to U.S. export control laws, including without limitation the U.S. Export Administration Act and its associated regulations, and may be subject to export or import regulations of other countries. You agree not to export or re-export this document in any form in violation of the applicable export or import laws of the United States or any foreign jurisdiction.

For more information, please contact:

TIBCO Software Inc.
3303 Hillview Avenue
Palo Alto, CA 94304
USA

Table of Contents

1	Introduction.....	6
1.1	Scope	6
2	Exception Management and Handling.....	7
2.1	Overview	7
2.2	Relevant BW 6 Features and Best Practices	7
2.2.1	How to pass exceptions to the parent process.....	7
3	End-to-end Deployment Process (CI/CD Support)	11
3.1	Overview	11
3.2	Implementation Steps	11
3.2.1	Building and Deploying Applications via Maven.....	11
3.2.2	Storing BW Application on a Git Server	14
3.2.3	Creating a Jenkins Job.....	17
3.2.4	Automating Build and Deploy	21
3.2.5	Next Steps.....	23
4	Deployment Architecture and Tuning.....	24
4.1	Overview	24
4.2	BW 6 Domain and AppSpace Design Consideration	24
4.3	AppSpace and AppNode Tuning Parameters and Guidelines	25
4.4	BW6 JVM Tuning Considerations and its Parameters (AppNode and AppSpace)	27
5	Other BW 6 Best Practices Relevant to Customers	29
5.1	Overview	29
5.2	Migrating Java Custom Functions from BW 5.x to 6.x Custom XPath Functions	29
5.3	Sharing Project Artifacts through a Shared Module	29
5.4	Using Policy in BW 6.....	29
6	Frequently asked BW 6 Questions.....	30
6.1	Overview	30
6.2	Questions and Answers	30
	APPENDIX A. BW 6.x Fault Tolerance and Load Balancing	32
	Introduction	32
	Managed Fault Tolerance	32
	Managed Fault Tolerance: Single AppNode (Active-Passive)	33
	Managed Fault Tolerance: Multiple AppNode (Active-Active)	34
	Non Managed Fault Tolerance	35
	Configuration Background	36
	Adding multiple bwagents to a TEA admin	36
	FT Configuration Steps are as Follows for BW 6.x	37
	The bwagent Configuration (for 6.x)	38

Table of Figures

Figure 1. Example of a Simple Common Exception Schema.....	8
Figure 2. Adding Fault Element to a Sub-process WSDL	8
Figure 3. Referencing Common Exception Schema from Fault Element.....	9
Figure 4. Configuring “Reply With:” option on “General” tab to Return a Fault	9
Figure 5. Referencing Common Exception Schema from Throw Activity.....	10
Figure 6. Running “Generate POM” Menu Command.....	12
Figure 7. Maven Configuration Details Window	13
Figure 8. Running “Maven build...” Command	13
Figure 9. “Edit Configuration” Window	14
Figure 10. Creating a New GIT Repository	15
Figure 11. Retrieving GIT Commands for the Sample Repository	16
Figure 12. Checking in the Sample BW Projects to a Git Server	16
Figure 13. Verifying the Sample Project Artifacts being Pushed to the Remote Git Server.....	17
Figure 14. Configuring Jenkins Service to Run under Your User Account.....	18
Figure 15. Configuring Jenkins Source Code Management Section.....	18
Figure 16. Configuring a Build Step	19
Figure 17. Configuring a Build Step Continued	19
Figure 18. Running a Jenkins Job	20
Figure 19. Locating the Console Output for the Job.....	20
Figure 20. Console Output for a Jenkins Job Instance	21
Figure 21. Configuring Webhook in the Gogs Git Server	21
Figure 22. Configuring Webhook Continued	22
Figure 23 Managed Fault Tolerance	32
Figure 24 Managed Fault Tolerance: Single AppNode	33
Figure 25 Managed Fault Tolerance Multiple AppNode	34
Figure 26 Non Managed Fault Tolerance.....	35

Table of Tables

No table of figures entries found.

1 Introduction

1.1 Scope

The topics that should be covered by BW 6 Best Practices Guide would be vast. This guide is not exhaustive in its coverage of core BW 6 topics and has been specifically tailored to address topics that are most relevant.

2 Exception Management and Handling

2.1 Overview

There are differences between how BW 5 and BW 6 respectively manage and handle exceptions. For one, parent to child process interaction in BW 6 is implemented by invoking a service described by an abstract WSDL. This requires different techniques than how you might be used to handling exceptions that were thrown in BW 5. In a nested process hierarchy scenario (e.g., assume three-level process hierarchy), the most top level process can still catch the exception thrown by its child's child process without losing the exception details. However, such is not the case with BW 6. You would not see this issue in a simple parent to child interaction. However, once you start to have more than three levels of process hierarchy, then you need to figure out how you can consistently bubble up exceptions from the lowest level to the top most process without losing any exception details via the intermediary processes.

Due to this reason, we recommend to remove the current exception handling logic found in BW 5 assets before running the Migration tool. We have also observed while performing migration analysis that there were different exception handling schemes deployed by BW 5 projects. In one case, exceptions were caught at a given process level, which was then handled at the same level. In other cases, we saw exceptions being captured at different levels which were then bubbled up to the top most process wherein they were handled. We all understand that in such cases we need to strive for consistency and to promote ease of maintenance and readability, it still makes sense to bubble up exceptions to the top most process to handle them. We will cover exactly how to do this in BW 6 in following sections.

2.2 Relevant BW 6 Features and Best Practices

2.2.1 How to pass exceptions to the parent process

As was pointed out in the previous section, every parent to child interaction is described by a service invocation with an accompanying WSDL. To ensure such interactions are local within a BW instance, choose abstract WSDL option ignoring any binding details.

First step would be to standardize on a common exception schema to be used across different projects. Figure 1 below shows an example of a simple yet, effective common exception schema that can be used. Possible enumeration values you can set for "exceptionType" might include "TECHNICAL", "BUSINESS", and "UNHANDLED". The "exceptionCode" might contain generic exception code hierarchy that reflects certain structure and organizations on how errors are to be categorized as per needs. Some customers take the option of populating the "message" field by performing a cache lookup using the exceptionCode as input to manage error content and communications in a centralized manner. The last point is more of an optional area that can consider. The "KeyValuePair" provides flexible means in the form of a name value pair that can add any relevant contextual data that can help DevOps/Support to perform troubleshooting faster. The bottom line here is that it should standardize on a common exception and log schema that can capture the right size of exception related data to effectively perform troubleshooting tasks. Such common schemas and any related common processes should be configured as part of a BW Shared Module.

Once exception schemas have been identified, each WSDL describing parent-child interaction must be reflected with a Fault element referencing the aforementioned exception schema. Figure 2 below shows a Studio Designer opened with such a WSDL to which a Fault element is being added.

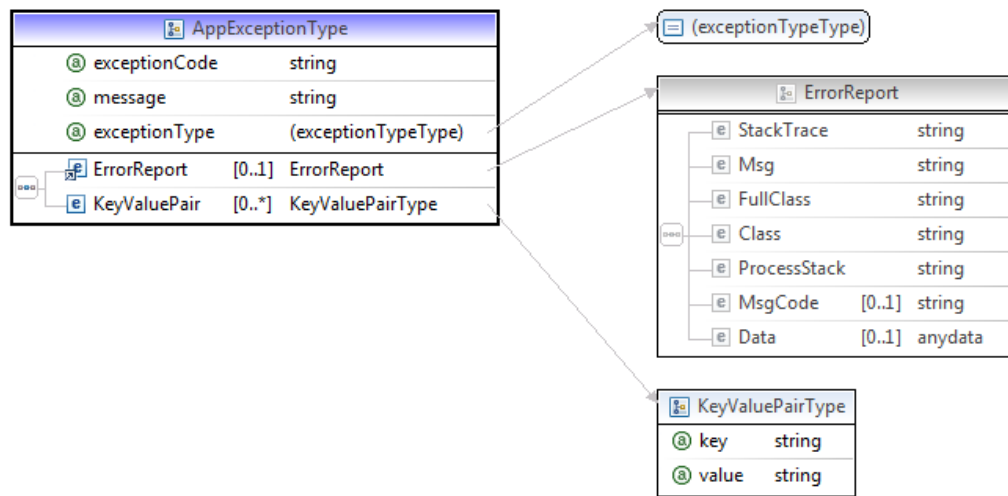


Figure 1. Example of a Simple Common Exception Schema

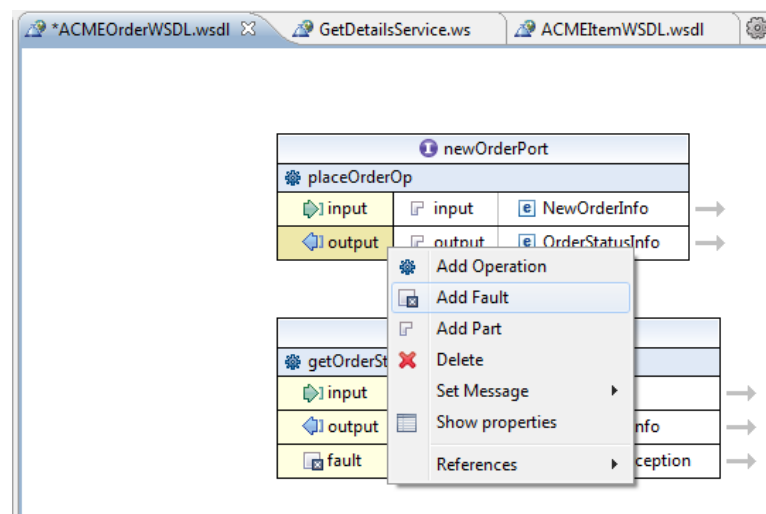


Figure 2. Adding Fault Element to a Sub-process WSDL

Figure 3 below shows how you can configure the Fault element to reference the common exception schema.

