



# TIBCO BusinessWorks™ Container Edition

## Samples

Version 2.10.0 | December 2024



# Contents

---

<b>Contents</b> .....	<b>2</b>
<b>Accessing Samples</b> .....	<b>8</b>
<b>Guided User Experience in TIBCO Business Studio for BusinessWorks</b> .....	<b>10</b>
<b>Creating a TIBCO BusinessWorks Container Edition Project from Template</b>	<b>13</b>
<b>Samples for Cloud Foundry</b> .....	<b>16</b>
Using REST to Manage Books for a Bookstore .....	16
Testing in TIBCO Business Studio for BusinessWorks .....	17
Understanding the Configuration .....	21
Testing the REST BookStore Sample .....	24
Generating an Application Archive File .....	28
Deploying the REST BookStore Application .....	28
Testing the REST Service .....	30
Troubleshooting the REST Bookstore Sample .....	31
REST BookStore Consumption .....	32
Consuming a BookStore REST Endpoint .....	32
Creating a REST Reference Binding .....	34
Soap with External Binding .....	35
Testing Locally .....	35
Understanding the Configuration .....	36
Testing the SOAP Sample .....	38
Deploying the Application .....	38
StockQuote Soap Client .....	39
Understanding the Configuration .....	40
Implementing a JMS Service that Returns Information Based on Zip Codes .....	41
Testing Locally .....	41

Understanding the Configuration .....	42
Testing the Sample .....	43
Deploying the Sample .....	44
Resilience4j .....	45
Testing in TIBCO Business Studio for BusinessWorks .....	45
Understanding the Configuration .....	47
Testing the Resilience4j Sample .....	48
Generating an Application Archive File .....	49
Deploying the Resilience4j Application .....	50
Service Discovery .....	51
Testing in TIBCO Business Studio for BusinessWorks .....	52
Understanding the Configuration .....	53
Testing the Service Registry Sample .....	55
Generating Application Archive Files .....	56
Deploying the Service Application .....	56
Deploying the Client Application .....	58
HTTP Request Response .....	59
Testing Locally .....	59
Understanding the Configuration .....	60
Testing the HTTP Request Response Sample .....	61
Generating an Application Archive File .....	62
Deploying the HTTP Request Response Sample .....	62
JDBC Basic .....	63
Testing in TIBCO Business Studio for BusinessWorks .....	63
Understanding the Configuration .....	65
Testing the JDBC Basic Sample .....	67
Generating an Application Archive File .....	72
Deploying the JDBC Basic Application .....	72
JMS Basic .....	73
Testing Locally .....	74
Understanding the Configuration .....	75
Testing the JMS Basic Sample .....	76

Generating an Application Archive File .....	78
Deploying the JMS Basic Application .....	78
Sending and Receiving Messages Using EMS-SSL .....	79
Testing in TIBCO Business Studio for BusinessWorks .....	79
Understanding the Configuration .....	81
Testing the EMS Secure Sample .....	82
Generating an Application Archive File .....	84
Deploying the EMS Secure Application .....	84
JMS Basic With Spring Cloud Config Server .....	85
Testing in TIBCO Business Studio for BusinessWorks .....	86
Understanding the Configuration .....	87
Testing the JMS Basic Sample with Spring Cloud Config Server .....	88
Generating an Application Archive File .....	91
Deploying the Application .....	91
Collecting Process Instance, Activity Instance, and Transition Statistics .....	92
<b>Samples for Docker .....</b>	<b>100</b>
REST Bookstore .....	100
Testing in TIBCO Business Studio for BusinessWorks .....	101
Understanding the Configuration .....	105
Testing the REST Sample .....	108
Generating an Application Archive File .....	109
Building the Application Image .....	109
Testing the Application Locally in a Docker Setup .....	110
Testing the REST Service .....	111
Test Your Application in the Kubernetes Setup on the Google Cloud Platform .....	112
REST BookStore Consumption .....	114
Consuming a BookStore REST Endpoint in TIBCO Business Studio for BusinessWorks .....	114
Creating a REST Reference Binding .....	116
Implementing a JMS Service that Returns Information Based on Zip Codes .....	117
Understanding the Configuration .....	118

Testing the Sample .....	120
Generating an Application Archive File .....	121
Test your Application Locally in a Docker Setup .....	121
Test your Application in the Kubernetes Setup on the Google Cloud Platform .....	122
Resilience4j .....	123
Testing in TIBCO Business Studio for BusinessWorks .....	124
Understanding the Configuration .....	125
Testing the Resilience4j Sample .....	126
Generating an Application Archive File .....	127
Building the Application Image .....	128
Testing the Application Locally in a Docker Setup .....	128
Test Your Application in the Kubernetes Setup on the Google Cloud Platform .....	129
Service Discovery .....	131
Testing in TIBCO Business Studio for BusinessWorks .....	131
Understanding the Configuration .....	133
Testing the Service Discovery Sample .....	134
Generating an Application Archive File .....	136
Building the Application Image for the Service Application .....	136
Building the Application Image for the Client Application .....	137
Testing the Service and Client Applications Locally in a Docker Setup .....	137
Test Your Application in the Kubernetes Setup on the Google Cloud Platform .....	139
HTTP Request Response .....	141
Testing in TIBCO Business Studio for BusinessWorks .....	141
Understanding the Configuration .....	142
Testing the HTTP Request Response Sample .....	143
Generating an Application Archive File .....	149
Building an Application Image .....	149
Test your Application Locally in a Docker Setup .....	149
Test your Application in the Kubernetes Setup on the Google Cloud Platform .....	150
JDBC Basic .....	152
Testing in TIBCO Business Studio for BusinessWorks .....	152
Understanding the Configuration .....	154

Testing the JDBC Basic Sample .....	155
Generating an Application Archive File .....	156
Testing the Application Locally in a Docker Setup .....	156
Testing the Application in the Kubernetes Setup on the Google Cloud Platform .....	157
JMS Basic .....	159
Testing in TIBCO Business Studio for BusinessWorks .....	159
Understanding the Configuration .....	160
Testing the JMS Basic Sample .....	161
Generating an Application Archive File .....	162
Building an Application Image .....	162
Test your Application Locally in a Docker Setup .....	163
Test Your Application in a Kubernetes Setup on the Google Cloud Platform .....	164
JMS Basic with Consul .....	165
Testing in TIBCO Business Studio for BusinessWorks .....	166
Understanding the Configuration .....	167
Testing the Sample .....	168
Generating an Application Archive File .....	169
Setting up the Consul Server .....	169
Test your Application Locally in a Docker Setup .....	171
Test your Application in the Kubernetes Setup on the Google Cloud Platform .....	172
Sending and Receiving Messages Using EMS-SSL .....	174
Testing in TIBCO Business Studio for BusinessWorks .....	174
Understanding the Configuration .....	175
Testing the Sample .....	177
Generating an Application Archive File .....	178
Creating the TIBCO BusinessWorks Container Edition Base Docker Image for Linux Containers .....	179
Test your Application Locally in a Docker Setup .....	183
Test your Application in the Kubernetes Setup on the Google Cloud Platform .....	184
SOAP over HTTP .....	186
Testing in TIBCO Business Studio for BusinessWorks .....	186
Understanding Configuration .....	188

Testing the Sample .....	189
Generating an Application Archive File .....	190
Building an Application Image .....	190
Testing the Application Locally in a Docker Setup .....	191
SOAP Client .....	191
Testing Your Application in Kubernetes Setup on the Google Cloud Platform .....	193
SOAP Client .....	194
Collecting Process Instance, Activity Instance, and Transition Statistics .....	195
<b>Unit Testing .....</b>	<b>204</b>
Maven Test Sample of Main Process .....	205
Maven Sample of Unit Test Demo Project .....	209
Maven Sample of Shared Module Unit Test Cases .....	214
<b>TIBCO Documentation and Support Services .....</b>	<b>217</b>
<b>Legal and Third-Party Notices .....</b>	<b>219</b>

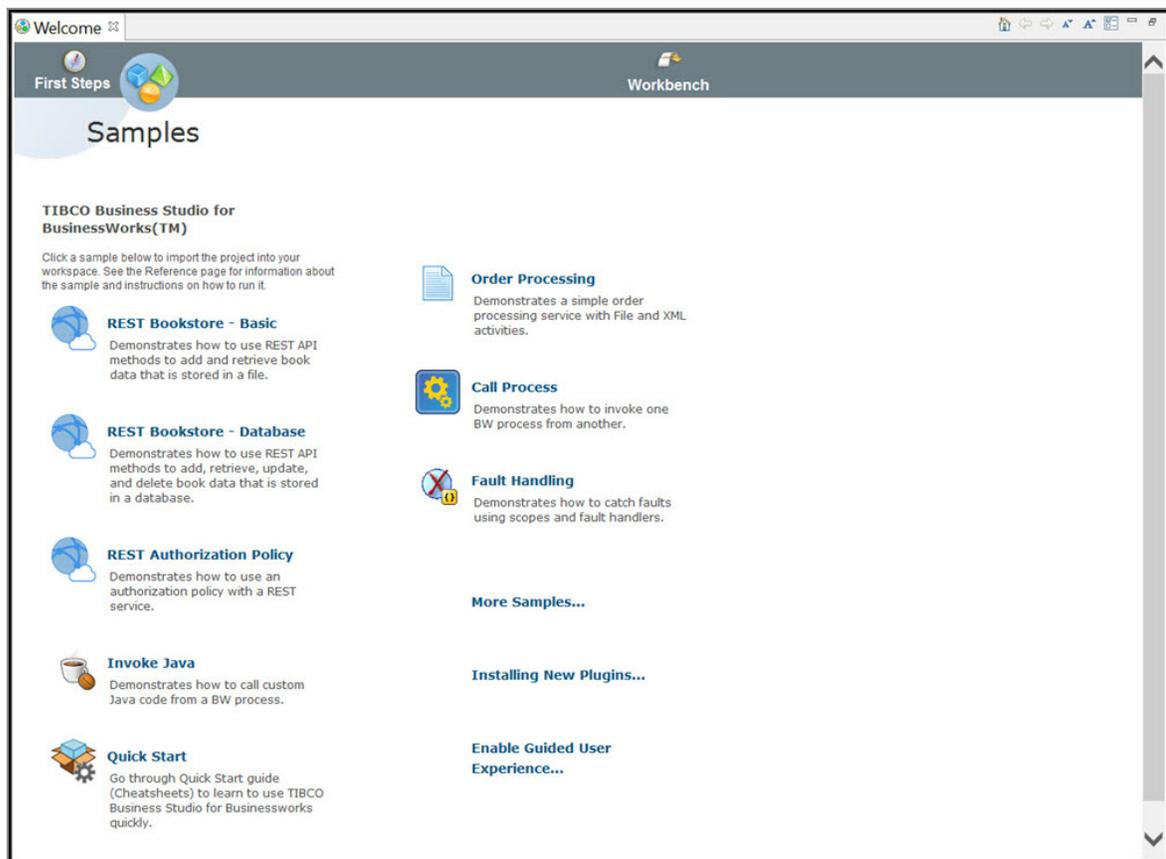
# Accessing Samples

You can access the samples in TIBCO Business Studio™ for BusinessWorks™ from the **File Explorer** view.

## Procedure

1. Launch TIBCO Business Studio for BusinessWorks from `$TIBCO_HOME\studio\<version>\eclipse\TIBCOBusinessStudio.exe`.  
By default, TIBCO Business Studio for BusinessWorks opens the Design perspective.
2. Click **Help > BusinessWorks Container Edition Samples**.

The **Welcome** screen is displayed with various available samples, for example, REST Bookstore - Basic, REST Bookstore - Database, Order Processing, and so on.

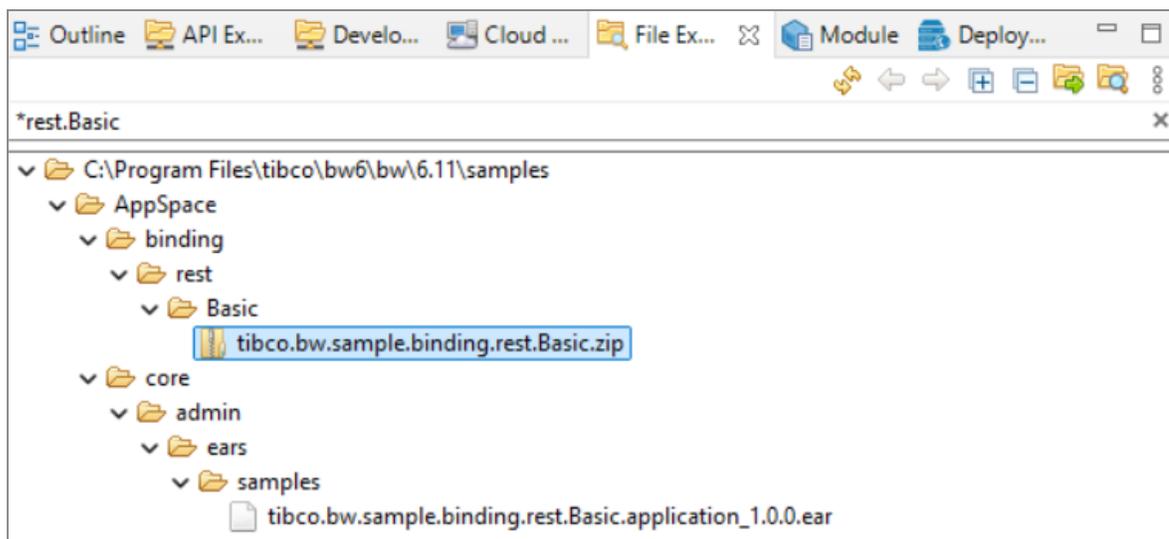


3. Select the required sample.

The **File Explorer** view displays a top level listing of all the samples. You can also search for the samples by using the "search" field in the **File Explorer** view.

**i Note:** If the workspace is from an older version, you need to locate the samples manually using the **Open Directory to Browse** button available on the **File Explorer** view. TIBCO Business Studio for BusinessWorks stores the samples at `tibco-home/bwce/2.5/samples`.

4. Navigate to the directory containing the sample you want to access and do the following:

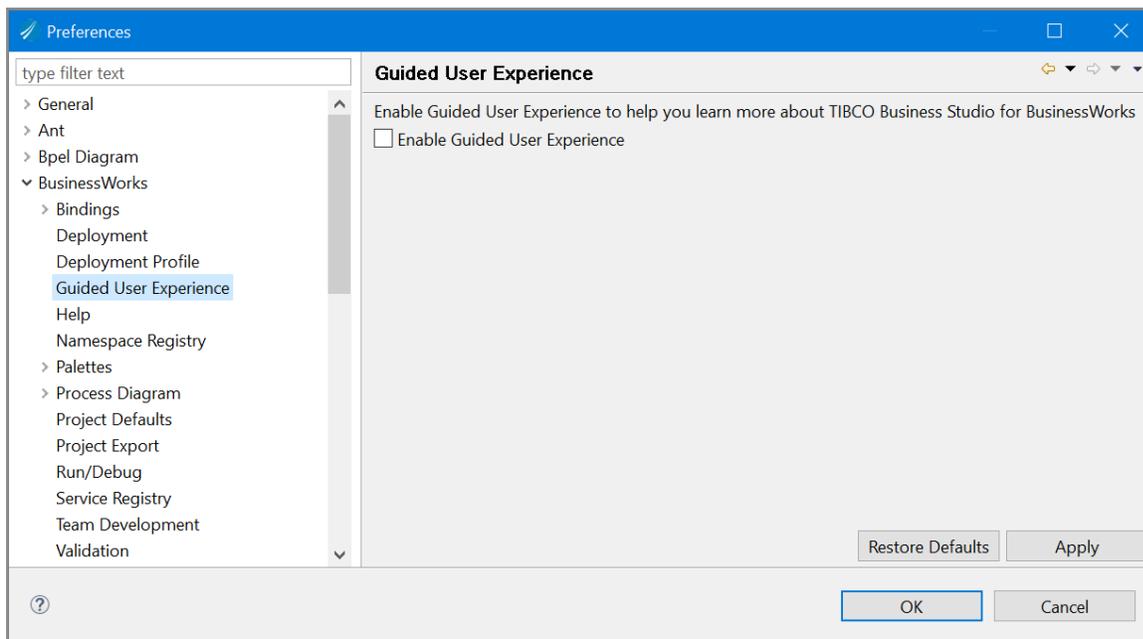


- Double-click the archive file, `<sample_name>.zip`, to import the sample application to your workspace. For example, double-click `tibco.bwce.sample.binding.rest.Bookstore.zip` to import the application and the application module to your workspace. Alternatively, you can right-click the required project folder and click **Import Selected Projects** option to import the required application.
- Click `readme.link` to open the sample Help page in the **Reference - online** view. For example, click **cloudfoundry > bindings > rest > Bookstore > readme.link** to access the help pages for the Bookstore Sample application. You can access the help pages of all samples by following the same process.