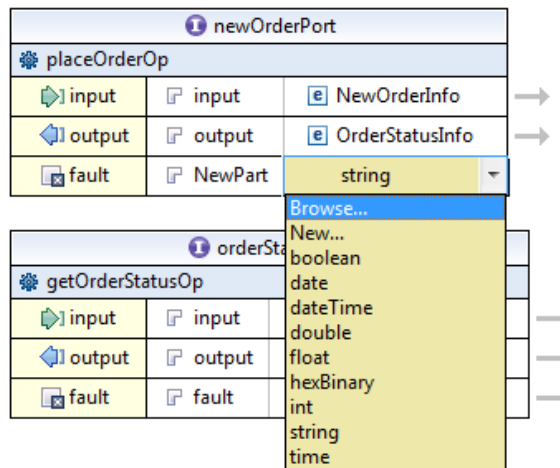


Figure 3. Referencing Common Exception Schema from Fault Element

Once the subprocess WSDL has been configured with a Fault exception schema, then you can use the “Basic Activities”-> “Reply” activity to return the exception to its parent process. You can apply this same technique to pass and bubble up exceptions to the top most process in a nested process hierarchy with a degree of two or more. The “Reply” activity as part of exception management should be called inside the “Catch” or “Catch All” block as shown in Figure 4 below. Make sure you choose the Fault element you have configured previously under the “Reply With:” option on the “General” tab within the “Reply” activity. You can then populate the Fault exception schema with Fault details on the “Input” tab.

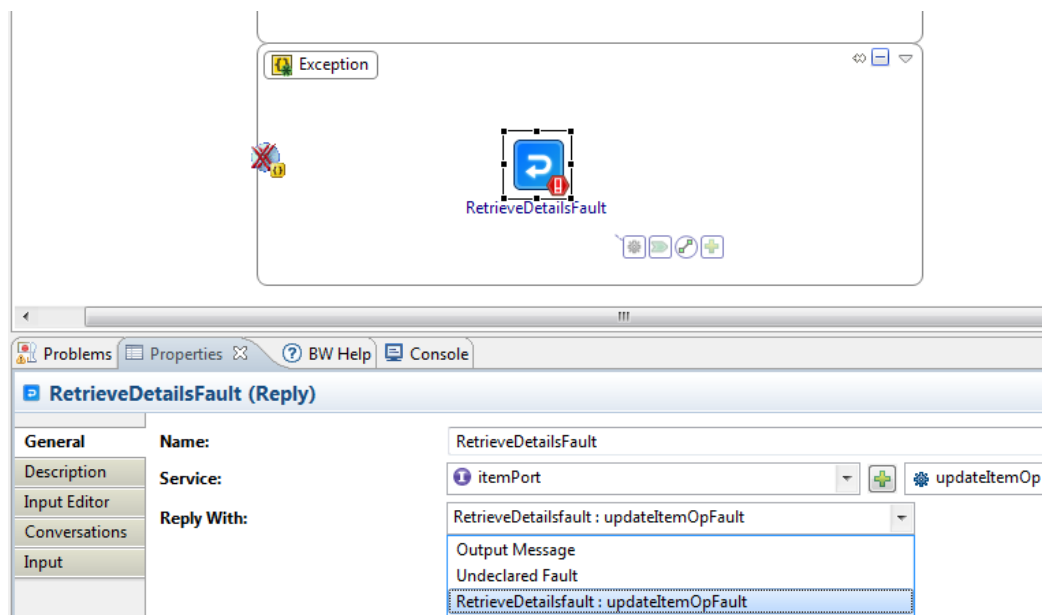


Figure 4. Configuring “Reply With:” option on “General” tab to Return a Fault

It would be prudent to determine the right extent to which specific errors would be caught and the rest being managed and categorized as “UNHANDLED” exceptions. To truly speaking, these are not unhandled per se, but rather generic exceptions since you are capturing such exceptions using “Catch All”. For throwing and catching specific exceptions, you can use a pair of “Basic Activities”->“Throw” activity along with a “Catch” block containing a “Reply” activity to pass it up to the parent process. In this case, you can use a “Throw” activity to check for both business and technical exceptions of importance to you. Using the transition type of “Success with condition” leading into the “Throw” activity, you can check for existence of business exception conditions to be reported and handled. One example of a business exception might involve more of a semantic validation/checking you need to perform beyond straight-forward syntactic validation. Similarly, you can also use “Error” transition type leading into the “Throw” activity to throw a technical exception that reports abnormal technical behavior.

While configuring the “Throw” activity, you can use “Select Input Editor Element” button on the “Input Editor” tab to select the desired error schema, preferably the common exception schema.

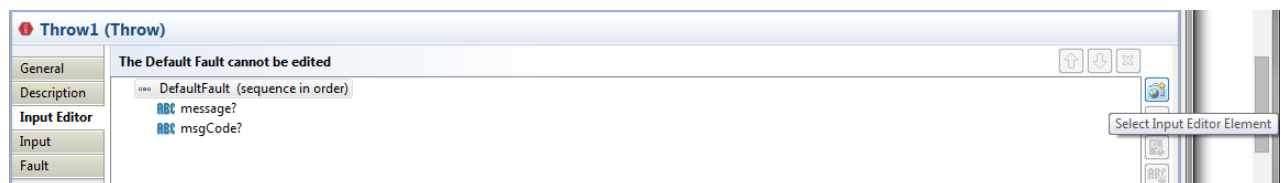


Figure 5. Referencing Common Exception Schema from Throw Activity

On the receiving end, you can use the accompanying “Catch” block to catch for that specific error. The “Input” tab within the “Reply” activity inside that “Catch” block would show the specific fault that was thrown by that “Throw” activity (The same technique can be used for catching for a specific fault thrown by a subprocess). Any of potential technical exceptions for which you are not explicitly throwing by using the “Throw” activity can be caught by using the “Catch All” block. In such cases, run-time determined exception details would be captured under the default “FaultDetails” element within the “Input” tab of the “Reply” activity inside the “Catch All” block. The “FaultDetails” element contains the optional “Data” element which in turn contains an any element. At run-time, depending on which activity that would be erroring out, the any element would be populated accordingly by BW 6.

Again, it would not make sense to use “Throw” activity to catch for every possible technical exceptions that can be thrown. Better approach would be to identify a list of technical exceptions of common occurrence and of high importance to be candidates for “Throw” activity and handle others via “Catch All” block. Based on DevOps/Support feedback over periods of use, refine this model iteratively to arrive at the right balance between specific and generic error capture.

Fault Handlers can also be attached at a scope level in addition to the process level. Such technique can help organize BW 6 exception handling logic to mimic try-and-catch block pattern in Java.

3 End-to-end Deployment Process (CI/CD Support)

3.1 Overview

There are two basic challenges pertaining to software delivery in a software development lifecycle: 1) Continuously building the artifacts and constantly testing the changes to make sure nothing is broken and superior software quality can be guaranteed; and 2) Promotion of code changes to different environments with quality control enforced.

An industry trend called Continuous Delivery (CD) addresses the aforementioned challenges. At its core, CD is all about streamlining the environment migration of the code base built by Continuous Integration (CI). This could be DEV, SIT, UAT, PERF, and PROD.

CI is the practice of automatically testing each change as early as possible thus allowing frequent code changes without comprising on quality. And CD automates the code promotion from environment to environment and leverages CI. Technical and business benefits of CI/CD are listed below.

Technical Benefits

- Eradication of manual deployment steps
- Prevention & reduction of production & staging errors
- Generation of analysis & reporting on health and quality of the code base

Business Benefits

- Proactively manages risks concerning schedule, cost and budget as quality and efficiency is at the core
- Reducing overheads across the development & deployment process by detecting system development problems earlier
- Enhancing the reputation of the delivery area by providing Quality Assurance at the early stage and throughout

With this in mind, we recommend that customer put together an end-to-end deployment framework/process along with supporting tools for the BW 6 environment. This chapter will provide guidance and best practices on how to enable such framework integrating BW 6 development (*BW 6 Studio Designer*) and run-time (*bwadmin* commands) environments using standard industry tools such as Git, Gogs, Maven, and Jenkins.

3.2 Implementation Steps

3.2.1 Building and Deploying Applications via Maven

1. Follow the steps outlined in the "Readme.txt" file under <TIBCO_HOME>\bw\6.3\samples\plugins\maven to install Maven and Maven plugin for BusinessStudio.
2. Import a sample application to be managed via Maven into BusinessStudio. For this sample, we have chosen <TIBCO_HOME>\bw\6.3\samples\binding\rest\Basic\ tibco.bw.sample.binding.rest.Basic.zip project.
3. Run "Generate POM" command from the popup menu as you right click on the BW Application as shown in Figure 6 below.

4. Once “Maven Configuration Details” window shows up, enter values for Domain, AppSpace, and AppNode to which the BW application is to be deployed as shown in Figure 7 below. You will see ‘M’ is added next to the projects’ icon to indicate Maven association.
5. You can now test Maven plugin by generating an EAR file. Right click on the sample application and select “Run As->2 Maven build...” as shown in Figure 8 below. This brings up the “Edit Configuration” Window to which you can enter “package” as the value for the “Goals:” text field. Click “Apply” and “Run”.
6. The EAR file will be created as a JAR file which you can confirm by going to <Your_BS_Workspace>\<name of application>\target directory. You should see a JAR file with a timestamp in its name.
7. To create the Domain, AppSpace, AppNode, and to upload, deploy, and run the sample Application, repeat the same step as 5 above, however, in the “Goals:” text field, enter the value of “install”. Click “Apply” and “Run”. Make sure that *bwagent* is already running on your machine and that it has already been registered with TEA.
8. You can confirm that Application has been successfully deployed and is running by logging into TEA.