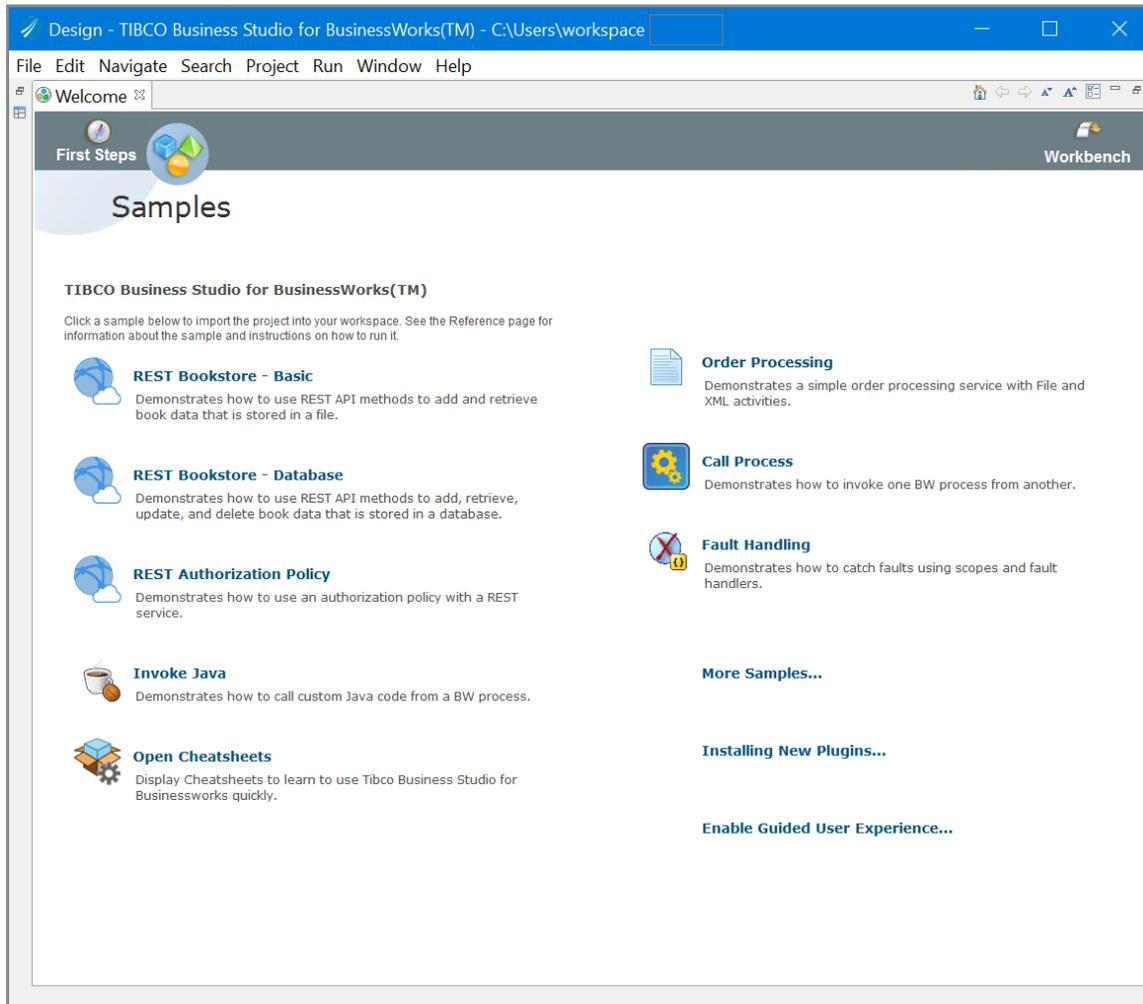
# Guided User Experience in TIBCO Business Studio for BusinessWorks

TIBCO Business Studio for BusinessWorks introduces a new feature, guided user experience, wherein you can see notifications, pop-up messages, tooltips, or suggestions while performing an operation.

To enable the guided user experience option, navigate to **Window > Preferences > BusinessWorks > Guided User Experience** and select the **Enable Guided User Experience** check box on the Preferences window.

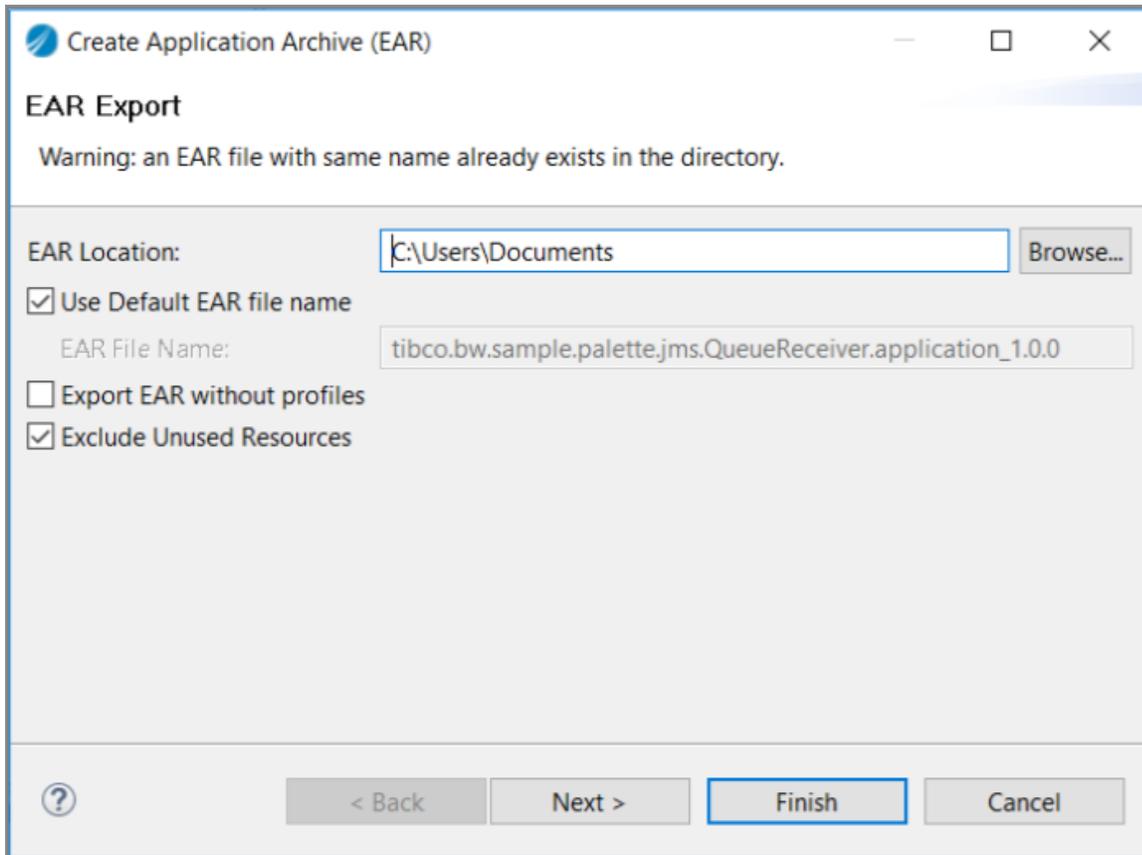


Or click **Enable Guided User Experience...** on the Welcome screen and select the **Enable Guided User Experience** check box on the Preferences window.



For example, while developing an application, when you click **Generating EAR**, the **Create Application Archive (EAR)** window is displayed. If you have already enabled guided user

experience, the **Exclude Unused Resources** check box is visible and selected by default.



# Creating a TIBCO BusinessWorks Container Edition Project from Template

---

You can use an existing TIBCO BusinessWorks Container Edition project template to create a new project.

## Before you begin

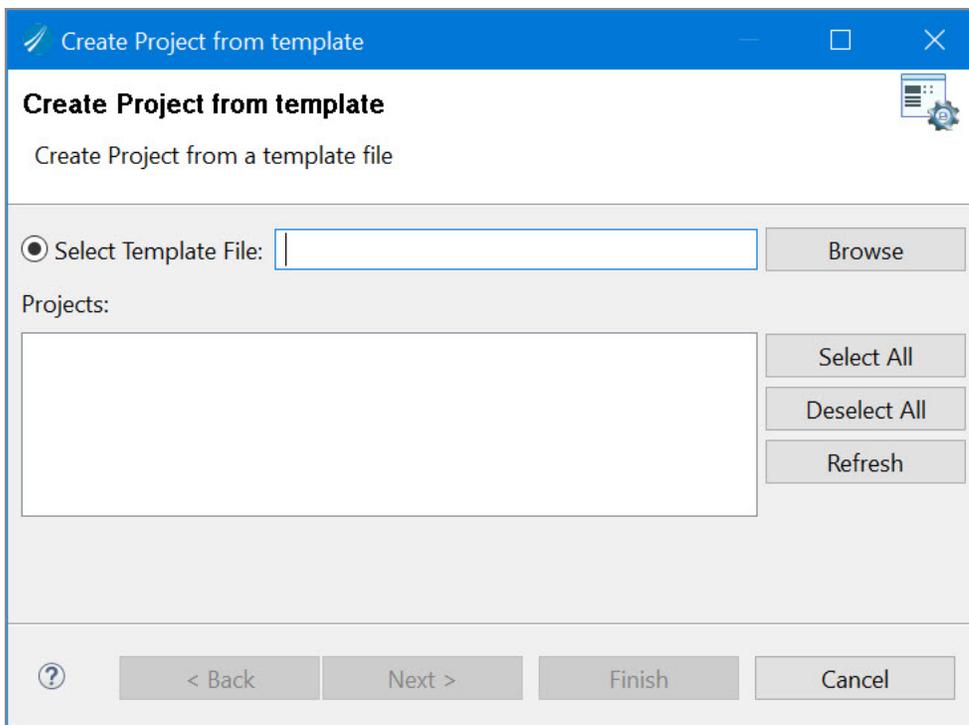
TIBCO Enterprise Message Service must be running.

## Procedure

1. Navigate to **File > New > Project > BusinessWorks** and click the **BusinessWorks Project from Template** option. Click **Next**.

Or click the  icon on the toolbar.

The *Create Project from template* dialog is displayed.



2. In the Select Template File field, provide the template file location from the samples directory, **bwTemplate > tibco.bw.sample.Template.TemplateSample.1.0.0.bwtemplate**. The project in the template is displayed.
3. To see information about template, click **Next**.
4. To rename the project, click **Next**. In the Existing Name column, select the project and provide the name in the New Name column.
5. Click **Finish**. On successful project creation, you can see the template project in the Project Explorer pane.
6. In the Project Explorer view, expand the **tibco.bw.sample.Template.TemplateSample** project.
7. Verify your TIBCO Enterprise Message Service connection.
  - a. Fully expand the **Resources** directory.
  - b. Double-click **JMSConnectionResource.jmsConnResource**.
  - c. In the Basic Configuration pane, click the **Test Connection** button to verify the connection.
8. Fully expand the **Processes** directory and double-click **Process.bwp**.
9. Click **Run > Debug Configurations**.
10. At the left of the Debug Configurations wizard, expand **BusinessWorks Application** and select **BWApplication**.
11. In the **Applications** tab, click the **Deselect All** button if you have multiple applications. Select the checkbox next to the **tibco.bw.sample.Template.TemplateSample.application** option.
12. Click **Debug**. This runs the sample in the Debug mode.
13. Click the  (Terminate) icon to stop the process.

## Result

The Console displays messages similar to the following:

```
13:11:42.211 INFO [bwEngThread:In-Memory Process Worker-1]
com.tibco.bw.palette.generalactivities.Log.tibco.bw.sample.Template.TemplateSample.Log - Sending JMS Message
```

```
13:11:42.243 INFO [bwEngThread:In-Memory Process Worker-1]
com.tibco.bw.palette.generalactivities.Log.tibco.bw.sample.Template.TemplateSample.Log1 - JMS Message sent successfully
```

```
13:11:42.337 INFO [bwEngThread:In-Memory Process Worker-2]
com.tibco.bw.palette.generalactivities.Log.tibco.bw.sample.Template.TemplateSample.Log2 - Message received JMS Message sent
```

## Understanding the Configuration

The JMS Send Message activity sends the messages with the message style set to Queue. The Get JMS Queue Message activity receives the JMS messages from a Queue.

# Samples for Cloud Foundry

---

The following samples are available for your Cloud Foundry environment:

- [REST BookStore](#)
- [REST BookStore Consumption](#)
- [SOAP with External Binding](#)
- [Implementing a JMS Service that Returns Information Based on Zip Codes](#)
- [Resilience4j](#)
- [Service Discovery](#)
- [HTTP Request Response](#)
- [JDBC Basic](#)
- [JMS Basic](#)
- [Sending and receiving Messages Using EMS-SSL](#)
- [JMS Basic with Spring Cloud Config Server](#)
- [Collecting Process Instance, Activity Instance, and Transition Statistics](#)

## Using REST to Manage Books for a Bookstore

This sample shows how to use the RESTful service to add, delete, update, and retrieve books from a bookstore.

The sample uses the following HTTP methods for the Books REST resource:

- POST: Add books to the bookstore.
- GET: Get books from the bookstore.
- PUT: Update books in the bookstore.
- DELETE: Delete books from the bookstore.

# Testing in TIBCO Business Studio for BusinessWorks

## Before you begin

- Access to a locally running PostgreSQL database.
- A web browser.

## Procedure

1. From the **File Explorer**, navigate to the **samples > Container > cloudfoundry > binding > rest > Bookstore** folder and double-click **tibco.bwce.sample.binding.rest.BookStore**.
2. From the **Project Explorer** expand the **tibco.bwce.sample.binding.rest.BookStore.application** project.
3. Expand **tibco.bwce.sample.binding.rest.Bookstore.application > Package Unit** folder. Provide valid values for the application properties. Provide a valid dbUserName, dbPassword and dbURL to connect to your locally running PostgreSQL database.
4. Set the **ApplicationProfile** to **local**. For more information, see [Setting the Default Application Profile](#).
5. Verify your JDBC connection.
  - a. Expand the **Resources** directory.
  - b. Double-click **JDBCConnectionResource.jdbcResource**.
  - c. Click the **Test Connection** button to verify the connection.
6. Click **File > Save**.
7. From the **Project Explorer** expand the **Processes** folder and the **tibco.bwce.sample.binding.rest.BookStore** package within that folder.
8. Click **Run > Debug**.
9. From the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
10. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications.
11. Select the checkbox next to **tibco.bwce.sample.binding.rest.BookStore**.

12. Click **Debug**.

The sample now runs in the debug mode.

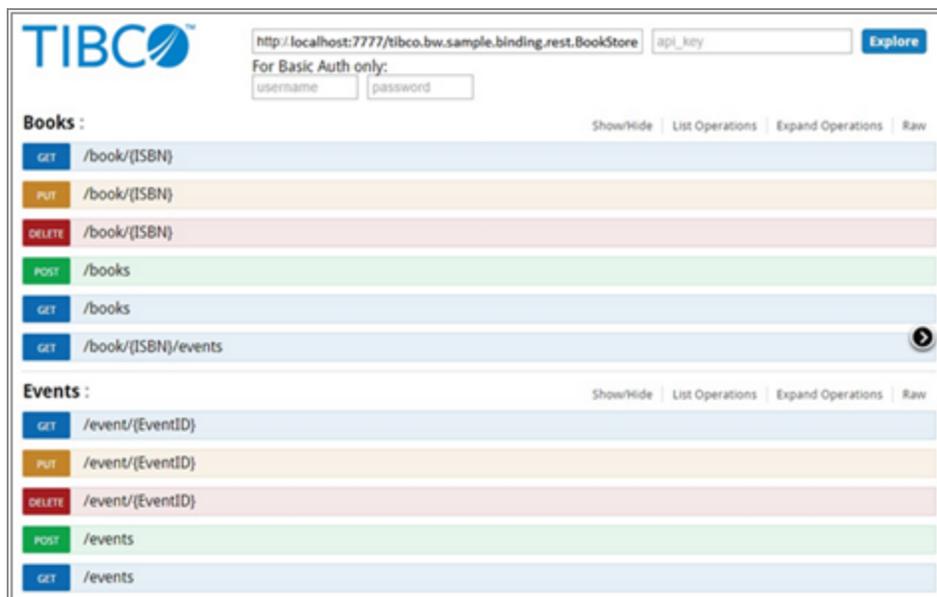
The console window shows engine messages similar to:

```
Started BW Application [tibco.bwce.sample.binding.rest.BookStore.
[tibco.bwce.sample.binding.rest.BookStore.application:1.0].
The OSGI command to list REST and Swagger URLs is
lrestdoc, which lists the discovery URL:[Application
tibco.bwce.sample.binding.rest.BookStore.application [Discovery
Url]:http://localhost:7777/tibco.bwce.sample.binding.rest.BookStore
.application
```

## 13. Launch the browser and open:

```
http://localhost:7777/tibco.bwce.sample.binding.rest.BookStore.appl
ication
```

The following page is displayed



## 14. Click any of the listed operations GET, PUT, PUT, or DELETE.

15. Click the Terminate  icon when you have completed using the listed operations.

## Result

The web page lists the following operations for Books and Events:

### Books

- POST books
- GET books
- GET book by ISBN
- PUT book by ISBN
- DELETE book by ISBN

### Events

- POST Events
- GET Events
- GET Event by EventID
- PUT Event by EventID
- DELETE Event by EventID

**GET books** returns an output similar to the following:

```
{
  "Book": [
    {
      "isbn": "0061122416",
      "name": "The Alchemist",
      "description": "Every few decades a book is published that changes
the lives of its readers forever. The Alchemist is such a book",
      "authorName": "Paul Coelho",
      "releaseDate": "2006-04-25",
      "vintage": true,
      "signed": true,
      "price": 11.9
    },
    {
      "isbn": "0071450149",
      "name": "The Power to Predict",
      "description": "How Real Time Businesses Anticipate Customer
Needs, Create Opportunities, and Beat the Competition",
    }
  ]
}
```

```

    "authorName": "Vivek Ranadive",
    "releaseDate": "2006-01-26",
    "vintage": false,
    "signed": true,
    "price": 15.999
  }
]
}

```

**GET books** by ISBN returns an output similar to the following for the ISBN 0061122416:

```

{
  "isbn": "0061122416",
  "name": "The Alchemist",
  "description": "Every few decades a book is published that changes
the lives of its readers forever. The Alchemist is such a book",
  "authorName": "Paul Coelho",
  "releaseDate": "2006-04-25",
  "vintage": true,
  "signed": true,
  "price": 11.9
}

```

The books.log file is generated with the following information:

```

POST Books----->{"Book":[{"isbn":"1451648537","name":"Steve
Jobs","description":"Biography of Apple Co-Founder Steve
Jobs","authorName":"Walter Isaacson","releaseDate":"2012-10-
24","vintage":false,"signed":false,"price":21},
{"isbn":"0385537859","name":"Inferno","description":"Robert Langdon
returns in Dan Brown's latest fast paced action
thriller","authorName":"Dan Brown","releaseDate":"2013-05-
14","vintage":false,"signed":true,"price":14.09},
{"isbn":"0399103421","name":"The Godfather","description":"The Godfather
is an epic story of a New York's top mafia family, loyalty, and how men
of honor live in their own world, and die by their own
laws.","authorName":"Mario Puzo","releaseDate":"1969-03-
10","vintage":true,"signed":true,"price":50}]}

```

```

*****

```

```

GET Books----->{"Book":[{"isbn":"1451648537","name":"Steve
Jobs","description":"Biography of Apple Co-Founder Steve
Jobs","authorName":"Walter Isaacson","releaseDate":"2012-10-

```

```
24+05:30","vintage":false,"signed":false,"price":21},
{"isbn":"0385537859","name":"Inferno","description":"Robert Langdon
returns in Dan Brown's latest fast paced action
thriller","authorName":"Dan Brown","releaseDate":"2013-05-
14+05:30","vintage":false,"signed":true,"price":14.09},
{"isbn":"0399103421","name":"The Godfather","description":"The Godfather
is an epic story of a New York's top mafia family, loyalty, and how men
of honor live in their own world, and die by their own
laws.,"authorName":"Mario Puzo","releaseDate":"1969-03-
10+05:30","vintage":true,"signed":true,"price":50}}]}
```

```
*****
```

```
GET Book By ISBN----->{"isbn":"1451648537","name":"Steve
Jobs","description":"Biography of Apple Co-Founder Steve
Jobs","authorName":"Walter Isaacson","releaseDate":"2012-10-
24+05:30","vintage":false,"signed":false,"price":21}
```

```
*****
```

```
DELETE Book By ISBN----->"Deleted book with ISBN - 1451648537"
```

```
*****
```

```
GET Events By ISBN---->{}
```

```
*****
```

## Understanding the Configuration

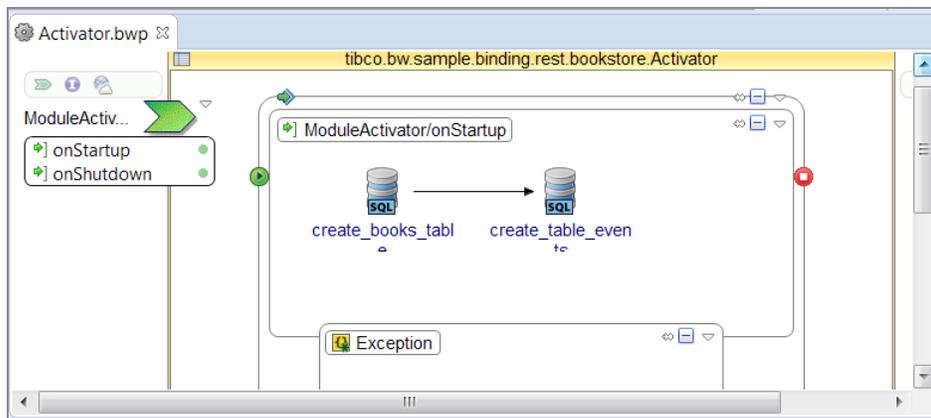
The project contains three processes (Activator, Books, and Events) and two subprocesses (BooksDB and EventsDB).

- **Activator Process** (Activator.bwp)

The BusinessWorks Activator process is used to perform pre-processing and post-processing tasks when the application is started and stopped respectively.

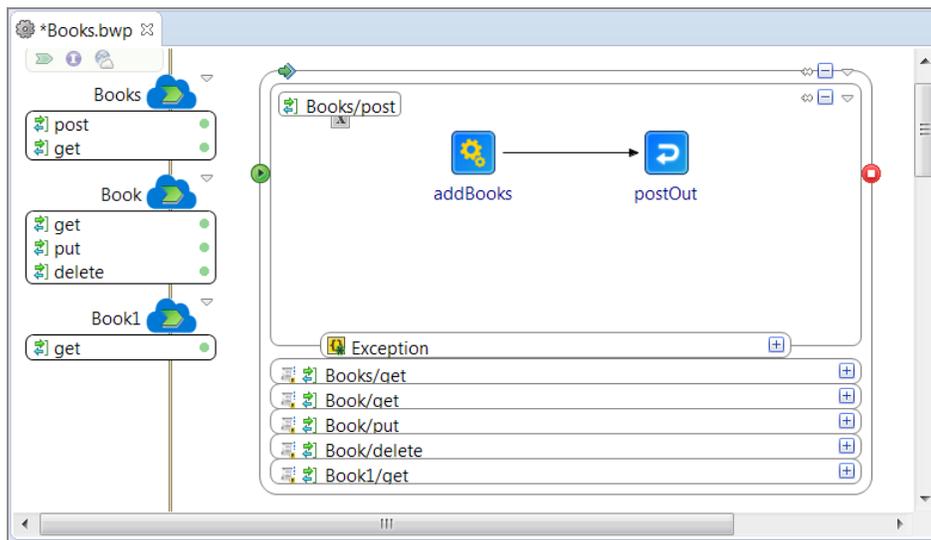
This contains a process service with two operations: OnStartup and OnShutDown.

The OnStartup process creates the Books and Events tables.



- **Books Process (Books.bwp)**

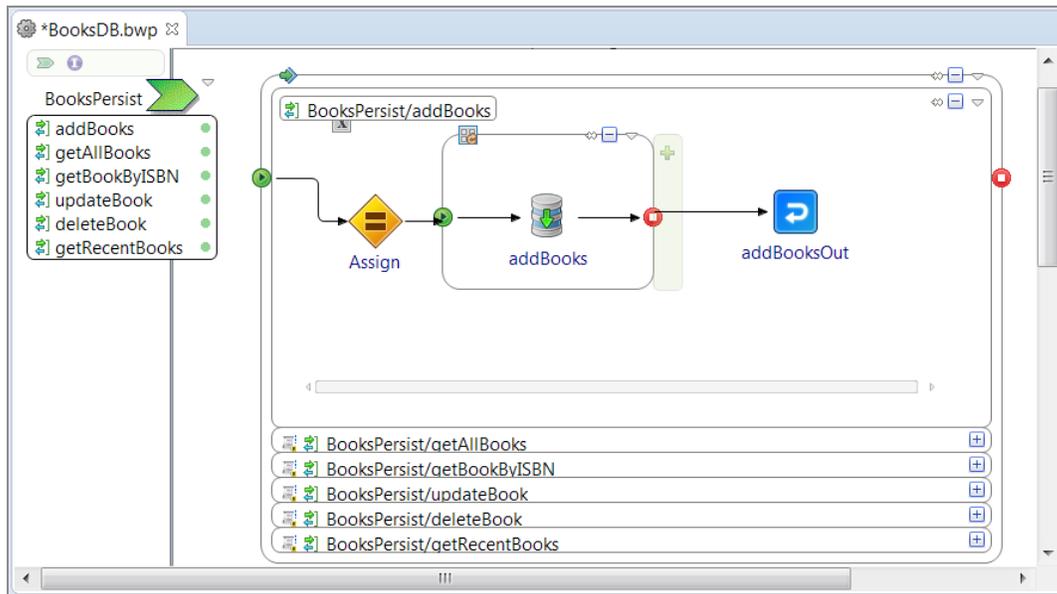
In this process, the Book REST service performs the post, get, put, delete operations. It posts one or more books to the bookstore and retrieves all the books from the bookstore. For the post operation, a sub process is called to add books to the database. The addBooks operation is implemented in the BooksDB subprocess. The Book REST service performs three operations: getting a book from the bookstore by ISBN, updating a book in the bookstore using ISBN, and deleting a specific book from the bookstore using ISBN. ISBN is passed as a parameter to these operations.



- **BooksDB subprocess (BooksDB.bwp)**

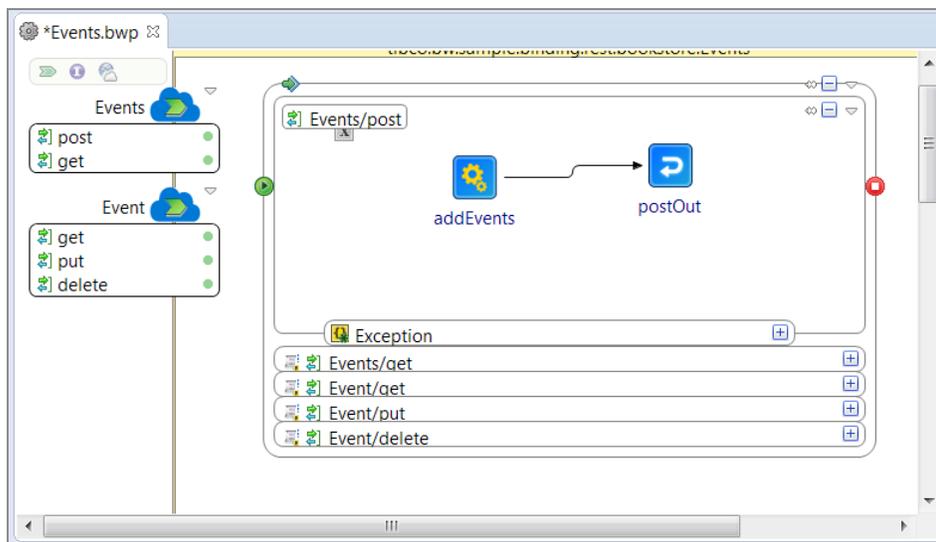
The subprocess handles deletes, updates, retrievals of books from the database. The BooksPersist service implements following operations: adding books to database, getting all books from database, getting a book by ISBN, updating a book by ISBN, deleting a book by ISBN. Using BooksPersist/addBooks, books are added to the

database.



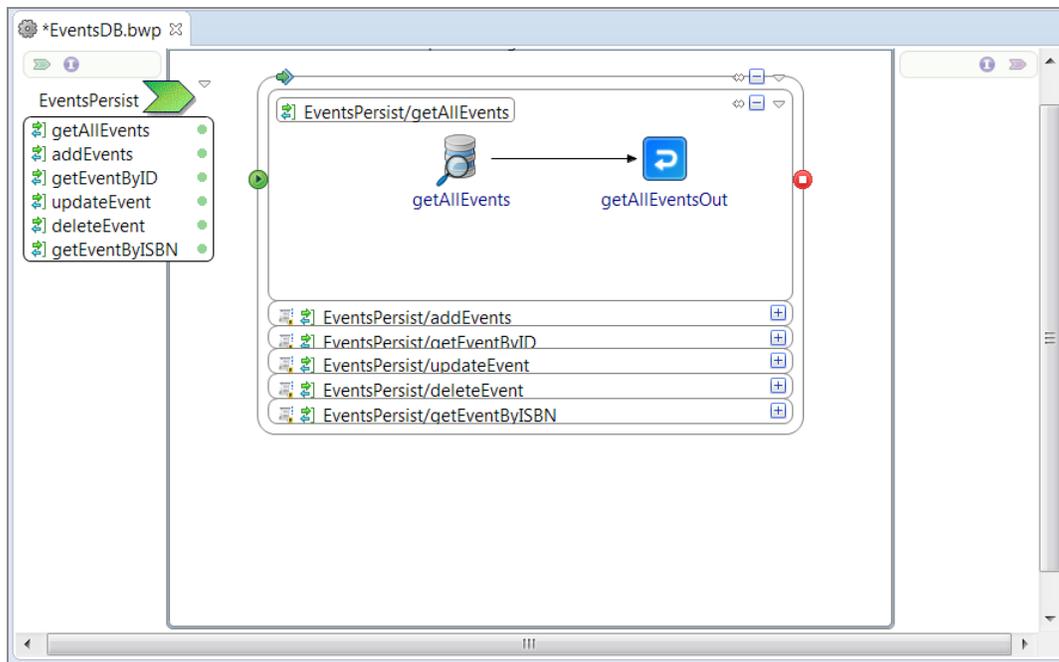
- **Events Process (Events.bwp)**

This process is similar to the Books process and is used to add, delete, update, and retrieve events.



- **EventsDB subprocess (EventsDB.bwp)**

This process is similar to the BooksDB process and is used to add, delete, update, and retrieve events from the database.



## Testing the REST BookStore Sample

This section describes the process of testing this sample in your Cloud Foundry environment.

### Before you begin

- The Cloud Foundry environment updated with TIBCO BusinessWorks Container Edition buildpack.
- A web browser.
- User provided service (CUPS) PostgreSQL service. Read the procedure for creating an user provided service from the Cloud Foundry documentation. For more information, see <https://docs.cloudfoundry.org/>.
- The cf command line interface tool installed on your machine.

## Creating User Provided Service Instance (CUPS) for Postgres

Cloud Foundry has commands for creating and updating user-provided service instances (cups). Once created, user-provided service instances can be bound to an application. To interact with the Cloud Foundry instance, the command line interface, cf CLI is used.

### Procedure

1. Log into your Cloud Foundry environment using the cf CLI.
2. Using the cf CLI, run

```
cf cups service instance -p "url,username,password"
```

For example,

```
cf cups postgres -p "url,username,password"
```

3. When prompted, specify values appropriate for your environment.

For example,

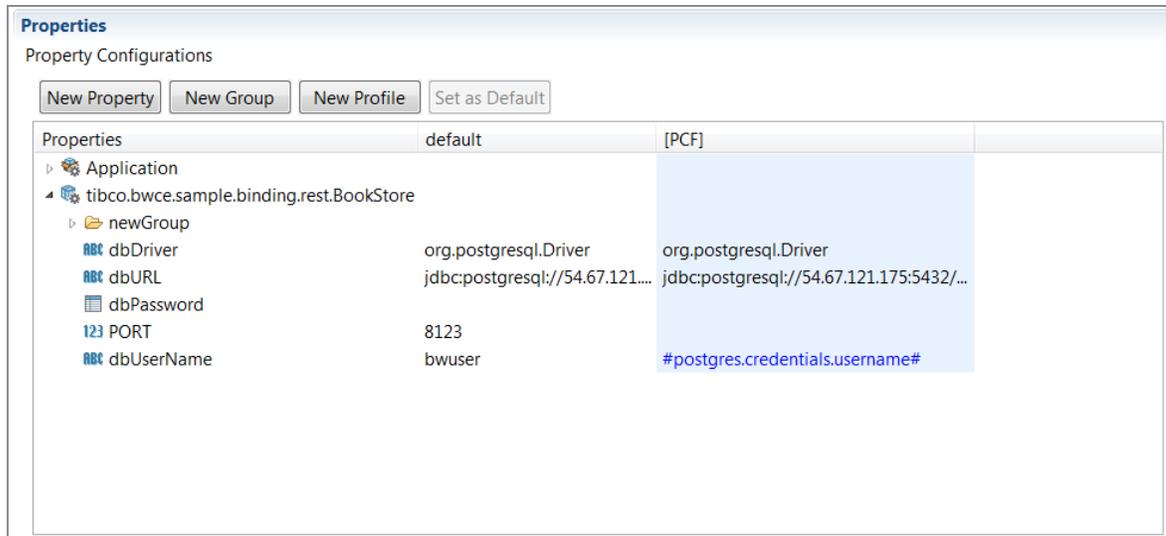
```
url> postgresql://191.181.20.101:5432/postgres
username>bwuser
password>#!b09Uc21Q4c6B7zaku2zx3rnzv8ImCxGddcVgi68ymo0=
```

## Setting the Default Application Profile

### Procedure

1. From Project Explorer, expand the **tibco.bwce.sample.binding.rest.BookStore.application**.
2. Expand the **Package Units** folder and click **Properties**.
3. Select the **PCF** profile and **Set As Default**.

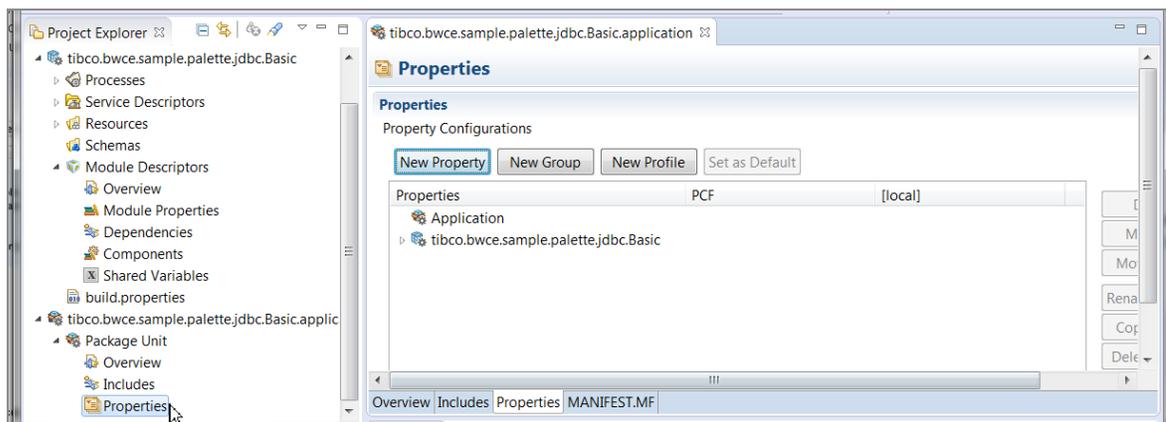
If you do not have a profile named **PCF**, click **New Profile** to create a profile and name it **PCF**.



## Modifying the Application Properties

### Procedure

1. From the Project Explorer navigate to **tibco.bwce.sample.binding.rest.BookStore.application** (as shown in the image) and click **Properties**.



Follow these steps to update the following properties:

- dbUserName
- dbPassword

- dbUrl

The values use the format `#<serviceName>.credentials.<parameter>#`

For example `#oracle.credentials.username#`.

- To update the **dbURL** property, first click the property name and then click the  icon.

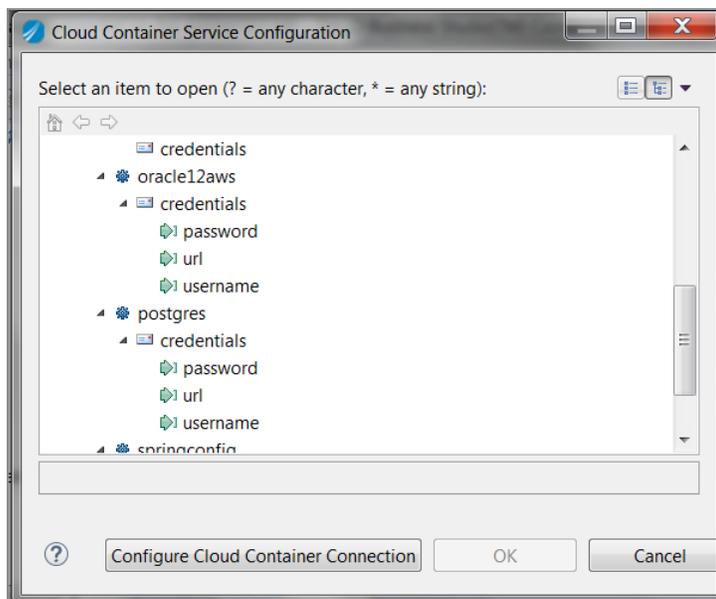
The **Cloud Container Service Configuration** window is displayed. This window initially contains no values.