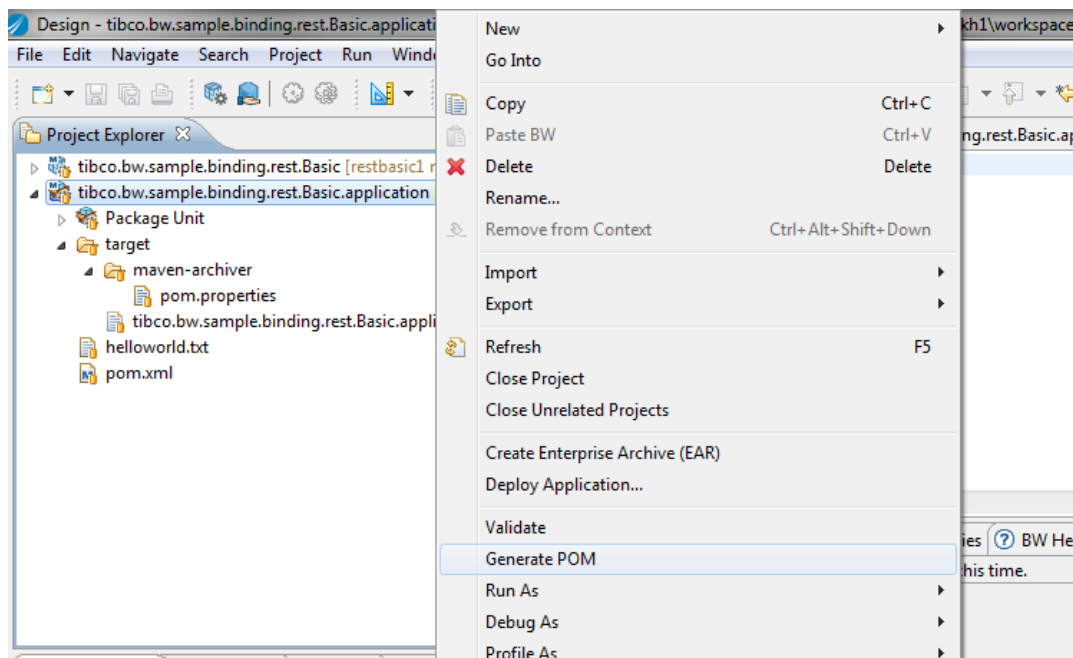
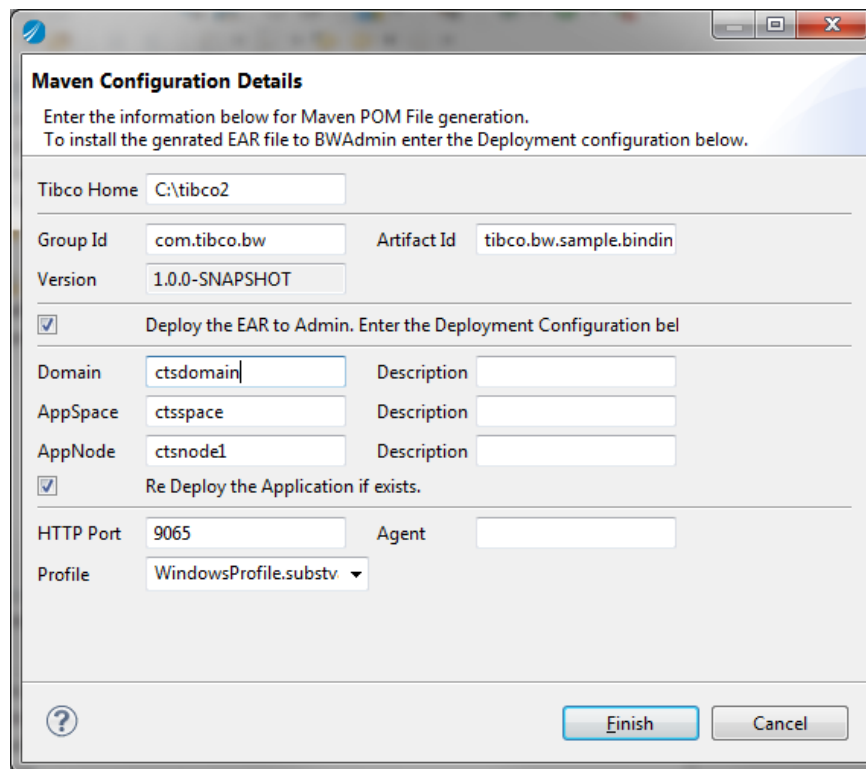


Figure 6. Running “Generate POM” Menu Command



Maven Configuration Details

Enter the information below for Maven POM File generation.
To install the generated EAR file to BWAdmin enter the Deployment configuration below.

Tibco Home

Group Id Artifact Id

Version

☒ Deploy the EAR to Admin. Enter the Deployment Configuration below

Domain Description

AppSpace Description

AppNode Description

☒ Re Deploy the Application if exists.

HTTP Port Agent

Profile

Figure 7. Maven Configuration Details Window

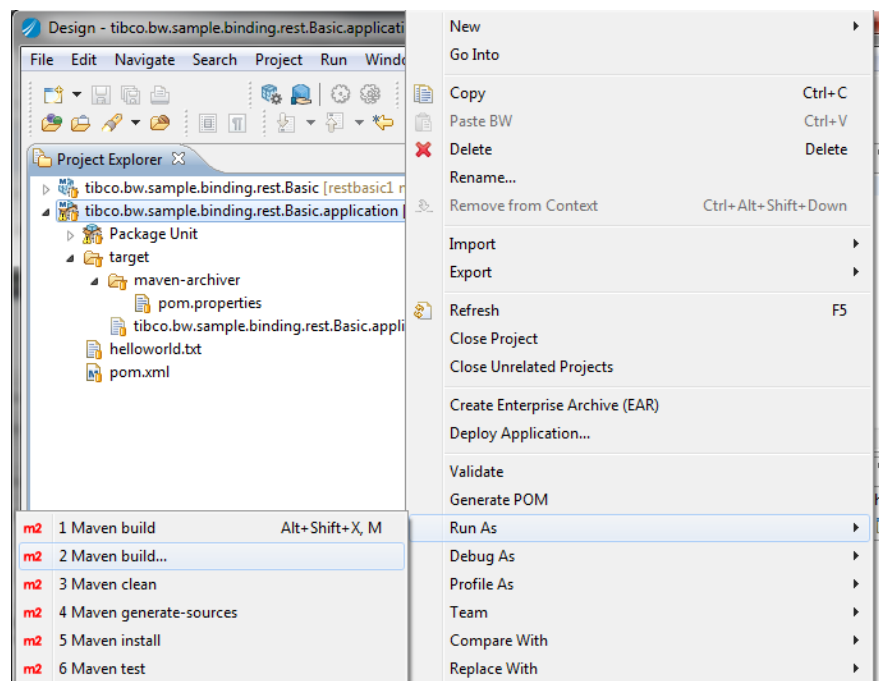


Figure 8. Running "Maven build..." Command

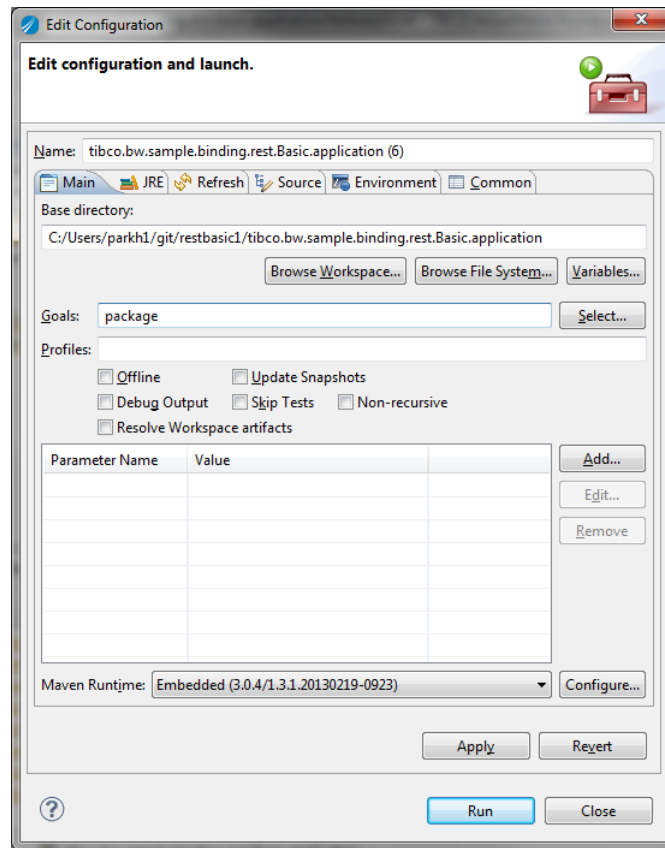


Figure 9. "Edit Configuration" Window

3.2.2 Storing BW Application on a Git Server

This section will walk you through on how to configure a Gogs Git server, add the sample BW project repositories to Git, and to create a job in Jenkins to automate the build.

1. Start Gogs Git web server by running `<gogs_home_dir>\gogs web`.
2. Create a new GIT repository by going to <http://localhost:3000> on your Chrome browser and sign in (top right) using the user name you created during the install.
3. Click on the green + symbol and choose "New Repository" option as shown in Figure 10.

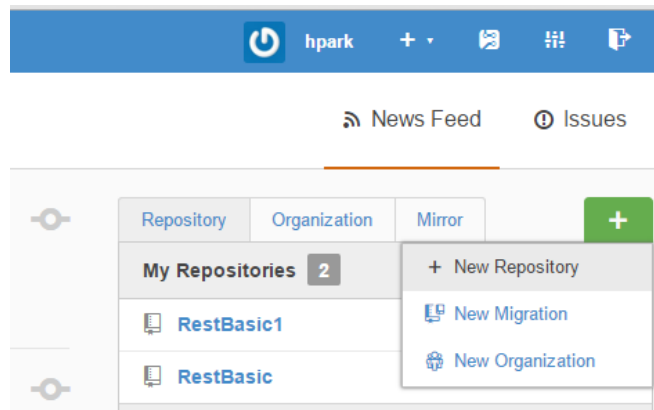


Figure 10. Creating a New GIT Repository

4. In the “New Repository” page, enter a value for the “Repository Name” text field. In our testing, we have chosen the value of “RestBasic2”. Leaving other values at their default, click “Create Repository” button to proceed.
5. In the new page that shows up, click “HTTPS” option as shown on Figure 11 below and note the HTTPS URL and the git commands under the “*Push an existing repository from the command line*” heading as these commands will be used later on in this section to add the newly created repository to the remote repo server and pushes the committed changes in the local repo to the remote repo.
6. Back in BusinessStudio, we need to configure the sample BW projects to work collaboratively with other team members. In the *Project Explorer*, highlight both projects and right-click on them and select **Team->Share Project...**
7. Select **Git** and click Next. Next to Repository text box, click the **Create...** button.
8. Once the “Create a Git Repository” popup window shows up, enter values for the **Parent Directory** and **Name** and click **Finish**. (e.g. In our setup, we have used “C:\Users\parkh1\git” for the **Parent Directory**, and “RestBasic2” for the **Name** text field.) Click **Finish** again.
9. Check in the entire project by pushing its files up to a Git server. In the *Project Explorer*, highlight both projects and right-click on them and select **Team->Commit...** as shown in Figure 12 below.
10. Once the “Commit Changes” popup window appears, enter a value for the “Commit message” text area. For example, we have entered the value of “v1.0.0”. Click the middle icon with a check mark on it to select all the project artifacts. Click the “Commit” button at the bottom.