- Click **Configure Cloud Container Connection**.

The **Cloud Container Connection Configuration** window is displayed.

- Specify the configuration properties and click **Test Connection**.

If the connection is successful, variables are populated in the **Cloud Container Service Configuration** window as shown in the next figure.



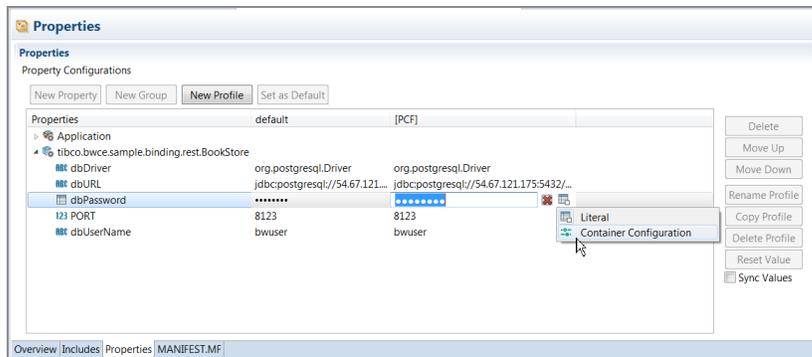
- Choose **url** under **postgres** and click **OK**.
- Similarly update the dbUsername property. The **Cloud Container Service Configuration** window will now contain values and you need not click **Configure Cloud Container Connection**.
- Follow steps in [dbPassword Property](#) to modify the dbPassword property.
- Click **Save**.

## dbPassword Property

Follow these steps to modify the dbPassword value:

### Procedure

1. Click the property name and then click the  icon and choose **Container Configuration** as shown in the next image.



The **Browse Container Configuration**  icon will now be visible.

2. Click **Browse Container Configuration**.  
The **Cloud Container Service Configuration** is displayed.
3. Choose **password** under the PostgreSQL service instance and click **OK**.
4. Click **Save**.

## Generating an Application Archive File

### Procedure

1. Expand the Package Unit and select **Overview** .
2. In the **Overview** window select **Export Application for Deployment**.
3. Enter the location of your EAR file.

## Deploying the REST BookStore Application

The following steps describe how to deploy the application to your Cloud Foundry environment.

## Procedure

1. Log into your Cloud Foundry environment using the `cf` command-line interface.
2. Copy the manifest file from the samples directory to the location where you placed your EAR file in the previous section.
3. Edit the **path** variable in the `manifest.yml` file and set the path to the location of the EAR file.

Check the CUPS service name, it should match with the service in your Cloud Foundry environment.

As the manifest file contains the service name, the **postgres** service instance automatically binds to this application at deployment.

4. Save the file.
5. In your Cloud Foundry environment change to the directory where you placed your EAR file.
6. Run

```
cf push -f manifest.yml
```

To see the log output use the command `cf logs <application name> --recent`  
For example,

```
ch logs tibco.bwce.sample.binding.rest.BookStore.application --  
recent
```

If the application deploys successfully, you can see a similar output in the console log.

```
bookstoreapp started 1/1 1G 1G bookstoreapp.tibcoqa.com
```

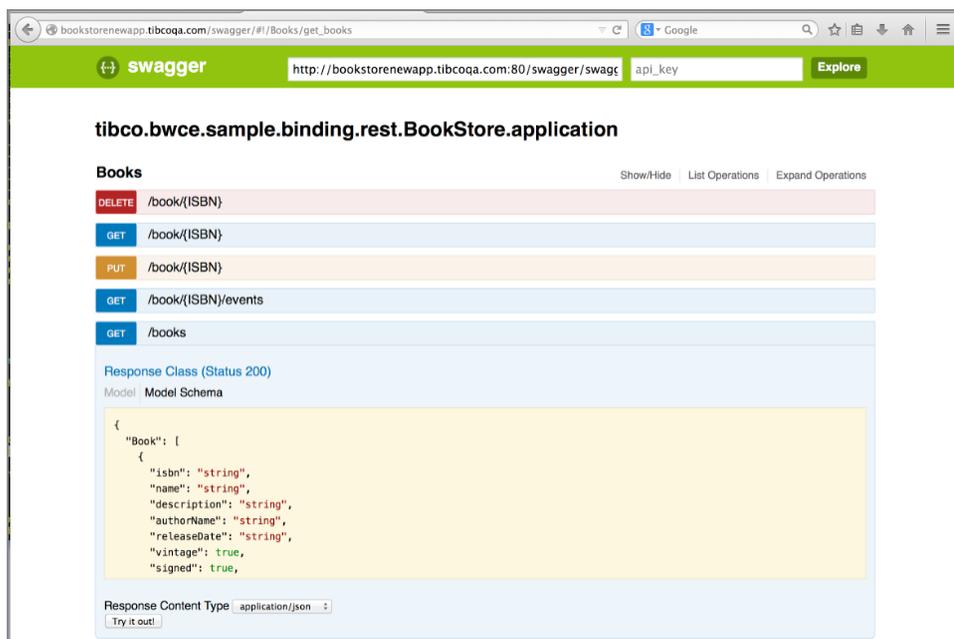
# Testing the REST Service

## Procedure

1. Append `swagger` to the routable url to access the swagger doc to test the Rest Service in the format `<routable url>/swagger`

For example:

```
bookstoreapp.testdomain.com/swagger/
```



2. Expand the **Books** and **Events** headers, and test the operations as listed below. Click any of the operations (POST, GET, PUT, DELETE) as displayed on the web page and click **Try it out!**

The screenshot shows a REST client interface with the following sections:

- GET /books**: The request method and path.
- Response Class (Status 200)**: A section for defining the response model.
- Model Schema**: A JSON schema defining the structure of the response:
 

```

{
  "name": "string",
  "description": "string",
  "authorName": "string",
  "releaseDate": "string",
  "vintage": true,
  "signed": true,
  "price": 0
}

```
- Response Content Type**: Set to `application/json`.
- Curl**: The command `curl -X GET --header "Accept: application/json" "http://bookstorenewapp.tibcoqa.com:80/books"`.
- Request URL**: `http://bookstorenewapp.tibcoqa.com:80/books`.
- Response Body**: The actual JSON response:
 

```

{
  "Book": [
    {
      "isbn": "1451648537",
      "name": "Cloud Foundry",
      "description": "Biography of Apple Co-Founder Steve Jobs",
      "authorName": "Walter Isaacson",
      "releaseDate": "2012-10-24Z",
      "vintage": false,
    }
  ]
}

```

3. Click the **Terminate** icon  to stop the process after you have completed using the operations on the page.

## Troubleshooting the REST Bookstore Sample

If `cf push` fails and the application fails to start. It is advisable to turn on the DEBUG logs to see if the application property values are correctly substituted.

To turn on the debug logs, add the environment variable `BW_LOGLEVEL` to your manifest file. For more information, see *Environment Variables* in the *Application Development* guide.

The debug log entries will help you troubleshoot whether the application properties are being correctly substituted.

```

OUT      </globalVariable>
OUT      <globalVariable>
OUT          <name>//tibco.bwce.sample.binding.rest.BookStore//dbUserName</name>
OUT          <value>#postgres.credentials.username#</value>
OUT          <deploymentSettable>>false</deploymentSettable>
OUT          <serviceSettable>>false</serviceSettable>
OUT          <type>String</type>
OUT          <isOverride>>true</isOverride>
OUT      </globalVariable>
OUT      <globalVariable>
OUT          <name>//tibco.bwce.sample.binding.rest.BookStore//BW.CLOUD.PORT</name>
OUT          <value>8080</value>
OUT          <deploymentSettable>>false</deploymentSettable>
OUT          <serviceSettable>>false</serviceSettable>
OUT          <type>Integer</type>
OUT          <isOverride>>false</isOverride>
OUT      </globalVariable>
OUT </globalVariables>
OUT </repository>
OUT PORT is 64752
ERR Value not found for Token [postgres.credentials.url].
ERR Exception in thread "main" java.lang.Exception: Value not found for Token [postgres.credentials.url].
ERR   at PCFProfileTokenResolver.resolveTokens(PCFProfileTokenResolver.java:341)
ERR   at PCFProfileTokenResolver.main(PCFProfileTokenResolver.java:46)
OUT ERROR: Failed to substitute properties.
ERR Instance (index 0) failed to start accepting connections
OUT App instance exited with guid b93ae570-12b0-47bb-980f-68c98421c1d9 payload: {"cc_partition"=>"default", "

```

Additionally, also check your CUPS service credentials in your environment.

## REST BookStore Consumption

This sample walks you through the process of creating an application in TIBCO BusinessWorks Container Edition to consume a RESTful webservice running in your Cloud Foundry environment.

### Before you begin

The Rest BookStore sample under **Container > cloudfoundry > binding > rest > Bookstore > tibco.bwce.sample.binding.rest.BookStore** must be running in your Cloud Foundry environment.

For more information, see [REST Bookstore](#)

## Consuming a BookStore REST Endpoint

### Importing the REST API document

Swagger 2.0-compliant REST API documents must be imported into the TIBCO Business Studio™ for BusinessWorks™ Service Descriptors folder of the project. This gives you the

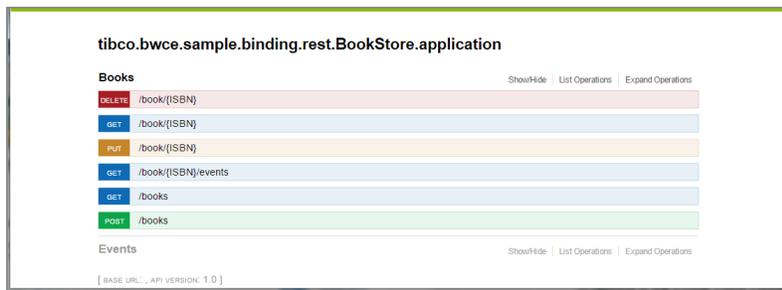
ability to expand and collapse endpoints, operations, parameters and response codes in the Project Explorer view.

## Accessing the BookStore Swagger

For the BookStore sample running in the Cloud Foundry environment, the swagger URL should be accessible using the routable URL in the format `http://routable URL/swagger`.

For example,

```
bookstoreapp.test.com:80/swagger/
```



## Retrieving the Swagger Document

You can now retrieve the swagger document by accessed using the URL

```
http://routable URL/swagger/swagger.json
```

or using the following curl command:

```
curl http://bookstoreapp.test.com:80/swagger/swagger.json -o bookstore.json
```

## Importing the Swagger Document in Service Descriptors Folder

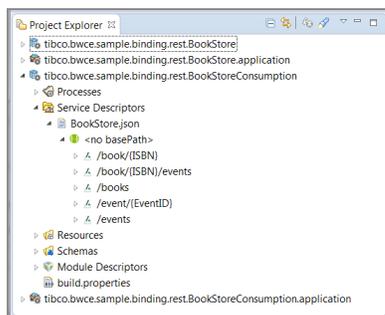
For configuring the design time, import the **bookstore.json** file in your workspace project under the **Service Descriptors** folder.

# Creating a REST Reference Binding

## Procedure

1. From the **File Explorer**, navigate to the samples directory and select **samples > container > cloudfoundry > binding > rest > BookStoreConsumption** and double-click **tibco.bwce.sample.binding.rest.BookStoreConsumption**.
2. From the Project Explorer, navigate to **tibco.bwce.sample.binding.rest.BookStoreConsumption > Service Descriptors > bookstore.json** to view the endpoints,

This will create a cloud shaped icon with a right facing arrow. The cloud is an indication that it is a REST Reference whereas the arrow within the cloud indicates that it is a binding. Since the binding is within a cloud, it is an indication that it is a REST binding.

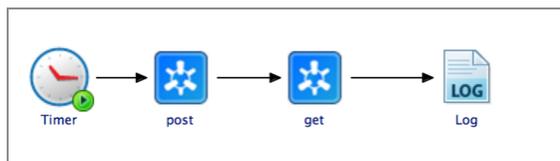


3. Drag and drop an endpoint ( **/books**) on the right side of the canvas to create a REST Reference Binding.

This creates an Invoke activity which is pre-configured to invoke the operation. It also creates an HTTP Client Shared Resource with the routable URL as the default host. The configuration for these entities is copied from the Swagger document from which you created the Reference Binding. The Reference consists of the name of the API as well as the operations it supports.

4. Test the configured process using the TIBCO Business Studio Debugger

The process will first try to run a POST a book and then get all the books from the BookStore Service.



The output of the GET operation can be seen as

```
09:05:42.933 INFO [bwEngThread:In-Memory Process Worker-3]
c.t.b.p.g.L.t.b.s.b.s.r.B.module.Log - First Book
Returned:Biography of Apple Co-Founder Steve Jobs
```

## Soap with External Binding

In this sample, a soap over HTTP service implements a simplified StockQuote application. The StockQuote service exposes a service binding for a StockQuote endpoint, that returns the StockQuote value when it receives a request with a stock ticker symbol. The StockQuoteClient application uses an invoke activity to trigger the StockQuote service.

## Testing Locally

This sample calls out to a public web service endpoint to return the stock quote value. For testing purposes, the samples directory includes a **tibco.bwce.sample.binding.soap.http.StockQuoteExternal** sample. This acts as a public endpoint that needs to be deployed in your Cloud Foundry environment and the generated routable URL used in the HTTPClient configuration of the StockQuoteService.

### Before you begin

The **tibco.bwce.sample.binding.soap.http.StockQuoteExternal** has to be deployed in your Cloud Foundry environment.

### Procedure

1. In the samples directory, select **samples > Container > cloudfoundry > binding > soap > http > StockQuoteService** and double-click **tibco.bwce.sample.binding.soap.http.StockQuote**. See [Accessing Samples](#) for more information.
2. From the **Project Explorer** expand the **tibco.bwce.sample.binding.soap.http.StockQuote** project.

3. Set the **Application Profile** to **PCF**. See [Setting the Default Application Profile](#).
4. Click **Run > Debug Configurations**.
5. From the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications.
7. Select the checkbox next to **tibco.bwce.sample.binding.soap.http.StockQuote.application**.
8. Click **Debug**.

The sample now runs in the debug mode.

The console window shows engine messages similar to:

```
12:07:31.300 INFO [Thread-25] com.tibco.thor.frwk.Application -  
TIBCO-THOR-FRWK-300006: Started BW Application  
[tibco.bwce.sample.binding.soap.http.StockQuote.application:1.0]  
12:07:31.301 INFO [Framework Event Dispatcher: Equinox Container:  
90d08c1f-1e9d-0015-1579-c1f8429c33a2]  
com.tibco.thor.frwk.Application - Started by BusinessStudio,  
ignoring
```

## Result

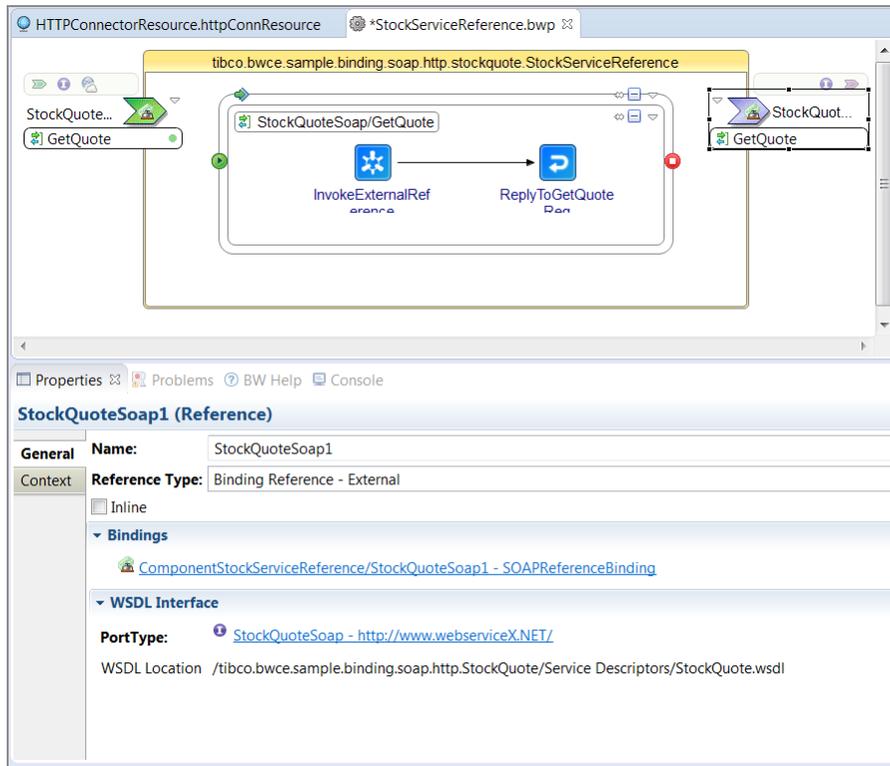
This sample is triggered using the soap client sample under **cloudfoundry > binding > soap > http > StockQuoteClient**. For more information, see [StockQuote Soap Client](#)

## Understanding the Configuration

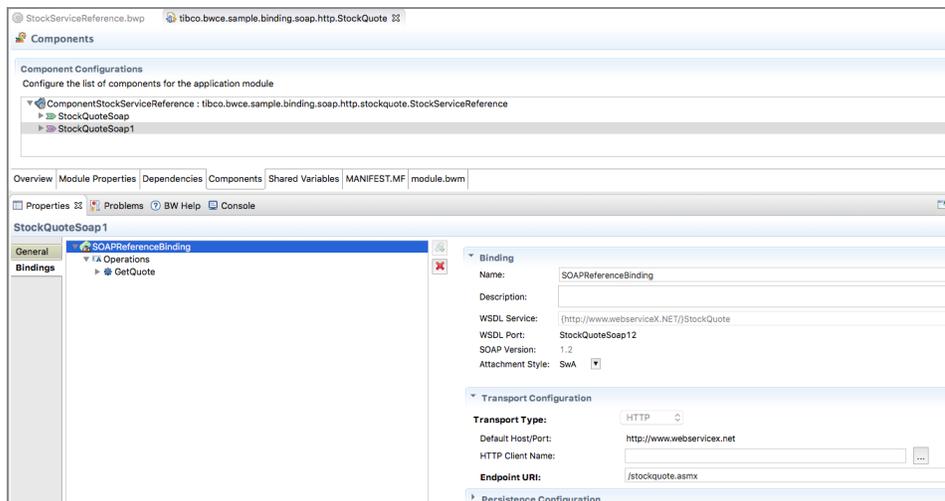
The following shared resources are defined in the project:

- HTTPConnectorResource.httpConnResource: The HTTP Connection Shared Resource that contains the transport configuration for the StockQuoteService.
- HTTPClientResource.httpClientResource: The HTTP Client Shared Resource that contains the transport configuration for the StockQuote external web service.
- StockQuote.wsdl: The WSDL file that describes the StockQuote webservice. The stockquote process consists of an invoke activity that calls out to a public webservice

endpoint and in the context of this sample,  
**tibco.bwce.sample.binding.soap.http.StockQuoteExternal.**



- The Reference binding `StockQuoteSoap1` is configured with **Binding Reference>External**. In this case the Binding is configured at the component level.



## Testing the SOAP Sample

### Before you begin

- The Cloud Foundry environment updated with TIBCO BusinessWorks Container Edition buildpack.

## Setting the Default Application Profile

### Procedure

1. From the Project Explorer expand the **tibco.bwce.sample.binding.soap.http.StockQuote.application**.
2. Expand the **Package Units** folder and click **Properties**.
3. Select the **PCF** profile and **Set As Default**.

If you do not have a profile named **PCF**, click **New Profile** to create a profile and name it **PCF**.

## Generating an Application Archive File

### Procedure

1. Expand the Package Unit and select **Overview** .
2. In the **Overview** window select **Export Application for Deployment**.
3. Enter the location of your EAR file.

## Deploying the Application

The following steps describe how to deploy the application to your Cloud Foundry environment using the cf command line interface tool.

### Procedure

1. Log into your Cloud Foundry environment using the cf command-line interface.

2. Copy the manifest file from the samples directory to the location where you placed your EAR file in the previous section.
3. Edit the **path** variable in the `manifest.yml` file. Set the path to the location of the EAR file and Save the file.

As the manifest file contains the **service** name, the service automatically binds to this application at deployment.

4. In your Cloud Foundry environment change to the directory where you placed your EAR file.
5. Run the following command .

```
cf push -f manifest.yml
```

To see the log output use the command `cf logs <application name> --recent`.  
For example,

```
cf logs tibco.bwce.sample.binding.soap.http.StockQuote.application  
--recent
```

## StockQuote Soap Client

The StockQuote Client is a Soap Client for the StockQuote Service and uses an Invoke Activity to trigger the StockQuote application.

### Before you begin

The sample StockQuote is deployed and running in your Cloud Foundry environment and you have access to the routable URL to access the sample.

### Procedure

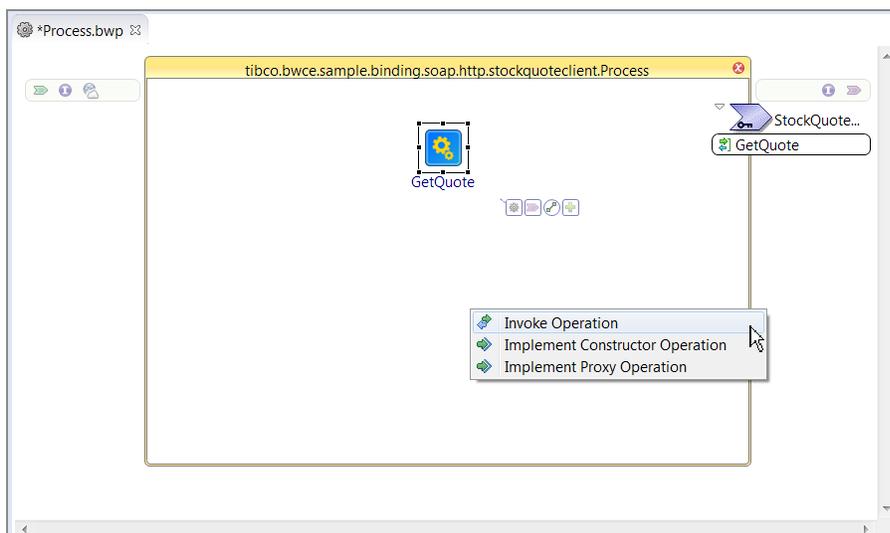
1. From the **File Explorer** navigate to the **samples** directory, select **binding > soap > http > StockQuoteClient** and double-click **tibco.bwce.sample.binding.soap.http.StockClient**. For more information, see [Accessing Samples](#)

2. From the **Project Explorer** expand the **tibco.bwce.sample.binding.soap.http.StockQuoteClient** project.
3. Set the **Application Profile** to **PCF**. For more information, see [Setting the Default Application Profile](#).
4. Click **Run > Debug Configurations**.
5. From the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications.
7. Select the checkbox next to **tibco.bwce.sample.binding.soap.http.StockQuoteClient.application**.
8. Click **Debug**.

The sample now runs in the debug mode.

## Understanding the Configuration

The Process uses an Invoke Activity to call out to the StockQuote application. The process can be configured by dragging the concrete WSDL, in this case StockQuote\_gen1.wsdl onto the workspace canvas and selecting the **Invoke Operation** option



The Reference created is set to type Binding - Reference. Create a SOAP/HTTP reference binding that uses a HTTP Client Resource.

The default host for the HTTP Client Resource should point to your Cloud Foundry routable URL for the Stock Quote application. For example, `stockquote.testqa.com`. The Log output will display the current stock quote value for the symbol which in this case is hard coded to `G00G`.

**i Note:** The external service that the StockQuote service talks to an external service, it may intermittently go down and in that case the StockQuoteClient may not get the correct response back from the StockQuote Service.

## Implementing a JMS Service that Returns Information Based on Zip Codes

The JMS service returns information about a city given a zip code. The JMS service also provides the distance between two cities as defined by their zip codes.

### Before you begin

- TIBCO Enterprise Message Service must be running.
- Ensure the queues `jmsbasic.queue` and `reply.queue` have been created.

## Testing Locally

### Procedure

1. In the samples directory, select **binding > soap > jms > ZipCodeLookup** and double-click **tibco.bwce.sample.binding.soap.jms.ZipCodeLookup**. For more information, see [Accessing Samples](#).
2. From the Project Explorer expand the **tibco.bwce.sample.binding.soap.jms.ZipCodeLookup** project.
3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see [Setting the Default Application Profile](#).
4. Expand the Processes directory and double-click **ZipInfoService.bwp**.

5. Verify your TIBCO Enterprise Message Service connection.
  - a. Expand the **Resources** directory.
  - b. Double-click **JMSConnectionResource.jmsConnResource**.  
Click the **Test Connection** button to verify the connection.
6. Click **File > Save**
7. Click **Run > Debug**
8. From the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
9. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications.
10. Select the checkbox next to **tibco.bwce.sample.binding.soap.jms.ZipCodeLookup.application**.
11. Click **Debug**.  
The sample now runs in the debug mode.
12. Click the Terminate  icon when you have completed using the listed operations.

## Result

- For the **getCityInfo** service, information corresponding to the zip code defined in the zip field is returned. The default value is **61801**, which returns information about Urbana, IL.
- For the **getCityDistance** service, information corresponding to the distance between two zip codes is returned. The default values are **61801** for Urbana IL and 61820 for Champaign IL.

## Understanding the Configuration

The project contains the **ZipInfoService** process that provides city information about a zip code. JMS is used for the transport. The project represents a simple JMS based service using SOAP.

The **getCityDistance** and **getCityInfo** operations are implemented in the **ZipInfoServiceprocess**.

The sample also includes a test client process called **TestService.bwp**, that invokes the **ZipInfoService** process.

## Testing the Sample

### Before you begin

- The Cloud Foundry environment updated with the TIBCO BusinessWorks Container Edition buildpack.
- A user provided service instance for TIBCO EMS Server. Read the procedure for creating a user provided service from the Cloud Foundry documentation. For more information, see <https://docs.cloudfoundry.org/>.

## Creating User Provided Service (CUPS) for TIBCO EMS Server

Cloud Foundry has commands for creating and updating user-provided service instances (cups). Once created, user-provided service instances can be bound to an application. To interact with the Cloud Foundry instance, the command line interface, cf CLI is used.

### Procedure

1. Log into your Cloud Foundry environment using the cf CLI.
2. Using the cf CLI, run `cf cups ems -p "url"`.
3. When prompted, specify values appropriate for your environment.

For example,

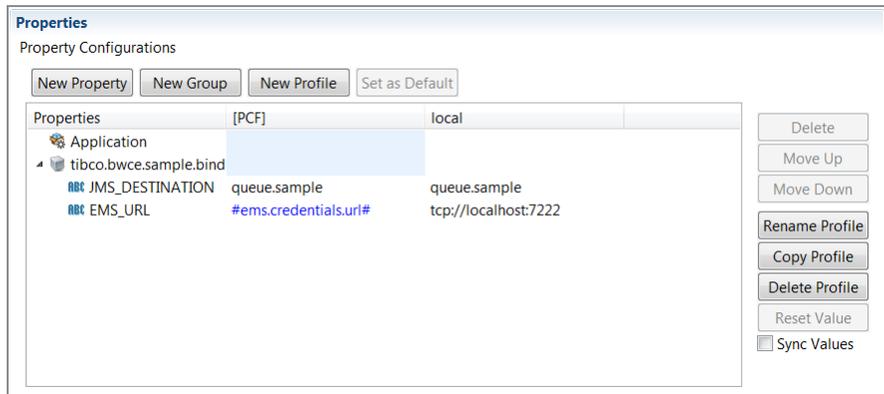
- `url> tcp://152.188.28.123.7222`

## Setting the Default Application Profile

### Procedure

1. From the Project Explorer expand the **tibco.bw.sample.binding.soap.jms.ZipCodeLookup.application**.
2. Expand the **Package Units** folder and click **Properties**.
3. Select the **PCF** profile and **Set As Default**.

If you do not have a profile named **PCF**, click **New Profile** to create a profile and name it **PCF**.



## Deploying the Sample

The following steps describe how to deploy the application to your Cloud Foundry environment.

### Procedure

1. Log into your Cloud Foundry environment.
2. Copy the manifest.yml file from the **samples** directory to the directory that contains the EAR file.
3. Edit the **PATH** variable in the manifest.yml file. Set the path to the location of the EAR file and **Save** the file.  
As the manifest file contains the service name, the service automatically binds to this application at deployment.
4. In your Cloud Foundry environment change to the directory where you placed your EAR file.
5. From the terminal, run `cf push -f manifest.yml`.

## Result

To see the log output use the command `cf logs <application name> --recent`.

If the application deploys successfully, you can see a similar output in the console log.

```
13:49:06.733 INFO [Framework Event Dispatcher: Equinox Container:
608aaa95-52d7-0015-1f90-d3fd61937bf8] com.tibco.thor.frwk.Application -
Started by BusinessStudio, ignoring .enabled settings.
```

```
13:49:11.962 INFO [bwEngThread:In-Memory Process Worker-6]
c.t.b.p.g.L.t.b.s.b.s.j.Z.Log - Output of getCityInfo : 61801 Urbana
Illinois Urbana, Illinois, United States 40.11 88.207
```

```
Output of getCityDistance : Distance between two cities is : 4 miles.
61801 61820 Urbana Champaign Illinois Illinois Urbana, Illinois, United
States Champaign, Illinois, United States
```

## Resilience4j

This sample describes the Resilience4j cloud native functionality using a HTTP Request Response sample in TIBCO BusinessWorks Container Edition.

## Testing in TIBCO Business Studio for BusinessWorks

### Before you begin

- Your computer must be connected to the Internet.
- Setup Prometheus to monitor events and Grafana dashboard provided by Resilience4j for visualization.

### Procedure

1. From the **File Explorer**, navigate to the samples directory and select **samples > Container > cloudfoundry > core > Resilience4j** and double-click `tibco.bwce.sample.core.Resilience4j`.
2. From the **Project Explorer** expand the `tibco.bwce.sample.core.Resilience4j` project.

3. Expand the **Processes** folder and double-click **HTTP\_Request\_Response\_Example.bwp**.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bwce.sample.core.Resilience4j.application**.
7. Click **Debug**.  
This runs the sample in the debug mode.
8. Open the `request_news.html` file found in **samples > Container > cloudfoundry > core > Resilience4j > Basic** folder.
9. Click the **Get News from Wiki!** button to request headlines from the associated web page.

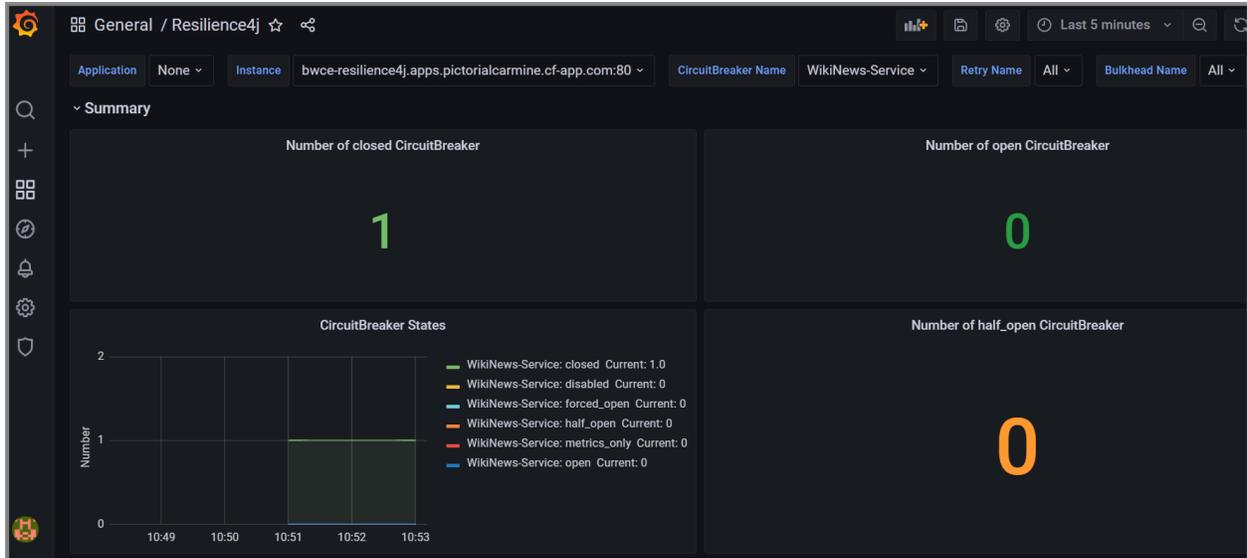
## Result

The Wiki web page displays in your default browser. The message `Response sent successfully!!` is printed on the TIBCO Business Studio for BusinessWorks console.

In order to view resilience4j metrics, you need a resilience4j stream URL which is as follows:

```
http://<Route URL>/resilience4j_metrics
```

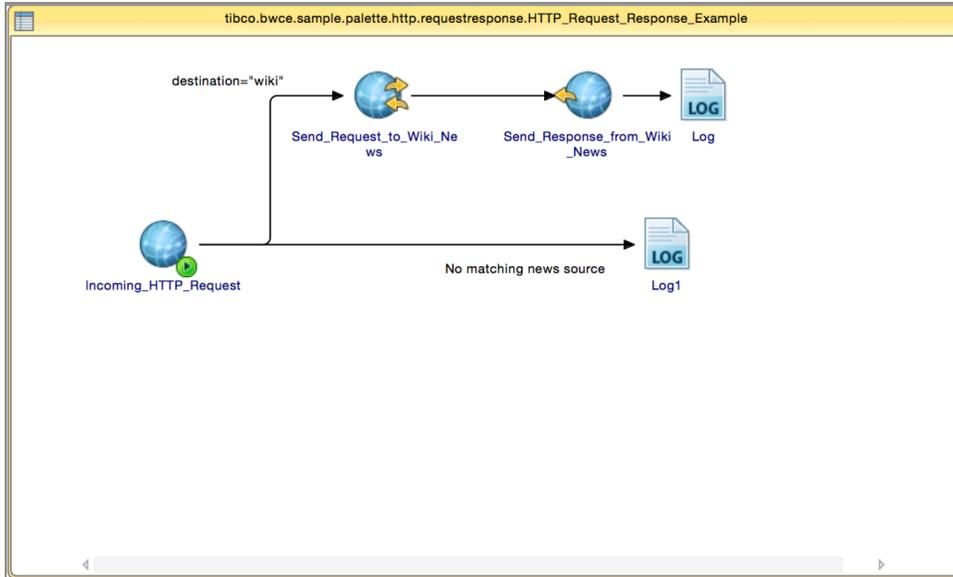
You can see the Grafana dashboard locally and the command name given is displayed as shown below.



## Understanding the Configuration

The following shared resources are defined in the project:

- The Incoming HTTP Request process starter listens on the connection specified in `ListeningHTTPConnection.httpConnResource`. The `request_news.html` file contains a form and clicking the **Get News from Wiki!** button sends the corresponding text string to the Incoming HTTP Request activity.
- The conditional transition routes the request to the Send HTTP Request activity, which sends the request to the host using the `ListeningHTTPConnection`. After receiving a response from the remote site, the Send HTTP Response activity closes the HTTP Connection established by the Incoming HTTP Request process starter.
- An internet connection is required for the sample to connect to Wiki News. The `ListeningHTTPConnection` listens on `BW.CLOUD.PORT` (default value 8080).