Clone this repository

SSH HTTPS <http://localhost:3000/hpark/restbasic2.git>

Copy

Need help cloning? Visit [Help!](#)

Create a new repository on the command line

```
touch README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin http://localhost:3000/hpark/restbasic2.git
git push -u origin master
```

Push an existing repository from the command line

```
git remote add origin http://localhost:3000/hpark/restbasic2.git
git push -u origin master
```

Figure 11. Retrieving GIT Commands for the Sample Repository

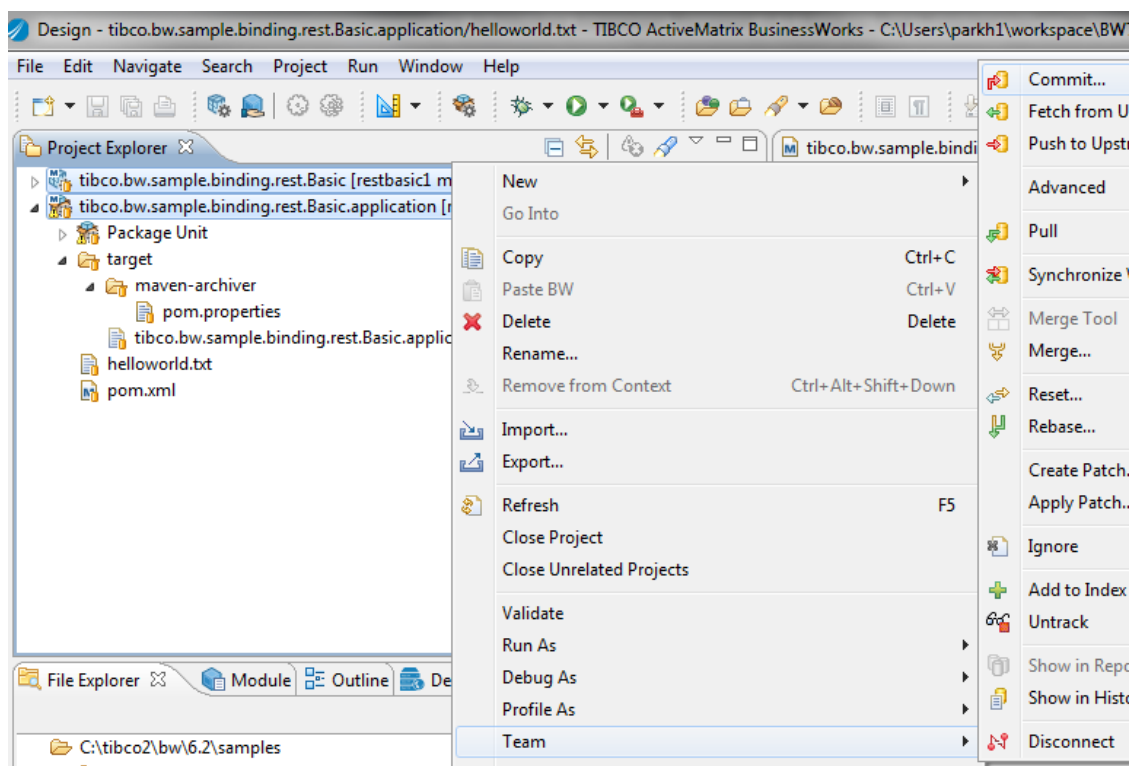


Figure 12. Checking in the Sample BW Projects to a Git Server

11. The project code is now checked in to a local Git repo. Now we can push this code into the remote Git server managed by the Gogs Git server. Open up a Windows command window and change directory to your local repository. Recall from step 8 above that we have set the local repository directory at C:\Users\parkh1\git\RestBasic2.
12. Run the Git commands you have noted from step 5 above.
c:\Users\parkh1\git\restbasic2>git remote add origin <http://localhost:3000/hpark/restbasic2.git>
c:\Users\parkh1\git\restbasic2>git push -u origin master
13. Verify by going to <http://localhost:3000/hpark/RestBasic2> that both of your project artifacts are visible as shown in Figure 13 below. In your case, you should see v1.0.0 for all of your project artifacts, not v2.0.0 as shown in below Figure.

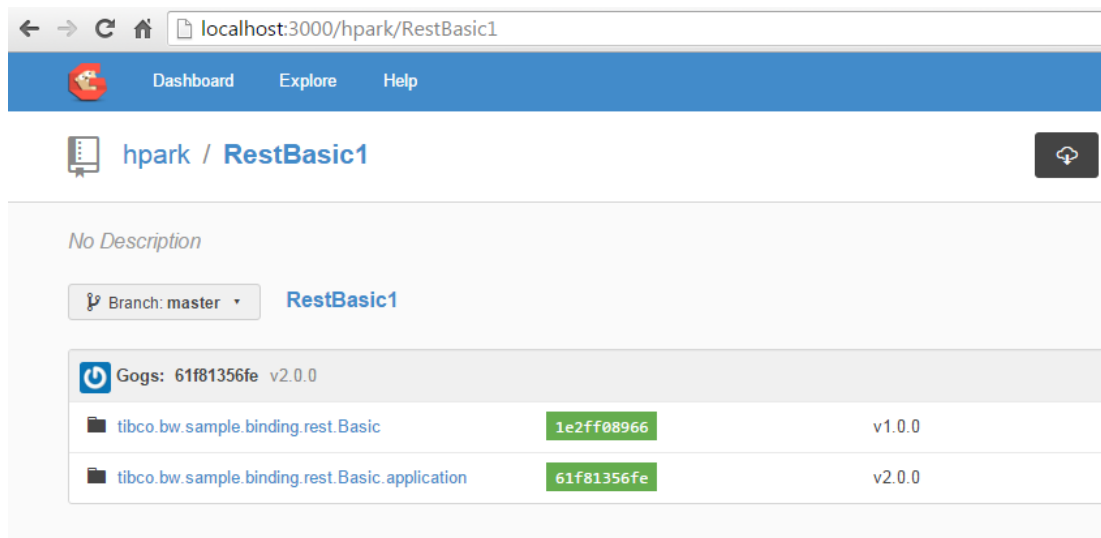


Figure 13. Verifying the Sample Project Artifacts being Pushed to the Remote Git Server

14. At this point, both BW projects and all their files are now imported into a Git server. However, the process requires manual build from BW Studio. In the next section, we will walk you through on how to create a Jenkins job to start to automate your BW build and deployment.

3.2.3 Creating a Jenkins Job

This section will walk you through on creating a Jenkins job.

1. Before you proceed further, make sure that Jenkins service is running under your account, not the default system user as shown in Figure 14 below. This advice will save you a lot of time. ☺
2. Change directory to the Maven bin directory per your installation and run the following command.
e.g. C:\work\Software\apache-maven-3.3.3\bin>copy mvn.cmd mvn.bat
3. The previous step is required as Maven version 3.3.3 changed the main executable name from *mvn.bat* to *mvn.cmd* which produces errors when other tools were calling *mvn.bat* commands in their scripts.
4. On a Chrome web browser, go to <http://localhost:8080>. Click “New Item” at the top left corner. In the new page, enter a value for the “Item name” field. In our setup, we have used the value of “restbasic1”. Select the “Freestyle project” and click the “OK” button. Locate the “Source Code Management” section in the new page.

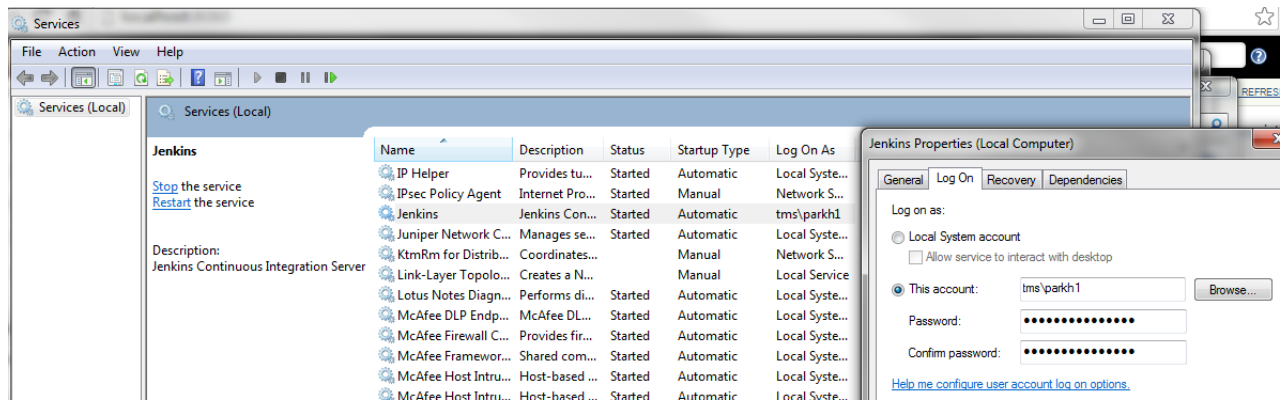


Figure 14. Configuring Jenkins Service to Run under Your User Account

5. Select the Git option. If you do not see this option show up, that means you have not yet installed the Git Plugin – Jenkins (v2.4.0). Enter the value for the “Repository URL” field as shown in Figure 15 below.

Source Code Management

☐ None
☐ CVS
☐ CVS Projectset
☒ Git

Repositories

Repository URL:

Credentials:

Branches to build

Branch Specifier (blank for 'any'):

Repository browser:

Additional Behaviours:

Figure 15. Configuring Jenkins Source Code Management Section

6. On the same page, scroll down to “Build” section. Click on the “Add build step” and choose “Invoke top-level Maven targets”. For the “Maven Version” field, choose the “(Default)” dropdown value. Configure the “Goals” field with the value of “package”. Click “Advance” and configure the “POM” field with the value of “tibco.bw.sample.binding.rest.Basic.application/pom.xml” as shown in Figure 17 below.
7. Click “Add build step” again and choose “Invoke top-level Maven targets”. Configure the “Maven Version” and the “POM” field with the same values as in Step 6 above. For the “Goals” field, for this new step enter the value of “install”. Click “Save”. You have now created a Jenkins job called “restbasic1”.