- The HTTP client shared resource contains the Circuit Breaker configuration as shown below:

**Circuit Breaker Configuration**

**Enable Circuit Breaker:**  **Circuit Breaker Name:**

**Circuit Breaker Property**

Sliding Window Type:	<input type="text" value="COUNT_BASED"/>	Slow Call Rate Threshold (%):	<input type="text" value="100"/>
Failure Rate Threshold(%):	<input type="text" value="50"/>	Slow Call Duration Threshold(ms):	<input type="text" value="60000"/>
Minimum Number of Calls:	<input type="text" value="100"/>	Automatic Transition From Open To HalfOpen Enabled:	<input type="checkbox"/>
Permitted Number Of Calls In HalfOpen State:	<input type="text" value="10"/>	Sliding Window Size:	<input type="text" value="100"/>
Wait Duration In Open State (ms):	<input type="text" value="60000"/>	Max Wait Duration In HalfOpen State (ms):	<input type="text" value="6000"/>

## Testing the Resilience4j Sample

### Before you begin

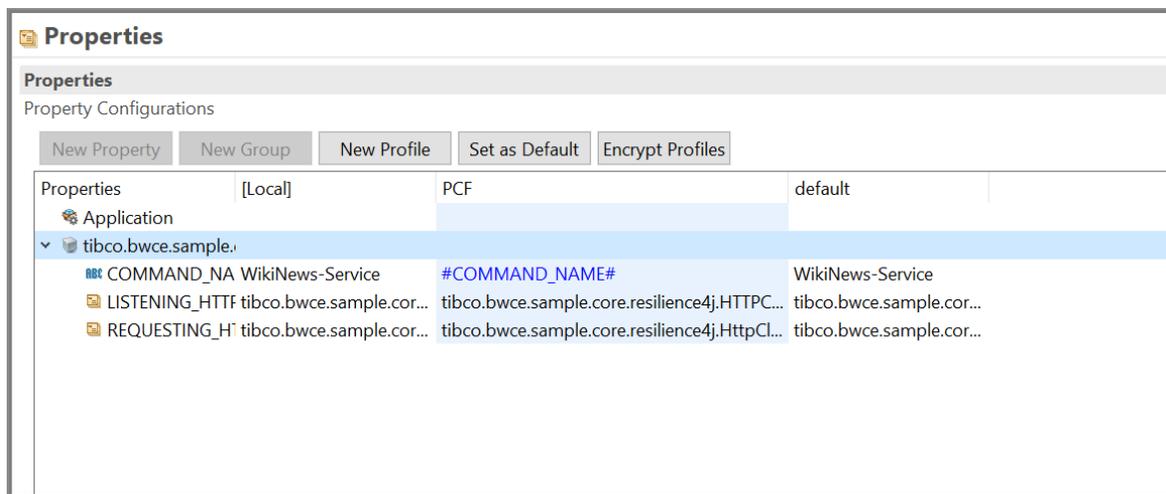
- Your Cloud Foundry environment updated with TIBCO BusinessWorks Container Edition buildpack.
- A web browser.
- The cf command line interface tool installed on your machine.

## Setting the Default Application Profile

### Procedure

1. From the **Project Explorer** expand the **tibco.bwce.sample.core.Resilience4j.application**.
2. Expand the **Package Units** folder and click **Properties**.
3. Select a profile and **Set As Default**.

If you do not have a profile named **PCF**, click **New Profile** to create a profile and name it **PCF**.



## Generating an Application Archive File

Follow these steps to generate the .EAR file:

### Procedure

1. Expand the Package Unit and select **Overview**.
2. In the **Overview** window select **Export Application for Deployment**.
3. Enter the location of your EAR file.

# Deploying the Resilience4j Application

The following steps describe how to deploy the application to your Cloud Foundry environment using the cf command line interface tool.

## Procedure

1. Log into your Cloud Foundry environment using the cf command line interface.
2. Copy the manifest file from the samples directory to the location where you placed your EAR file in the previous section.
3. Edit the **path** variable in the manifest.yml file.  
Make sure you have the name of your Circuit Breaker service in your manifest file.
4. Set the **path** to the location of the EAR file.  
Check the user provided or managed service name, it should match with the service in your Cloud Foundry environment.
5. **Save** the file.
6. In your Cloud Foundry environment change to the directory where you placed your EAR file.
7. Run the following command:

```
cf push -f manifest.yml
```

8. Copy the routable URL of the application once it starts executing.
9. Edit the request\_news.html page found in **samples > container > cloudfoundry > core > Basic > Resilience4j** directory.
10. Replace the URL `http://127.0.0.1:8080` with the routable URL of the application.  
For example,

```
bwce-resilience4j.<domainName>.com
```

11. **Save** the file.
12. Open the request\_news.html file in a web browser

13. Click the **Get News from Wiki!** button to request headlines from the associated web page.

## Result

To see the log output use the command:

```
cf logs application name --recent
```

When the application deploys successfully, you can see a similar output in the console log:

```
16:13:31.231 INFO [Framework Event Dispatcher: Equinox Container:
007ddba5-b2ac-0016- 16a8-b400b6aea43d] com.tibco.thor.frwk.Application -
TIBCO-THOR-
FRWK-300018: Deploying BW Application
[tibco.bwce.sample.core.Resilience4j.application:1.0].
16:13:36.025 INFO [Framework Event Dispatcher: Equinox Container:
007ddba5-
b2ac-0016- 16a8-b400b6aea43d] com.tibco.thor.frwk.Application - TIBCO-
THOR-FRWK-300021: All Application dependencies are resolved for
Application[tibco.bwce.sample.core.Resilience4j.application:1.0]
16:13:38.266 INFO [Thread-21] com.tibco.thor.frwk.Application - TIBCO-
THOR-FRWK-300006: Started BW Application
[tibco.bwce.sample.core.Resilience4j.application:1.0] 05:00:28.966 INFO
[bwEngThread:In-Memory Process Worker-2] c.t.b.p.g.L.t.b.s.p.h.R.Log -
Response sent successfully!!
```

## Service Discovery

This sample walks you through the process of creating an application in TIBCO BusinessWorks Container Edition for registering a Service and discovering the registered service and deploying it in your Cloud foundry environment.

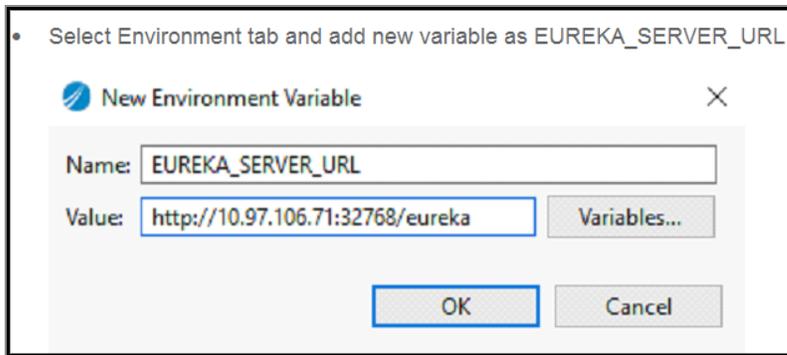
# Testing in TIBCO Business Studio for BusinessWorks

## Before you begin

- Access to a local Consul or Eureka server URL used for Service Registration and Discovery.
- A web browser.

## Procedure

1. From the **File Explorer**, navigate to the samples directory and select **Container > cloudfoundry > core > ServiceDiscovery** and double-click `tibco.bwce.sample.core.servicediscovery.Service`.
2. From the **File Explorer**, navigate to the samples directory and select **Container > cloudfoundry > core > ServiceDiscovery** and double-click `tibco.bwce.sample.core.servicediscovery.Client`.
3. From the **Project Explorer** expand the `tibco.bwce.sample.core.servicediscovery.Service.application` project.
4. Expand the **Package Unit** folder and double-click **Properties**.
5. Set the **Application Profile** to **Local**. For more information, see [Setting the Default Application Profile](#).
6. From the **Project Explorer** expand the `tibco.bwce.sample.core.servicediscovery.Client.application` project.
7. Expand the **Package Unit** folder and double-click **Properties**.
8. Set the **Application Profile** to **Local**. For more information see [Setting the Default Application Profile](#).
9. Click **File > Save** to save the project.
10. Expand the **Processes** folder and the **com.tibco.sample.servicediscovery.client** package within that folder.
11. Click **Run > Debug Configurations**.
12. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
13. Select the Environment tab to add an environment variable.



14. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications.
15. Select the check boxes next to **tibco.bwce.sample.core.servicediscovery.Service** and **tibco.bwce.sample.core.servicediscovery.Client**.
16. Click **Debug**.  
The sample now runs in the debug mode.
17. Trigger the application by accessing:

```
http://<hostname>:8080
```

18. Click the **Terminate**  icon to stop the process.

## Result

The console window shows messages similar to:

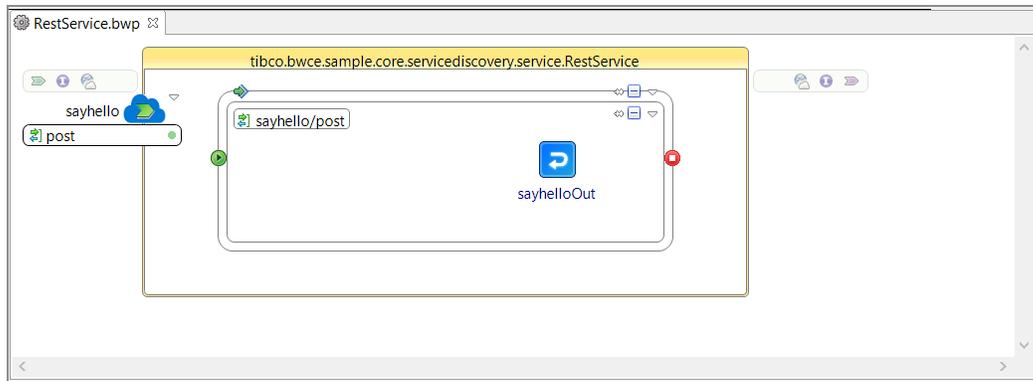
```
17:35:39.320 INFO [bwEngThread:In-Memory Process Worker-2]
c.t.b.p.g.L.t.b.s.S.Hystrix.Log - "Hello from TIBCO"
```

## Understanding the Configuration

The Service project contains the following:

- The RestService Process (**RestService.bwp**)

This process contains a process service with one operation sayhello.



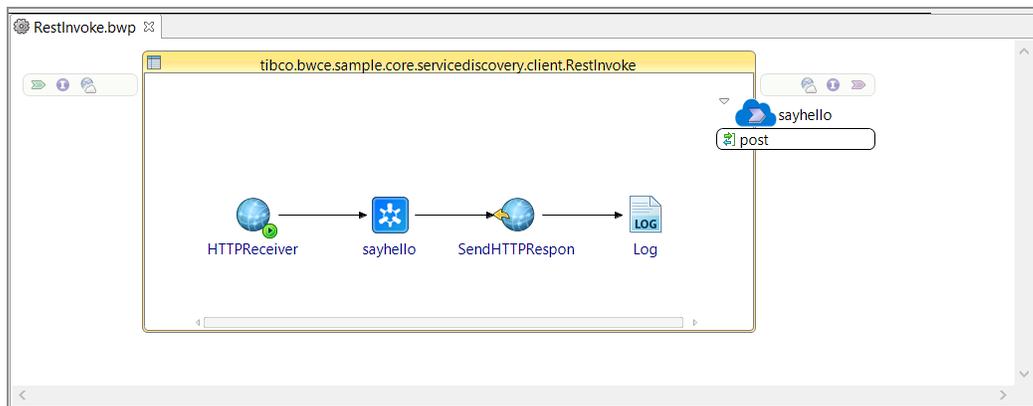
- To discover your service, enable service discovery and Circuit Breaker in the HTTP connector shared resource of your Service project.



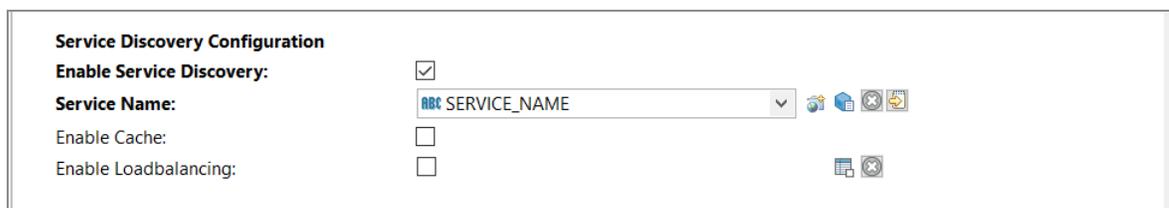
The Client project contains the following:

- The RestInvoke Process (**RestInvoke.bwp**)

This process contains Invoke operation which invokes the `sayhello` operation. The HTTP Receiver is the process starter which triggers the process when it receives the request on the listening port.



- In the client project, you have to enable the service discovery configuration in HTTP client shared resource.



# Testing the Service Registry Sample

## Before you begin

- Your Cloud Foundry environment updated with TIBCO BusinessWorks Container Edition buildpack.

For Eureka, add the service from the VMware Tanzu marketplace.

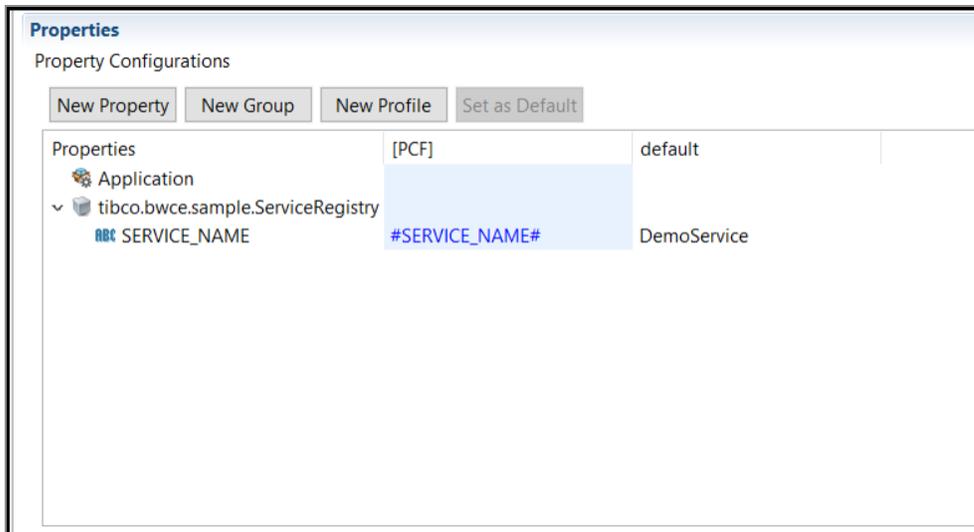
- A web browser.
- The cf command line interface tool installed on your machine.
- Eureka service must be configured in your Cloud Foundry environment.

## Setting the Default Application Profile

### Procedure

1. From the **Project Explorer** expand the **tibco.bwce.sample.core.servicediscovery.Service.application**.
2. Expand the **Package Units** folder and click **Properties**.
3. Select a profile and **Set As Default**.

If you do not have a profile named **PCF**, click **New Profile** to create a profile and name it **PCF**.



4. Similarly set the default application profile to **PCF** for the **tibco.bwce.sample.core.servicediscovery.Client.application**.

## Generating Application Archive Files

Follow these steps to generate the .EAR files:

### Procedure

1. Click **tibco.bwce.sample.core.servicediscovery.Service.application**.
2. Expand the Package Unit and select **Overview**.
3. In the **Overview** window select **Export Application for Deployment**.
4. Enter the location of your EAR file.
5. Repeat Steps 1 to 4 for **tibco.bwce.sample.core.servicediscovery.Client.application**

## Deploying the Service Application

The following steps describe how to deploy the service application to your Cloud Foundry environment using the cf command line interface tool.

## Procedure

1. Log into your Cloud Foundry environment using the cf command line interface.
2. Copy the manifest file from the samples directory to the location where you placed your EAR file in the previous section.
3. Edit the **path** variable in the `manifest-service.yml` file.
4. Set the **path** to the location of the EAR file.
5. Check the user provided or managed service name, it should match with the service in your Cloud Foundry environment.

As the manifest file contains the name of the Eureka service, the **EUREKA-Registry** service automatically binds to this application at deployment.

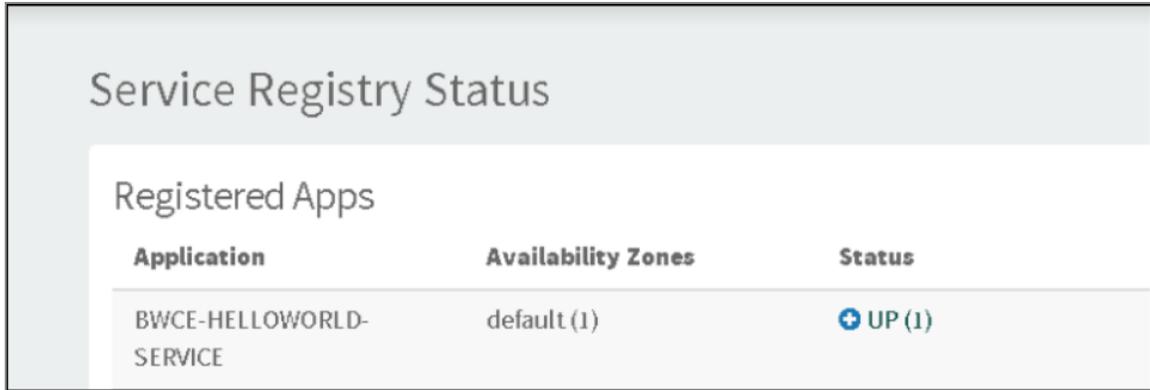
6. **Save** the file.
7. In your Cloud Foundry environment navigate to the directory where you placed your EAR file.
8. Run the following command:

```
cf push -f manifest-service.yml
```

To see the log output use the command `cf logs <application name> --recent`. For example,

```
cf logs bwce-helloworld-service --recent
```

When the application deploys successfully, your service is registered with the Eureka Server as shown in the following figure:



Service Registry Status		
Registered Apps		
Application	Availability Zones	Status
BWCE-HELLOWORLD-SERVICE	default (1)	UP (1)

## Deploying the Client Application

The following steps describe how to deploy the service application to your Cloud Foundry environment using the cf command line interface tool.

### Procedure

1. Log into your Cloud Foundry environment using the cf command line interface.
2. Copy the manifest file from the samples directory to the location where you placed your EAR file in the previous section.
3. Edit the **path** variable in the `manifest-client.yml` file.
4. Set the **path** to the location of the EAR file.
5. Check the user provided or managed service name, it should match with the service in your Cloud Foundry environment.

As the manifest file contains the name of the Eureka service, the **EUREKA-Registry** service automatically binds to this application at deployment.

6. **Save** the file.
7. In your Cloud Foundry environment navigate to the directory where you placed your EAR file.
8. Run the following command:

```
cf push -f manifest-client.yml
```

To see the log output use the command `cf logs <application name> --recent`. For example,

```
cf logs bwce-helloworld-client --recent
```

When the application deploys successfully, using a browser access the application using its routable URL.

You should see the following output:

```
"Hello from TIBCO"
```

## HTTP Request Response

This sample demonstrates how to use the HTTP palette activities to configure requests to a web server and manage the responses.

### Before you begin

- The Cloud Foundry environment updated with the TIBCO BusinessWorks Container Edition buildpack.

## Testing Locally

### Procedure

1. In the samples directory, select **Container > cloudfoundry > palette > http > RequestResponse** and double-click **tibco.bwce.sample.palette.http.RequestResponse**. For more information, see [Accessing Sample](#)
2. From the **Project Explorer** expand the **tibco.bwce.sample.palette.http.RequestResponse** project.

3. Click **Run > Debug Configurations**.
4. From the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
5. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications.
6. Select the checkbox next to **tibco.bwce.sample.palette.http.RequestResponse.application**.
7. Click **Debug**.  
The sample now runs in the debug mode.
8. Open the `request_news.html` file found in **samples > cloudfoundry > palette > http > RequestResponse**.
9. Click the **Get News from Wiki!** button to request headlines from the associated web page.

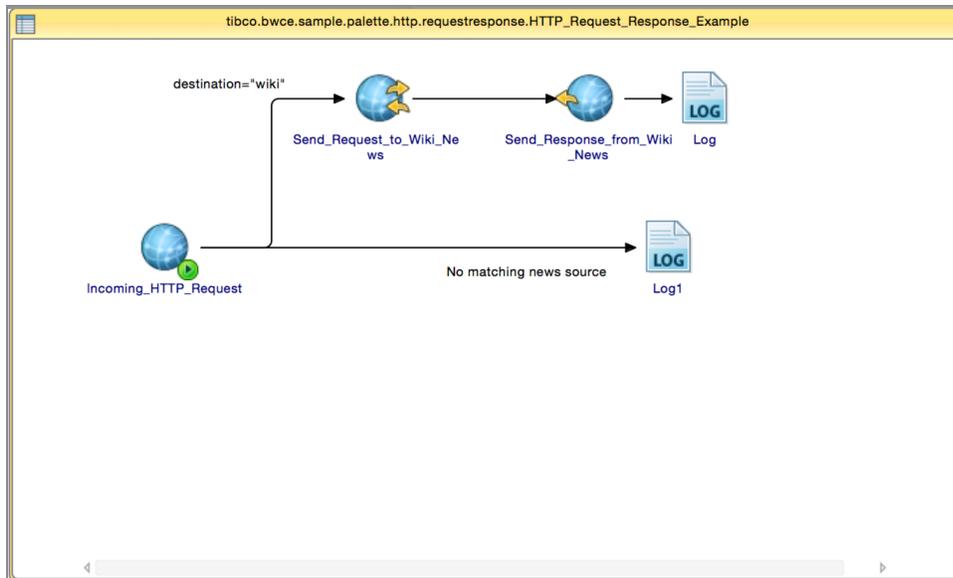
## Result

The Wiki web page displays in your default browser. The message `Response sent successfully!!` is printed on the TIBCO Business Studio for BusinessWorks console.

# Understanding the Configuration

The following shared resources are defined in the project:

- The Incoming HTTP Request process starter listens on the connection specified in `ListeningHTTPConnection.httpConnResource`. The `request_news.html` file contains a form and clicking the **Get News from Wiki!** button sends the corresponding text string to the Incoming HTTP Request activity.
- The conditional transition routes the request to the `Send HTTP Request` activity, which sends the request to the host using the `ListeningHTTPConnection`. After receiving a response from the remote site, the `Send HTTP Response` activity closes the HTTP Connection established by the Incoming HTTP Request process starter.
- An internet connection is required for the sample to connect to Wiki News. The `ListeningHTTPConnection` listens on `BW.CLOUD.PORT` (default value 8080).



## Testing the HTTP Request Response Sample

### Before you begin

- The Cloud Foundry environment updated with TIBCO BusinessWorks Container Edition buildpack.
- The cf command line interface tool installed on your machine.

## Setting the Default Application Profile

### Procedure

1. From the Project Explorer expand the **tibco.bwce.sample.palette.http.RequestResponse.application**.
2. Expand the **Package Units** folder and click **Properties**.
3. Select the **PCF** profile and **Set As Default**.

If you do not have a profile named **PCF**, click **New Profile** to create a profile and name it **PCF**.



## Generating an Application Archive File

### Procedure

1. Expand the Package Unit and select **Overview**.
2. In the **Overview** window select **Export Application for Deployment**.
3. Enter the location of your EAR file.

## Deploying the HTTP Request Response Sample

The following steps describe how to deploy the application to your Cloud Foundry environment.

### Procedure

1. Generate the EAR file.
  - a. Select **tibco.bw.sample.palette.http.RequestResponse.application**
  - b. Navigate to the **Overview** tab
  - c. Click **Export Application for Deployment**.
  - d. Click **Finish**.
2. Log into your Cloud Foundry environment.
3. Copy the manifest.yml file from the **samples** directory to the directory that contains the EAR file.

4. Edit the `manifest.yml` file, set the value of the `PATH` variable to point to the location of the EAR file, and save the file.
5. Open up a command terminal and navigate to the directory where the EAR file and `manifest.yml` file are stored.
6. From the terminal, run `cf push -f manifest.yml`.
7. Copy the routable URL of the application once it starts executing.
8. Edit the `request_news.html` page found in **samples > palette > http > RequestResponse** directory.
9. Replace the URL `http://127.0.0.1:8080` with the routable URL of the application.
10. Click the **Get News from Wiki!** button to request headlines from the associated web page.

## Result

To see the log output use the command `cf logs application name --recent`.

If the application deploys successfully, you can see a similar output in the console log:

```
05:00:15.674 INFO [Thread-14] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300006: Started BW Application
[tibco.bwce.sample.palette.http.RequestResponse.application:1.0]
05:00:28.966 INFO [bwEngThread:In-Memory Process Worker-2]
c.t.b.p.g.L.t.b.s.p.h.R.Log - Response sent successfully!!
```

## JDBC Basic

This sample establishes a JDBC connection with a database and runs the `SELECT` and `UPDATE` queries on the tables. The main intention of the sample is to show how to use DataDirect or native third party jars in the TIBCO BusinessWorks Container Edition runtime and deploy it in the Cloud Foundry environment.

## Testing in TIBCO Business Studio for BusinessWorks

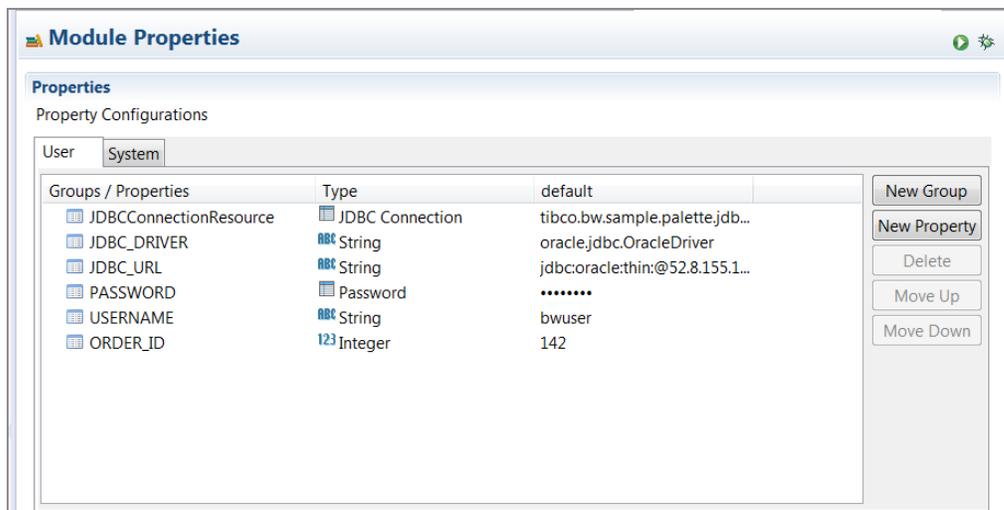
This section describes how to setup a JDBC Connection to Query and Update tables.

## Before you begin

- Access to a database.

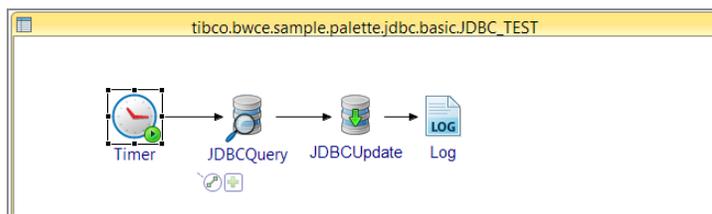
## Procedure

1. From the **File Explorer**, navigate to the samples directory and select **Container > cloudfoundry > palette > jdbc > Basic** and double-click `tibco.bwce.sample.palette.jdbc.Basic`.
2. From the **Project Explorer** expand the `tibco.bwce.sample.palette.jdbc.Basic` project.
3. Expand the **Module Descriptors** folder and double-click **Module Properties**.



The JDBC properties defined for the application are displayed in the dialog. Provide a valid username, password, and database URL to connect to your database.

4. Update the values under **Module properties** and save your project.
5. Expand the **Processes** directory and double-click **JDBC\_TEST.bwp**.



6. Verify your JDBC connection.
  - a. Expand the **Resources** directory.

- b. Double-click **JDBCConnection\_Oracle.jdbcResource**
  - c. In JDBC Driver section, click **Click Here to Set Preferences**.
  - d. Click **Browse** to choose the location of TIBCO DataDirect Driver and then click **Apply** and then click **OK**.
  - e. Click the **Test Connection** button to verify the connection.
7. Click **File > Save** to save the project.
  8. Click **Run > Debug** Configurations.
  9. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
  10. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bwce.sample.palette.jdbc.Basic.application**.
  11. Click **Debug**.  
This runs the sample in Debug mode.
  12. Click the **Terminate**  icon to stop the process.

## Understanding the Configuration

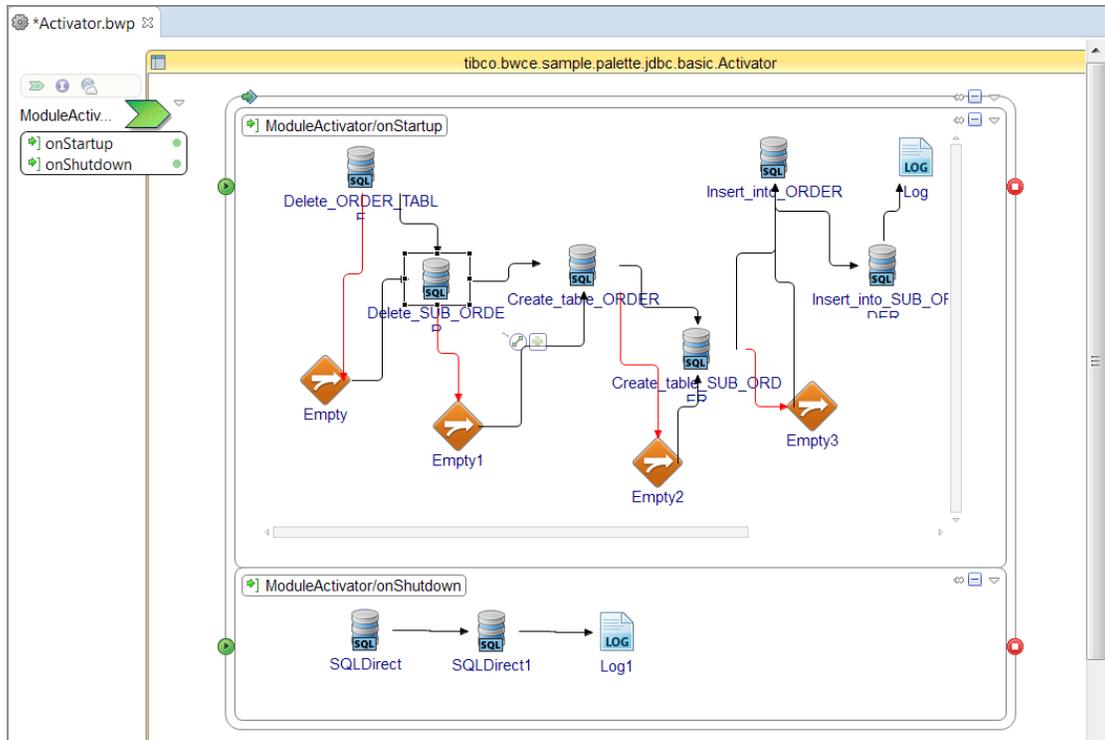
This project contains the following processes:

- The Activator Process (**Activator.bwp**)

The BusinessWorks Activator process is used to perform pre-processing and post-processing tasks when the application is started and stopped respectively.

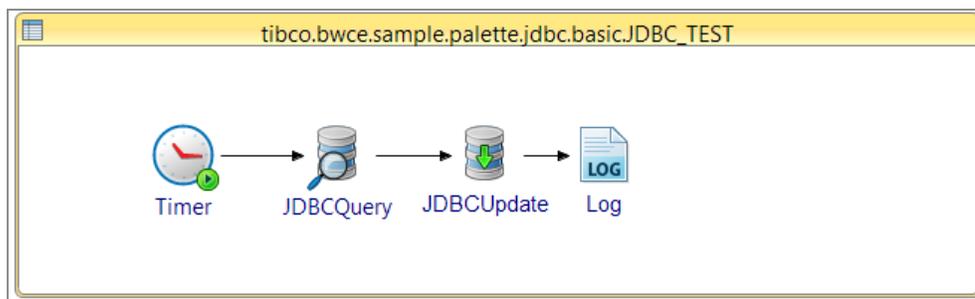
This contains a process service with two operations: **OnStartup** and **OnShutdown**.

The onStartup operation creates two database tables: order\_table and sub\_order and the onShutdown operation drops both the tables at the end of the application execution.



- The JDBC test process (**JDBC\_TEST.bwp**)

This process uses the JDBC Connection in the Resources folder to establish a connection to the database and to run SELECT and UPDATE queries. The process first queries the table for a specific record using the JDBC\_Query activity and then it updates another table using the JDBC\_Update activity. Both activities use prepared parameters as input. This shows the ability to run the same statement with multiple values by caching the statement at runtime.



# Testing the JDBC Basic Sample

## Before you begin

- Your Cloud Foundry environment updated with TIBCO BusinessWorks Container Edition buildpack.
- A web browser.
- User provided service (for Oracle) or Managed Service (for MySQL). Refer to <https://docs.cloudfoundry.org/> for the procedure for creating an user provided service or a managed service from the Cloud Foundry documentation.
- The cf command line interface tool installed on your machine.

## Creating a User Provided Service Instance (CUPS) for Oracle

Cloud Foundry has commands for creating and updating user-provided service instances (cups). Once created, user-provided service instances can be bound to an application. To interact with the Cloud Foundry instance, the command line interface, cf CLI is used. For more information on cf CLI, refer to <https://docs.cloudfoundry.org/cf-cli/>.

### Procedure

1. Log into your Cloud Foundry environment using the cf CLI.
2. Using the cf CLI, run the following command to create an Oracle cups service instance named oracledb.

```
cf cups oracledb -p "DB_URL, DB_USERNAME, DB_PASSWORD"
```

3. When prompted, specify values appropriate for your environment.

For example,

```
DB_  
URL>jdbc:tibcosoftwareinc:oracle://152.8.15.176:1521;ServiceName=or  
cl
```

```
DB_USERNAME>bwuser  
DB_PASSWORD>password
```

## Creating a Managed Service for MySQL

To install the MySQL managed service, see <http://docs.pivotal.io/>

After installation, follow these steps to create the managed service.

### Procedure

1. Log into your Cloud Foundry environment using the cf command-line interface.
2. In your cf CLI, run:

```
cf create-service p-mysql 100mb-dev mysql00b
```

## Provisioning the Drivers

Follow these steps to provision the database drivers:

### Procedure

1. Follow steps outlined in the **JDBC Connection** section under **Shared Resources** in the *Palette Reference* guide.

2. Copy all the contents

```
from TIBCO_HOME/bwce/version/config/drivers/shells/driverspecific  
runtime/runtime/plugins/
```

to the `/resources/addons/jars` folder in your temporary location.

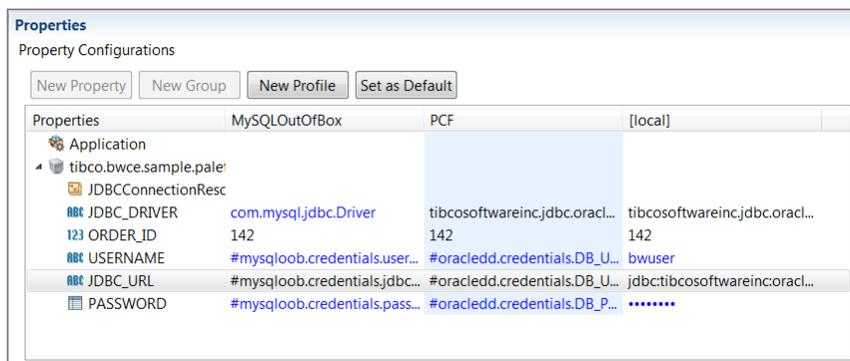
## Setting the Default Application Profile

### Procedure

1. From the **Project Explorer** expand the **tibco.bwce.sample.binding.jdbc.Basic.application**.
2. Expand the **Package Units** folder and click **Properties**.
3. Select a profile and **Set As Default**.