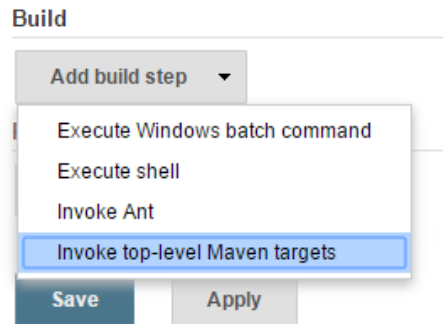


Figure 16. Configuring a Build Step

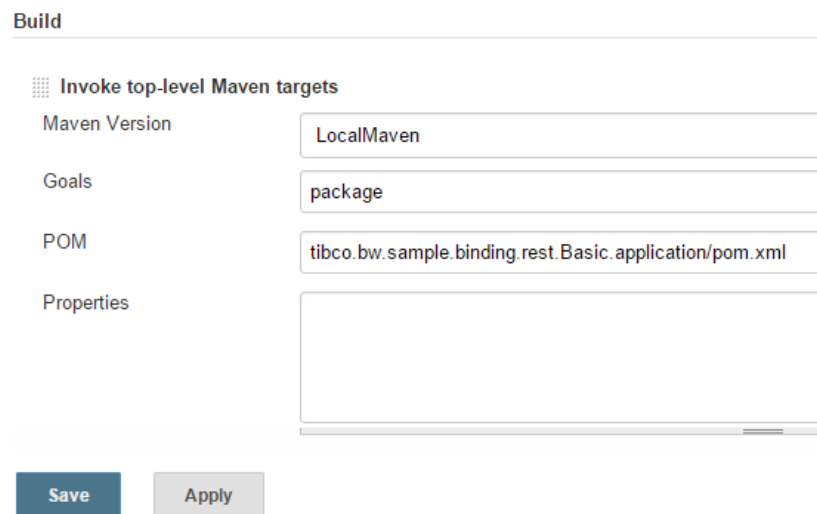


Figure 17. Configuring a Build Step Continued

8. On the main Jenkins page, click on the small inverse triangle next to the name of the Jenkins Job you want to run and choose “Build Now” as shown in Figure 18 below.
9. On the main Jenkins page, under the “Build Executor Status” section, locate the Jenkins job you just started. Click on the small inverse triangle next to the number that is located to the right of the name of the job. Choose the “Console Output” option as shown in Figure 19 below. This number represents the specific job instance which is incremented each time a job is invoked.
10. Verify the console output which gives detail logs of the build. Make sure the build was successful as shown in Figure 20 below.
11. We have now configured a Jenkins job that builds an EAR file from the Git repository, creates run-time objects such as Domain, AppSpace, and AppNode per our specification (recall Figure 7 above), uploads the EAR file to the specified Domain, deploys the EAR file to the specified AppSpace, and runs the application on the specified AppNode. However, we have run this Jenkins job from the Jenkins Dashboard by manually invoking it. The next section shows you how you can automate the build by invoking a Jenkins job when a new code is committed.

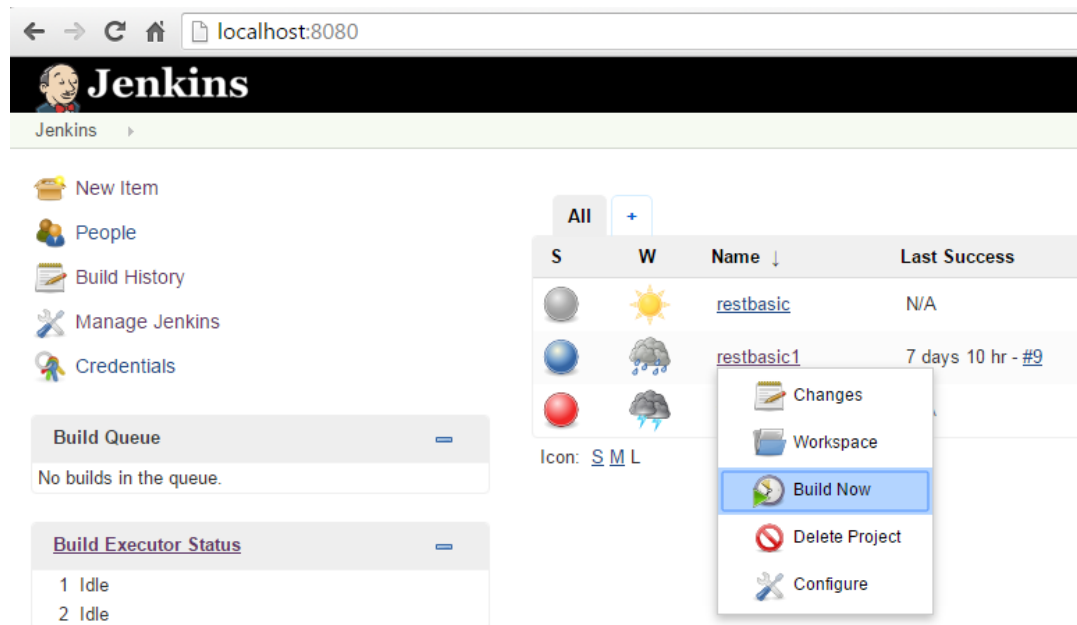


Figure 18. Running a Jenkins Job

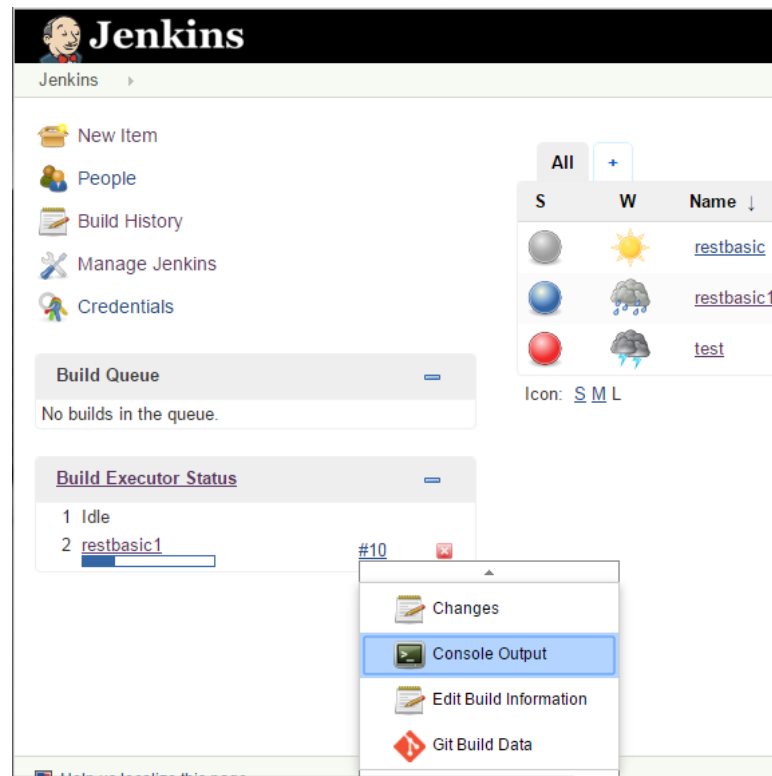


Figure 19. Locating the Console Output for the Job

```
[INFO] Executing command => [C:\tibco2\bw\6.2\bin\bwadmin.exe, deploy, -domain, ctsdomain, -appspace, ctsspace, -p,
WindowsProfile.substvar, -as, tibco.bw.sample.binding.rest.Basic.application.ear]
[INFO] Exit Value is 0
[INFO] TIBCO ActiveMatrix BusinessWorks version 6.2.2, hotfix 2, build V26, 2015-8-27Connecting to the BW Agent using
TIBCO ActiveSpaces (Java). Enterprise Edition. Version 2.1.4.022 [2015-11-19T21:58:30.447][19092][21604][INFO]
[bwmetaspace.metaspace] proxy PL9414482:5075 resolved to 10.61.248.211:5075[2015-11-19T21:58:30.448][19092][21604]
[WARNING][bwmetaspace.metaspace][SpaceManager.cpp:77][SpaceManager::initialize] bad autojoin.role remoteclient using
seeder[2015-11-19T21:58:30.463][19092][20276][INFO][bwmetaspace.metaspace] connected to proxy,
proxy_address=PL9414482:5075[2015-11-19T21:58:30.542][19092][20276][INFO][bwmetaspace.metaspace] handshake with proxy
success[2015-11-19T21:58:30.542][19092][20276][INFO][bwmetaspace.metaspace] remote client joined with proxy[2015-11-
19T21:58:30.604][19092][21604][INFO][bwmetaspace.metaspace] Connected metaspace name=[bwmetaspace], discovery=[tibpgm],
remote_discovery=[tcp://PL9414482:5075?remote=true], member name=[PL9414482.admin.cli.c3ea052e-79f4-47a8-95f2-
cde24b74c5a9]], version=2.1.4.022Connected to BW AgentTIBCO-BW-ADMIN-CLI-300417: The application
[tibco.bw.sample.binding.rest.Basic.application:1.0] deployed and started.
[INFO] -----
[INFO] Reactor Summary:
[INFO]
[INFO] tibco.bw.sample.binding.rest.Basic ..... SUCCESS [ 3.050 s]
[INFO] tibco.bw.sample.binding.rest.Basic.application .... SUCCESS [02:07 min]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:30 min
[INFO] Finished at: 2015-11-19T21:58:39-08:00
[INFO] Final Memory: 43M/500M
[INFO] -----
Finished: SUCCESS
```

Page generated: Nov 19, 2015 9:55:37 PM [REST API](#) [Jenkins ver. 1.638](#)

Figure 20. Console Output for a Jenkins Job Instance

3.2.4 Automating Build and Deploy

In this section, we will configure the Gogs Git server to trigger an event by creating a webhook. The webhook will notify Jenkins that the repository had changes, and Jenkins will start a build job.

1. Add a Webhook in Gogs that posts an event each time the repository gets updated. Log into Gogs at <http://localhost:3000>. Click “RestBasic2”. Select “Settings” icon under master to the right as shown in Figure 21 below.
2. On the new page that shows up, choose the “Webhooks” option under the “Settings” menu on the left side. Once a new page is loaded, click on “Add Webhook” button on the top right side.

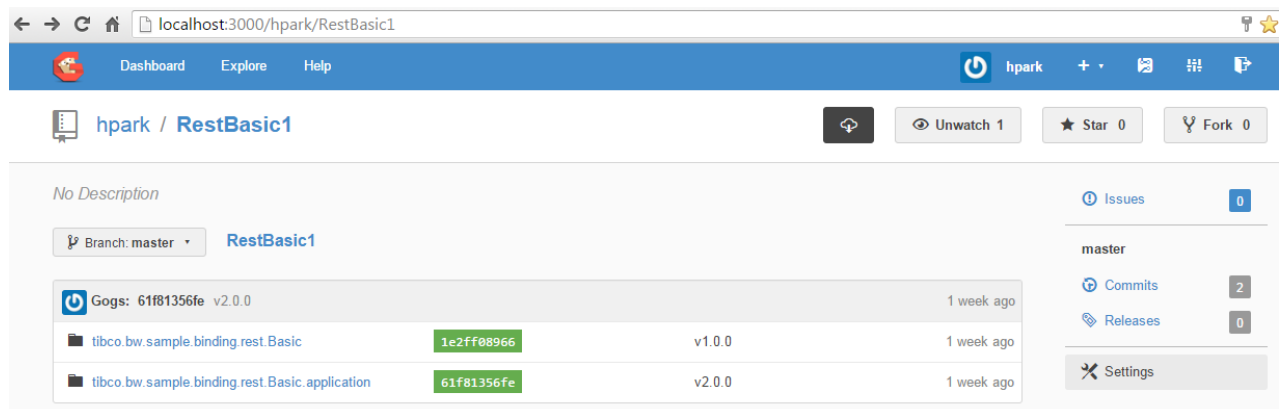


Figure 21. Configuring Webhook in the Gogs Git Server

Add Webhook

Hook Type* Gogs ▼

Gogs will send a POST request to the URL you specify, along with regarding the event that occurred. You can also specify what kind of data format you'd like to get upon triggering the hook (JSON, x-www-form-urlencoded, XML, etc). More information can be found in our [Webhooks Guide](#).

Payload URL* Content Type*

Secret

.....

Upon which events should this webhook be triggered?

☒ Just the push event.Active ☒ Details regarding the event which triggered the hook will be delivered as well.

Add Webhook

Figure 22. Configuring Webhook Continued

- On the “Add Webhook” page, configure the “Payload URL” field with the value of <http://127.0.0.1:8080/job/restbasic2/build>. This is the URL of the specific Jenkins job you want to call. Set the rest of the fields matching the values shown in the Figure 22 above. For the value of the “Secret” field, enter the password for the user name under which you have logged onto the Gogs Git Server. Click on the “Add Webhook” button. This webhook is configured to send event messages to the Payload URL whenever repository changes occur, which will trigger a Jenkins job to start a build.
- To test Gogs webhook, add a test file to the Basic REST application. Back in Business Studio, right-click and select **tibco.bw.sample.binding.rest.Basic.application > New > File**. Enter a value for the field, “File name” and click “Finish”. For our setup, we have named the file as “helloworld.txt”. Double click on the file and enter some texts like “Hello World”. Click “Save”.
- Check in the sample project to the Git repository by right-clicking on the sample application and selecting **Team>Commit...** Enter “v2.0.0” as the value for the “Commit message” text area. Click “Select All” in the middle section and click “Commit” button at the bottom. As you click “Commit” button, new and/or updated artifacts are committed into the local Git repo. In order to push this change into the remote Git repo, you need to run the following command:

```
c:\Users\parkh1\git\restbasic2>git push -u origin master
```

This then updates the remote repo which in turn triggers the webhook to send a HTTP post event to the Jenkins build job to start the automatic build! Confirm that a Jenkins job has been started.

3.2.5 Next Steps

What we have covered so far provides the foundational work on which we can build/augment/modify additional features and capabilities as it sees fit. Some of additional capabilities that we should consider integrating into Jenkins build process are:

- Procure, install, configure, and integrate automated document and quality control tools such as Make Doc.
- Procure, install, configure, and integrate automated unit testing tools such as junit.
- Procure, install, configure, and integrate automated regression testing tools such as SoapUI which could be extended further with advanced tools such as HPQC or iTKO LISA.
- Procure, install, configure, and integrate automated load testing tools such as LoadUI and/or JMeter.

4 Deployment Architecture and Tuning

4.1 Overview

There are many factors that customer should consider as it constructs deployment architecture. The following list is not exhaustive but presents key factors customers should evaluate.

- 24x7 availability requirements under any of the following conditions
 - Application failure
 - Hardware and OS failure
 - Hardware maintenance
 - Operating System upgrade
 - TIBCO Software upgrade
 - Application upgrade
- Is there a maintenance window or not?
- Application performance SLA (throughput and response time)
- Monitoring and management
- TIBCO product runtime characteristics

This chapter will focus more on the TIBCO product runtime characteristics concerning BW 6 to guide customers on configuring and tuning various run-time objects and relevant engine parameters.

4.2 BW 6 Domain and AppSpace Design Consideration

In contrast to BW 5, BW 6 offers more flexibility in how you can organize your applications at run-time via constructs such as Domain, AppSpace, and AppNode. We should consider the following points as it configures BW 6 run-time objects:

- AppSpace
 - When an AppSpace is stopped, all AppNodes in the AppSpace are stopped as well. In other words, all applications deployed in an App Space – even if they are deployed on multiple machines—are stopped as well.
 - Applications in an AppSpace must run the same version of the EAR files. This limitation prevents hot (zero down time) application upgrade. Changes such as simple OS or security patch/upgrade might still work applying them one AppNode at a time in a rolling upgrade manner. However, for changes involving application EAR version or TIBCO software versions, you cannot apply changes/upgrades one AppNode at a time.

- If zero downtime (which also means no maintenance window) is a must, you must have two identical domains to mirror the AppSpaces.
- Some application cannot be deployed as active-active. You cannot mix applications that require active-active and active-passive in the same AppSpace.
- Generalizing the previous point, AppSpace should be created for groups of applications sharing similar run-time characteristics such as on-line vs. batch-oriented, failure recovery requirements, scalability, and/or availability requirements.
- AppNode
 - It's not entirely transparent how much performance impact there is when multiple applications are running in the same AppNode. However, it is likely to be related to the jobs and resources running in an AppNode. See the section **4.4: BW6 JVM Tuning Considerations and its Parameters (AppNode and AppSpace)** below for more in-depth analysis and guidelines on this topic.
- Domain
 - Key consideration in domain design is around how to manage large number of BW 6 applications in an effective manner. Given that most of build and deployment tasks would be automated via CI/CD enabled end-to-end deployment processes, manual human interventions would be minimal in carrying out tasks involving deployment, starting, and stopping of engines. This factor alone renders multi-domain setup more of an overkill as administrators would not need to interact with TEA in handling such tasks. However, we would be supporting a new runtime platform based on BW 6 across a large portfolio of applications, it is anticipating that human interactions would be inevitable most notably during troubleshooting. In such cases, a single domain setup might be more difficult to manage than functionally segregated domains with a fewer number of applications per each domain. At the same time using this setup as a means to provide isolations for a given domain from implications stemming from failures, changes/upgrades being originated from other domains.

4.3 AppSpace and AppNode Tuning Parameters and Guidelines

This section summarizes the key engine parameters available in BW 6 that we should be aware. All of run-time engine parameters are configured either via the *config.ini* or *TRA* file of an engine.

Engine and job tuning properties are configured at the AppNode level in the AppNode's *config.ini* file. These properties can also be specified at the AppSpace level, but the AppNode level property setting will take precedence.

The properties specified in the AppSpace *config.ini* file apply to all AppNodes associated with the AppSpace; however the properties specified in the AppNode *config.ini* file only apply to a specific AppNode and they overwrite any property specified in the AppSpace *config.ini* file.

Engine Threads

Property - `bw.engine.threadCount`

This parameter specifies the BW Engine job thread pool size. This value can be set in the AppSpace *config.ini* file. Engine threads are shared by all the applications deployed on the same AppNode. The value will be applied to all the AppNodes in the AppSpace.

Default Value = 8

Step Count

Property - `bw.engine.stepCount`

This property specifies the number of activities to execute for a job, before BW Engine yields the thread. The value of "-1" will allow the BW Engine to determine the necessary *Step Count* value. This value can be set either in the AppSpace or AppNode *config.ini* file. The value will be applied to all the AppNodes in the AppSpace if set through AppSpace *config.ini*.

Default Value = -1

Flow limit

Property - `bw.application.job.flowlimit.<UsersBWApplicationName> [.<UsersBWApplicationVersion>] [.<UsersBWComponentName>]`

This property specifies the BW Application's Process Starter(s) or Service Binding(s) flow limit value. This value can be set in the AppSpace *config.ini* file. The value will be applied to all the AppNodes in the AppSpace.

This property is helpful when the engine needs to be throttled i.e. the number of incoming requests is overwhelming the engine performance.

Default Value = 8

Page Threshold (*same as BW 5 max jobs*)

Property - `bw.application.job.pageThreshold.<UsersBWApplicationName> [.<UsersBWApplicationVersion>] [.<ComponentName>]`

This property specifies the BW Application's Process Starter(s) or Service Binding(s) job page threshold value. This value can be set in the AppSpace *config.ini* file. The value will be applied to all the AppNodes in the AppSpace.

It is not enforced by the engine unless the *Page Threshold* property is specified for an application. This property is helpful in limiting number of running jobs in memory. To use this property, engine must be configured in datastore or group persistence mode. Once threshold is reached, all new jobs are paged out to the database.

Default Value = 10

Priority

Property - `bw.application.job.priority.<UsersBWApplicationName> [.<UsersBWApplicationVersion>] # [.<UsersBWComponentName>]`

This property specifies the application job's priority. The option for this property is one of low, normal, or high. Engine threads process lower priority jobs only when higher priority jobs are all blocked from continuing. Lower priority jobs are not preempted while in execution. The relevance of using this property makes the most sense in situations where applications with different priorities are co-deployed onto a same AppSpace.

Default Value = normal

4.4 BW6 JVM Tuning Considerations and its Parameters (AppNode and AppSpace)

You can add/replace the JVM tuning parameters by setting “*java.extended.properties*” property in a given *TRA* file. Consider the following example:

```
java.extended.properties=-Xmx1024m -Xms256m -XX:PermSize=64m -
XX:MaxPermSize=128m -XX:-HeapDumpOnOutOfMemoryError
```

If this property were to be specified in the AppSpace *TRA* file, it would be applied to all AppNodes associated with the AppSpace; however the property specified in the AppNode *TRA* file only applies to a specific AppNode and it overwrites the property specified in the AppSpace *TRA* file. You need to restart the process (AppNode and/or AppSpace) for the change to take effect.

One of frequently asked questions on BW 6 tuning revolves around how to optimally set heap size for AppSpace or AppNode. In order to answer this question more effectively, we need to understand the difference between BW 5 and BW 6 in terms of how applications are deployed to the run-time engines. In BW 5, regardless of how complex or simple of a project is, its EAR is deployed to a dedicated BW engine, a JVM. A common practice people adopted on tuning BW 5 engines was to categorize BW applications into a few buckets depending upon factors such as complexity and/or transaction volumetric to be supported, where each bucket is predefined with different JVM settings, most notably the heap size. Such framework was then complemented and refined by following up with performance testing on the actual run-time environments. Through such testing, it was possible to identify for a given application, its operational characteristics such as highest input rate that can be sustained without entering the overload region, and by measuring the actual resource usages such as memory at such input rate, you could identify the maximum heap size a given application must have. Along with this knowledge, considering that input rates from time to time can experience brief spikes, it was also proven prudent to apply throttling mechanisms by using parameters such as *flowlimit* and *max jobs*.

In BW 6, it is getting little bit more complicated when it comes to sizing JVM parameters. This is due to the fact that in BW 6, you no longer have the static one-to-one mapping between an EAR file and a JVM like in BW 5. In fact, you can now deploy multiple applications to an AppSpace, and all AppNodes associated with that AppSpace will run those applications at the same time. This definitely gives us more flexibility in how we organize and map applications into runtime engines. Also, in BW 5, as each application requires a dedicated JVM, for a fairly simple application, its resource utilization as a whole might not have been most optimal as it takes certain amount of overhead in running an instance of JVM. However, this new BW 6 model also poses a challenge for architects and administrators who need to tune the engine parameters particularly, JVM parameters.

First of all, there are no magic formulas here. Second, we should leverage and extend the methodology we have described above in configuring JVM settings for BW 5 to support the new BW 6 model. We suggest the following guidelines as you configure JVM settings for BW 6 engines:

- As you did in BW 5, categorize each application into a few buckets depending upon factors such as complexity and/or transaction volumetric to be supported, where each bucket is predefined with different JVM settings, most notably the heap size.
- Measure the resource utilization characteristics (e.g. memory high watermark as application runs in steady state) of representative applications for each of different buckets via performance testing.
- Organize and plan how you would map a group of applications to be deployed onto various AppSpaces. It's also important to consider that AppSpaces should support applications sharing similar run-time characteristics such as online vs. batch-oriented, non-functional requirements such as LB or FT, recovery requirements, and etc.

- Depending upon how many applications are to run as a group on the same AppSpace infrastructure, you can project the overall memory required at AppSpace level by summing memory high watermark of constituent applications plus/minus alpha where alpha is to be determined empirically. It might not be realistic to get performance and resource characteristics empirically, although desired, for all of BW 6 projects due to its sheer size and scope. In such cases, a technique of extrapolation should be used to estimate memory requirements for a given constituent application belonging to one of predefined application bucket based on actual measurements of representative applications for that bucket.
- Once the memory requirements for all the constituent applications planned to be supported under a given AppSpace are determined, then you can set the initial JVM setting for the AppSpace to be roughly the sum of all applications' memory requirements.
- Run the composite performance testing with all the planned applications fully deployed and running in steady states for a given AppSpace and measure memory or other relevant resource usage patterns. Note the difference between estimated total memory required vs. actual memory observed via testing. As more sample data is collected, we could explore the use of regression to estimate the alpha.
- Given that it can be more un-deterministic in terms of memory usage as multiple applications are run on a JVM, more emphasis should be given in conducting empirical performance testing in validating and tuning JVM settings for BW 6 run-time engines.

5 Other BW 6 Best Practices Relevant to Customers

5.1 Overview

This chapter covers other BW 6 related best practices topics that are of interest to customers.

5.2 Migrating Java Custom Functions from BW 5.x to 6.x Custom XPath Functions

The Migration tool available in BW 6.x provides support for migrating BW 5.x Java custom functions into BW 6.x custom XPath functions. First, you need to migrate a project using the Migration tool through the BW Studio. However, this step alone does not complete the process as you still need to perform a manual step to make the Custom XPath function visible in both BW 6 design and run-time environments. Use the following link to the official TIBCO BW 6 documentation: ***Bindings and Palettes Reference*** covering this topic to accomplish the task:

https://docs.tibco.com/pub/activematrix_businessworks/6.3.1/doc/html/GUID-13795D07-F63C-4B27-8A6B-9E620A74BB16.html

5.3 Sharing Project Artifacts through a Shared Module

A shared module is the smallest unit of resources that is named, versioned, and packaged as part of an application and can be used by other modules that are part of the same application. Shared modules export their functionality (process, shared resources, and schema namespaces) to application modules. When creating a new module, select a shared module if the business logic needs to be shared across multiple applications. Shared modules can also be used if XML collisions exist. Good starting points for using Shared Modules would be on managing common processes and frameworks around logging and exception handling and Java Custom Functions. For more info on how to use Shared Modules, please reference the BW 6 product documentation set. BW Application Development Guide would be an excellent resource.

As you plan to use Shared Modules in your project, consider the following points:

- You cannot deploy more than one copy of a Shared Module to an AppNode. This means that if multiple applications are packaged with respective Shared Modules of their own, they cannot be deployed to the same AppNode. This calls for upfront planning across portfolio of projects on how to organize and structure what type of processing logic and supporting resources to be managed via Shared Modules.
- You cannot have dependency between Shared Modules such as calling a sub-process or using artifacts from one Shared Module to another.

5.4 Using Policy in BW 6

Policies supported by BW 6 can be broadly categorized into two types; HTTP security and SOAP Security. For more details on how to manage and apply policy resources, you should reference the BW 6 application development guide under the topics such as “Managing Policy Resources”, “HTTP Security”, and/or “SOAP Security”.

6 Frequently asked BW 6 Questions

6.1 Overview

This chapter presents series of questions on BW 6 and related products that has raised while TIBCO PSG was engaging on its migration analysis. Questions that were already answered by various chapters on this document have been skipped.

6.2 Questions and Answers

- Product Management recommends using DB+EMS for supporting BW6 runtime. Is that recommendation also applicable to supporting Adapter runtime? In typical BW 6 runtime setup, you have a BW domain and for adapters you set up a separate adapter domain. For supporting adapter domain, should we also use DB+EMS or does it only support AS?

[Answer] Adapter currently supports only AS.

- Is BE supported for TEA server? How about standalone AS implementation? Does it run under TEA?

[Answer] BE support for TEA is planned for release in first half of year 2016. AS support for TEA is currently not being planned as customer requests for this support is minimal.

- When working with Adapter agent, it only supports command-line mode. Can we deploy/undeploy Adapter modules using the GUI option in TEA? Is this feature being planned to be supported soon?

[Answer] TIBCO is not currently planning for enhancing TEA UI concerning this feature but will continue to monitor if there is sufficient traction. Number of customer requests for this support has been minimal so far.

- For projects involving Adapters or Plugin, how are customers packaging the solution (Adapters and Process in the same archive, separate archive, or other arrangements) and how do they deploy such solutions?

[Answer] In a solution requiring BW and Adapters, you would need to deploy separately in BW domain and Adapter domain. Even though deployment is separate, you do not necessarily need to keep separate archives for each. We recommend that you create the archives relative to your business requirements.

- Is there any Siebel adapter or Plugin that is supported for BW6?

[Answer] Siebel Adapter is supported as a 'non-native' adapter. This means that Siebel is supported in a limited capacity in BW 6 and works only through the usage of Adapter Framework. To elaborate, you can create an adapter project in TIBCO Designer, validate the project and then migrate the project into TIBCO Business Studio. After migrating, the Eclipse GUI provided by TIBCO Business Studio is available for those non-native adapters. You can modify the process configuration, define new processes and add activities from the Adapter Palette panel.

However, you cannot modify adapter configurations for the migrated project, add new adapter configurations or adapter services directly in TIBCO Business Studio. If you want to modify an adapter configuration for the migrated project, you need to modify the instance in TIBCO Designer, save it and migrate the project into TIBCO Business Studio again.

- Is there a way to have application specific logs?

[Answer] The 'Log' activity uses the Java Logback API. For custom application logging, the AppNode's logback.xml file can be customized with additional appenders. The activity value for 'LoggerName' should refer to the custom appender.

- Can we actually configure BW 6 AppSpace and AppNode in FT?

[Answer] Yes. For detailed steps on how to do this, refer to the Appendix A at the end of this document.

APPENDIX A. BW 6.x Fault Tolerance and Load Balancing

Introduction

This is a chapter to introduce the way BW 6 manages fault tolerance and load balancing.

Managed Fault Tolerance

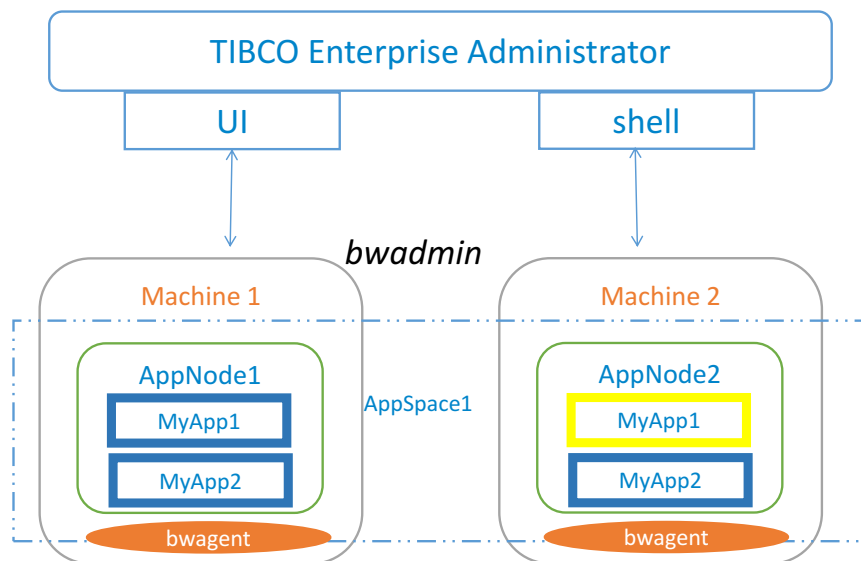


Figure 23 Managed Fault Tolerance

- In managed fault tolerance, when an AppNode fails, the application on another AppNode takes over automatically.
- The AppNodes in an AppSpace are aware of each other's existence.
- Engine persistence mode defines whether engines(appnode) located on one or more physical machines
- Collaborate with each other or work independently. Set this to "group" for Managed FT

Managed Fault Tolerance: Single AppNode (Active-Passive)

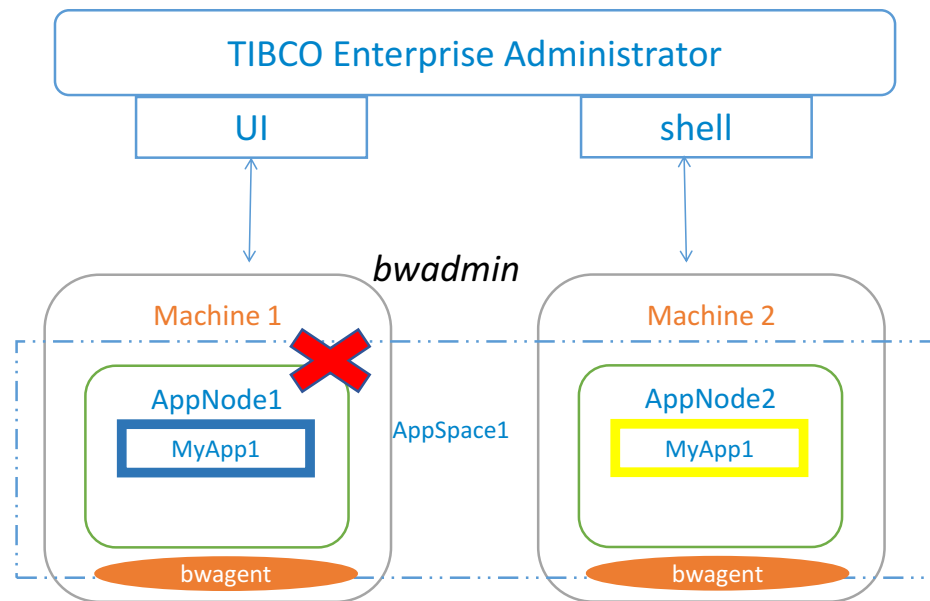


Figure 24 Managed Fault Tolerance: Single AppNode

- The incoming requests are only processed by an AppNode where the application is in an active state.
- On failure of an AppNode that has the application in an active state will automatically enable the application in another AppNode

Managed Fault Tolerance: Multiple AppNode (Active-Active)

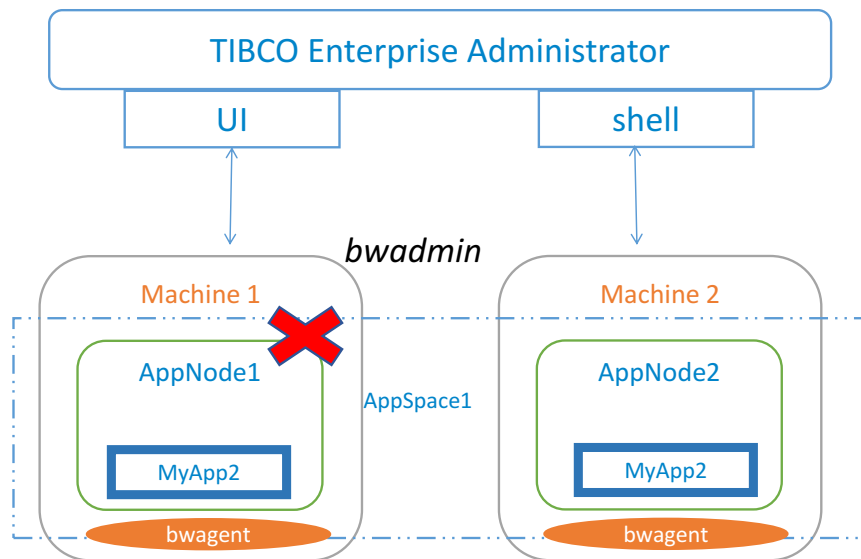


Figure 25 Managed Fault Tolerance Multiple AppNode

- The incoming requests can be processed by any AppNode since the application is active in all AppNodes.
- On failure of an AppNode, other AppNodes will continue to process new requests
- Effectively this does both load balancing and Fault Tolerance with the same config
- Ensure no sequencing issues when load-balancing requests..

Non Managed Fault Tolerance

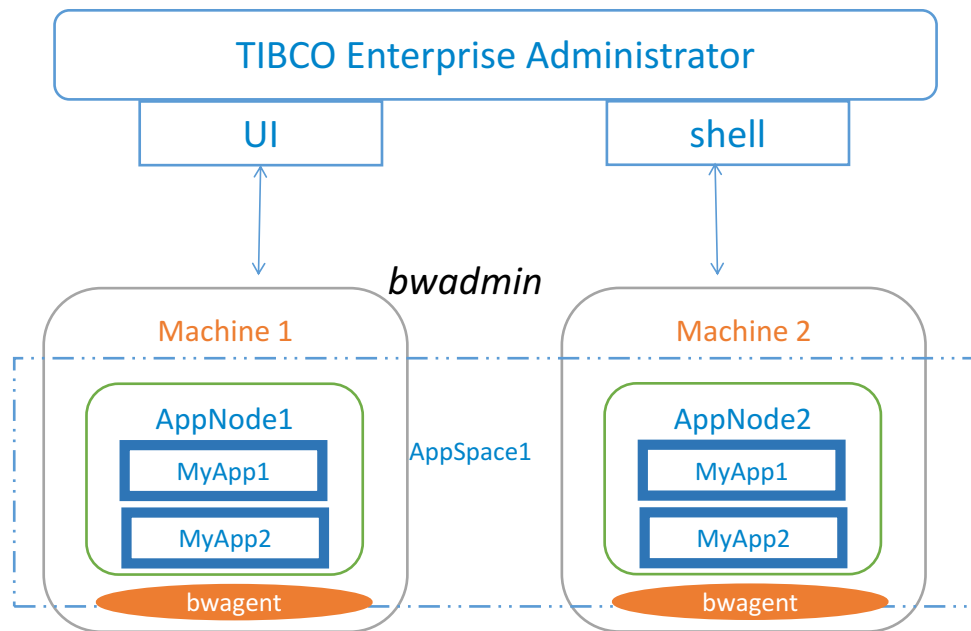


Figure 26 Non Managed Fault Tolerance

- In non managed fault tolerance, when an AppNode fails, the application on another AppNode DOES NOT takes over automatically.
- The AppNodes in an AppSpace are NOT aware of each other's existence.
- Engine(appnode) persistence mode set this to "database" for Managed FT
- Each appnode should have a unique database config
- All applications are active on all the nodes (No Active-Passive)

Configuration Background

We can only configure FT at the BW process level. However Fault Tolerance on BW6 works at AppNodes because when an AppNode fails, the application on another AppNode can take over automatically. AppNodes in an AppSpace are aware of each other's existence and the engines collaborate to provide FT. This can be configured using Activation Mode option set as Single AppNode in the process. Recall that Activation Mode setting is done at design-time through Business Studio.

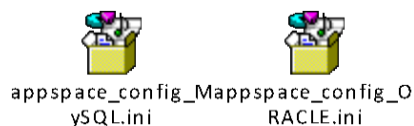
Adding multiple bwagents to a TEA admin

1. Machine 1:
 - a. install TEA admin
 - b. Start tea admin (change http port if required & start it)
 - c. install BW
 - d. Register it to the tea admin (registerteaadmin) and start the bwagent
2. Machine 2:
 - a. Install BW
 - b. Start bwagent
3. Machine 1:
 - a. Edit the bwagent.ini file and change the discovery URL server name to machine2 and save it (keep the default port 5050)
 - b. Set discoveryURL=tcp://<host1>:5050;<host2>:5050
 - c. Restart bwagent once
 - d. Create a domain D1
 - e. Add machine2 to this domain (registeragent domain <machine2 servrename>)
 - f. Create an appspace AS1
 - g. Add machine 2 to this appspace (registeragent appspace <machine2 servrename>)
 - h. Create 2 appnodes APP1_PR & APP2_PR
4. Machine 2:
 - a. Run show domains. It will list the domain D1 created in machine 1.
 - b. cd to the domain and then to the AppSpace.
 - c. Create 2 AppNodes APP1_SR & APP2_SR.

Check in TEA admin (of machine1). It will show the domain D1 distributed across machines M1 & M2 with one AppSpace AS1 and two nodes each.

FT Configuration Steps are as Follows for BW 6.x

1. Select you BW process (.bwp) in BusinessStudio and change the application activation modes to Single AppNode and save.
2. Create an ear.
3. Copy appspace_config.ini_template (located in <TIBCO_HOME>/bw/6.x/config/) to a temporary location. Rename it to config.ini.
4. Open the file and change the engine persistence mode to group.
5. Provide information in the EMS and Database related fields. Save.
6. Sample config.ini file is attached for MySql and Oracle.



7. Make sure you AppNode/AppSpace is not running.
8. Once you setup your database you need to run *oracle_create.sql* (From <TIBCO_HOME>/bw/6.x/config/dbscripts/admin/<<DB>>) and *create.sql* (From <TIBCO_HOME>/bw/6.x/config/dbscripts/engine/<<DB>>)
9. Push the changes to AppSpace by executing the command from bwadmin.exe:


```
>> config -d <DomainName> -a <AppSpaceName> -cf temporaryLocation/config.ini
```
10. Upload and deploy ear from step 2.
11. Both applications report the status of "Running" but if you check the node logs you will see that one is active and processing the jobs while the other is in standby mode.
12. To test if standby engine becomes active in case of failure, you can terminate the JVM. In your case it would be to end the process for the JVM.
13. After the AppNode has been terminated, you will see that application deployed in the standby AppNode will start to process the requests.
14. Attached log below shows the fail-over in action where the logger was configured to log the AppNode name on which messages were being processed.



logs.txt

Note: The same setup can be applied at AppNode level. That can be achieved by using appnode_config.ini file. It should be noted that sections for JMS and JDBC are not available in the appnode config file so you have to copy the content from appspace config file and paste it into appnode_config.ini file and change the values as needed.

The bwagent Configuration (for 6.x)

1. Use bw.agent.technology.type=DBEMS
2. Set two bwagents on two different servers (Machine_1 and Machine_2) with
 - a. bw.agent.technology.as.role=server on both bwagent
 - b. bw.agent.network.name=Network1 for both bwagents
 - c. bw.agent.memberName=Agent1 for Machine_1 and bw.agent.memberName=Agent2 for Machine_2
3. Create one Domain.
4. Create one AppSpace.
5. Create Node1 on Machine_1 and Node2 on Machine_2.
6. Then follow previous suggestions to deploy and run the applications in FT.