Use **PCF** profile to use with an Oracle database and select **MySQLOutOfBox** to use with a MySQL database. and

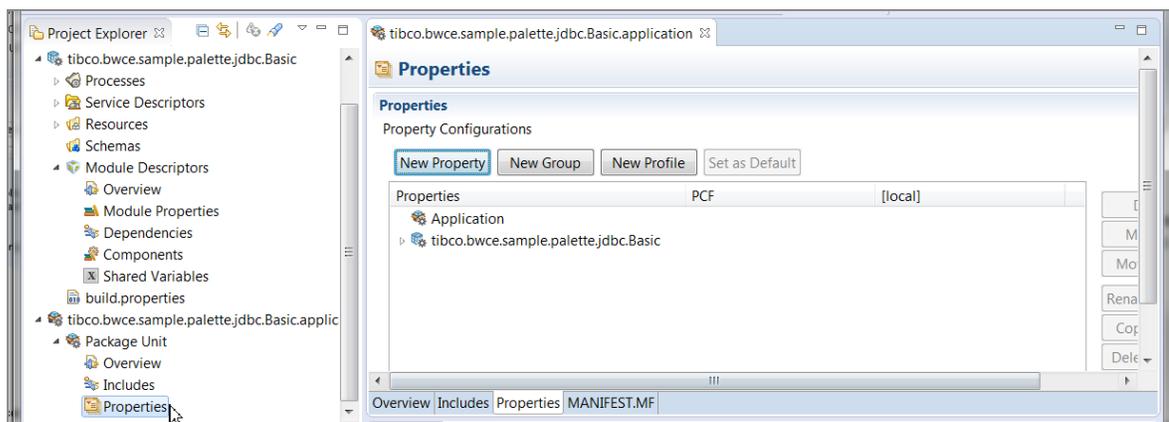
If you do not have a profile named **PCF** or **MySQLOutOfBox**, click **New Profile** to create a profile and name it accordingly.



## Modifying the Application Properties

### Procedure

1. From the Project Explorer navigate to **tibco.bwce.sample.binding.jdbc.Basic.application** (as shown in the image) and click **Properties**.



Assuming you have a running **oracledb** service instance in your Cloud Foundry, follow these steps to update values for the following properties in your PCF profile:

- USERNAME
- PASSWORD
- JDBC\_URL

The values use the format

`#<serviceName>.credentials.<parameter>#`

For example,

```
#oracledb.credentials.DB_USERNAME#
```

2. To update the **JDBC\_URL** property, first click the property name and then click the  icon.

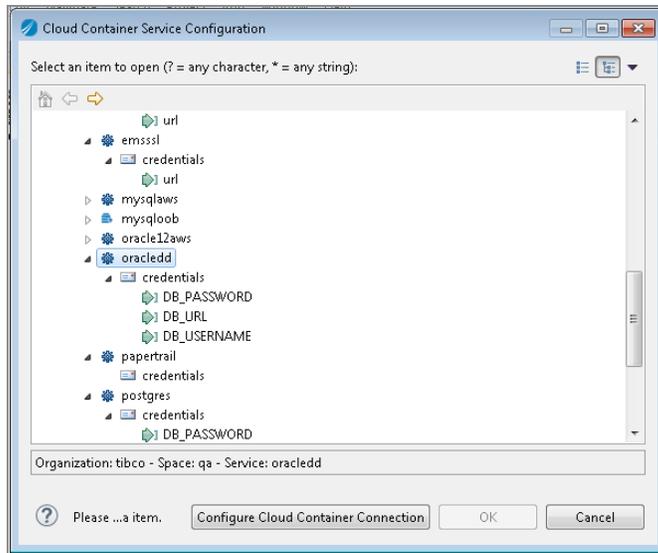
The **Cloud Container Service Configuration** window is displayed. This window initially contains no values.

3. Click **Configure Cloud Container Connection**.

The **Cloud Container Connection Configuration** window is displayed.

4. Specify the configuration properties and click **Test Connection** to connect to your Cloud Foundry environment to access values of the **oracledb** service instance.

If the connection is successful, variables are populated in the **Cloud Container Service Configuration** window as shown in the next figure.



5. Choose **DB\_URL** under **oracledd** and click **OK**.
6. Similarly update the **USERNAME** property. The **Cloud Container Service Configuration** window will now contain values and you need not click **Configure Cloud Container Connection** again.
7. Follow steps in [Password Property](#) to modify the **PASSWORD** value.
8. Click **Save**.

**Note:** To see the credentials for a managed service in the Cloud Container Service Configuration window, bind the managed service to a dummy application using the cf CLI command:

```
cf bind-service <appname><managed service name>
```

For example

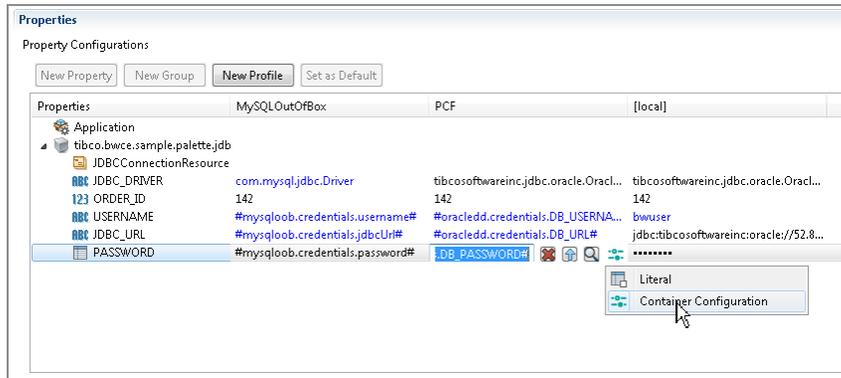
```
cf bind-service
tibco.bwce.sample.binding.jdbc.Basic.application oracledd
```

## Password Property

Follow these steps to modify the **DB\_PASSWORD** value:

## Procedure

1. Click the property name and then click the  icon and choose **Container Configuration** as shown in the next image.



The **Browse Container Configuration**  icon will now be visible.

2. Click **Browse Container Configuration**.  
The **Cloud Container Service Configuration** is displayed.
3. Choose **DB\_PASSWORD** under the **oracledd** service instance and click **OK**.
4. Click **Save**.

## Generating an Application Archive File

### Procedure

1. Expand the Package Unit and select **Overview**.
2. In the **Overview** window select **Export Application for Deployment**.
3. Enter the location of your EAR file.

## Deploying the JDBC Basic Application

The following steps describe how to deploy the application to your Cloud Foundry environment using the cf command line interface tool.

### Procedure

1. Log into your Cloud Foundry environment using the `cf` command line interface.
2. Copy the manifest file from the samples directory to the location where you placed your EAR file in the previous section.
3. Edit the **path** variable in the `manifest.yml` file. Set the **path** to the location of the EAR file. Check the user provided or managed service name, it should match with the service in your Cloud Foundry environment. Save the file.

As the manifest file contains the service name, the **oracledb** service automatically binds to this application at deployment.

4. In your Cloud Foundry environment change to the directory where you placed your EAR file.
5. Run the following command:

```
cf push -f manifest.yml
```

To see the log output use the command `cf logs <application name> --recent`. For example,

```
cf logs tibco.bwce.sample.palette.jdbc.Basic.application --recent
```

If the application deploys successfully, you can see a similar output in the console log.

```
05:16:00.763 INFO [bwEngThread:In-Memory Process Worker-5]  
c.t.b.p.g.L.t.b.s.p.jdbc.Basic.Log - Records Updated By JDBCUpdate  
Activity: 1
```

## JMS Basic

JMS Basic sample uses the JMS Queue Receiver and Reply to JMS Message activities to receive and respond to JMS messages. The JMS Queue Receiver activity is triggered by a

JMS Request Reply activity from another process. This sample demonstrates how TIBCO BusinessWorks Container Edition applications can send and receive messages from JMS destinations using JMS CUPS service in Cloud Foundry.

### Before you begin

- TIBCO Enterprise Message Server must be running.
- Ensure that the queues `jmsbasic.queue` and `reply.queue` have been created.

## Testing Locally

### Procedure

1. From the **File Explorer** navigate to the **samples** folder and select **Container > cloudfoundry > palette > jms > Basic** and double-click **tibco.bwce.sample.palette.jms.Basic**. For more information, see [Accessing Samples](#).
2. From the **Project Explorer** expand the **tibco.bwce.sample.palette.jms.Basic** project.
3. Verify your TIBCO Enterprise Message Service connection.
  - a. Completely expand the **Resources** directory.
  - b. Double-click **JMSConnectionResource.jmsConnResource**.
  - c. Click the **Test Connection** button to verify the connection.
4. From the **Project Explorer** expand the **Processes** folder and the **QueueReceiver.bwp** and **RequestReply.bwp** processes within that folder.
5. From the Debug Configuration wizard expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications.
7. Select the checkbox next to **tibco.bwce.sample.palette.jms.Basic.application**.
8. Click **Debug**.

The sample now runs in the debug mode.

The console window shows engine messages similar to:

```

00:48:05.773 INFO [Framework Event Dispatcher: Equinox Container:
70dfaf13-6384-0015-159e-c83afd447683]
com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300021: All
Application dependencies are resolved for Application
[tibco.bwce.sample.palette.jms.Basic:1.0]
00:48:07.518 INFO [Thread-43] com.tibco.thor.frwk.Application -
TIBCO-THOR-FRWK-300006: Started BW Application
[tibco.bwce.sample.palette.jms.Basic:1.0]
00:48:07.523 INFO [Framework Event Dispatcher: Equinox Container:
70dfaf13-6384-0015-159e-c83afd447683]
com.tibco.thor.frwk.Application - Started by BusinessStudio,
ignoring .enabled settings.
00:48:07.621 INFO [bwEngThread:In-Memory Process Worker-1]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogBeforeReq - Sending The Request
00:48:07.645 INFO [bwEngThread:In-Memory Process Worker-2]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogAfterReply - ***** JMSRequestReply
Received the Response ***** Sending A Reply Back For Request
Message Received *****
00:48:07.645 INFO [bwEngThread:In-Memory Process Worker-3]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogReceiveMessage - Received a Queue
Message. Request = Sending a Request JMS Message. JMSMessageID =
ID:EMS-SERVER.95A15639F97F35:1

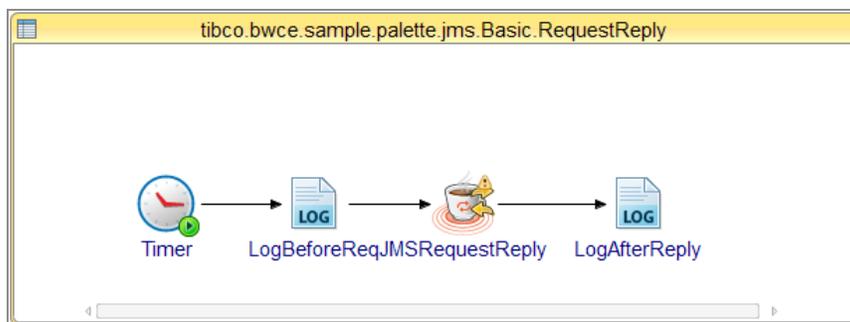
```

## Understanding the Configuration

The project uses two processes

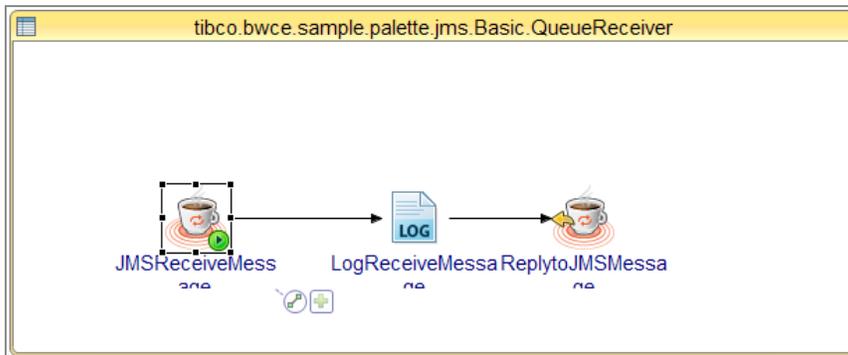
- The JMS Request Reply activity (**RequestReply.bwp**)

The JMS Request Reply activity sends a message to a queue and waits for the response by setting a **replyTo** destination to the message.



- The JMS QueueReceiver activity (**QueueReceiver.bwp**)

The JMS QueueReceiver activity in the second process receives the message from the queue and replies by setting the **correlation ID** and **replyTo** from the incoming message.



## Testing the JMS Basic Sample

### Before you begin

- The Cloud Foundry environment updated with TIBCO BusinessWorks Container Edition buildpack.
- A user provided service instance for TIBCO EMS Server. Read the procedure for creating an user provided service from the Cloud Foundry documentation. For more information, see <https://docs.cloudfoundry.org/>.
- The cf command line interface tool installed on your machine.

## Creating User Provided Service Instance (CUPS) for the TIBCO EMS Server

Cloud Foundry has commands for creating and updating user-provided service instances (cups). Once created, user-provided service instances can be bound to an application. To interact with the Cloud Foundry instance, the command line interface, cf CLI is used.

### Procedure

1. Log into your Cloud Foundry environment using the cf CLI.
2. Using the cf CLI, run

```
cf cups ems -p "url"
```

- When prompted, specify the value of the TIBCO EMS Server.

For example,

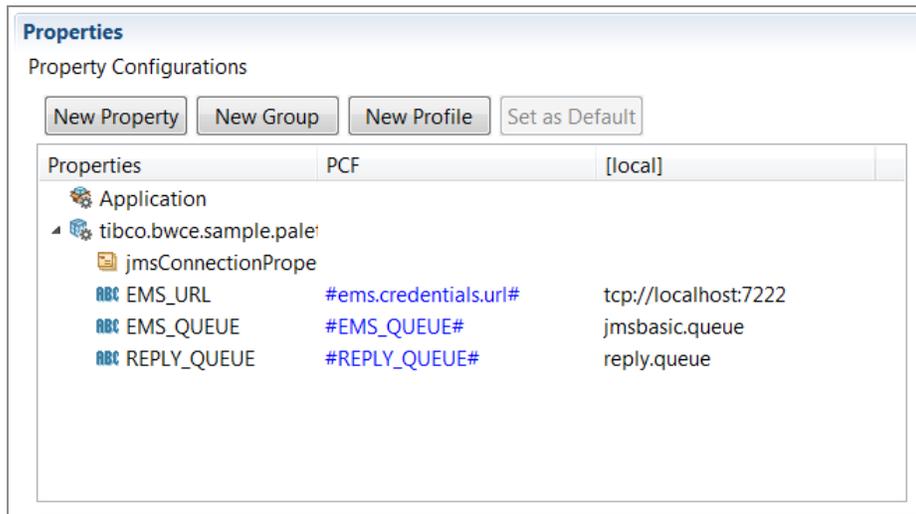
```
url> tcp://154.136.28.123.7222
```

## Setting the Default Application Profile

### Procedure

- From the Project Explorer expand the **tibco.bwce.sample.palette.jms.Basic.application**.
- Expand the **Package Units** folder and click **Properties**.
- Select the **PCF** profile and **Set As Default**.

If you do not have a profile named **PCF**, click **New Profile** to create a profile and name it **PCF**.



## Generating an Application Archive File

### Procedure

1. Expand the Package Unit and select **Overview** .
2. In the **Overview** window select **Export Application for Deployment**.
3. Enter the location of your EAR file.

## Deploying the JMS Basic Application

The following steps describe how to deploy the application to your Cloud Foundry environment.

### Procedure

1. Log into your Cloud Foundry environment using the `cf` command-line interface.
2. Copy the manifest file from the samples directory to the location where you placed your EAR file in the previous section.
3. Edit the **path** variable in the `manifest.yml` file. Set the path to the location of the EAR file and **Save** the file.

As the manifest file contains the **service** name, the **ems** service automatically binds to this application at deployment.

4. In your Cloud Foundry environment change to the directory where you placed your EAR file.
5. Run the following command:

```
cf push -f manifest.yml
```

To see the log output use the command `cf logs <application name> --recent`.

For example,

```
cf logs tibco.bwce.sample.palette.jms.Basic.application --recent
```

If the application deploys successfully, you can see a similar output in the console log.

```
00:48:07.621 INFO [bwEngThread:In-Memory Process Worker-1]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogBeforeReq - Sending The Request
00:48:07.645 INFO [bwEngThread:In-Memory Process Worker-2]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogAfterReply - ***** JMSRequestReply
Received the Response *****=** Sending A Reply Back For Request
Message Received *****
00:48:07.645 INFO [bwEngThread:In-Memory Process Worker-3]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogReceiveMessage - Received a Queue
Message. Request = Sending a Request JMS Message. JMSMessageID =
ID:EMS-SERVER.95A15639F97F35:1
```

## Sending and Receiving Messages Using EMS-SSL

This sample demonstrates how to design and run a TIBCO BusinessWorks Container Edition application that connects to an EMS Server via SSL in your Cloud Foundry environment.

### Before you begin

- TIBCO Enterprise Message Service configured with SSL must be running.
- Ensure the queue `emssl.queue` has been created.

## Testing in TIBCO Business Studio for BusinessWorks

### Procedure

1. From the **File Explorer**, navigate to the samples directory, select **Container >**

**cloudfoundry > palette > jms > SSL** and double-click **tibco.bwce.sample.palette.jms.SSL**.

2. From the **Project Explorer** expand the **tibco.bwce.sample.palette.jms.SSL.application** project.
3. Verify your JDBC connection.
  - a. Expand the **Resources** directory.
  - b. Double-click **JMSConnectionResource.jmsConnResource**.
  - c. Click the **Test Connection** button to verify the connection.
4. From the **Project Explorer** expand the **Processes** folder and the **GetQueueMessage.bwp** project within that folder.
5. From the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications.
7. Select the checkbox next to **tibco.bwce.sample.palette.jms.SSL.application.application**.
8. Click **Debug**.

The sample now runs in the debug mode.

The console window shows engine messages similar to:

```
16:08:26.242 [PVM:In-Memory STWorkProcessor:1] INFO
c.t.b.p.g.L.t.b.p.j.G.Log - Message=Sending 5 Queue Messages
16:08:26.292 [PVM:In-Memory STWorkProcessor:1] INFO
c.t.b.p.g.L.t.b.p.j.G.Log1 - Message=Finished sending 5 Queue
messages. Getting 5 Queue messages...
16:08:26.303 [PVM:In-Memory STWorkProcessor:3] INFO
c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 1.
Message = This is message number 1
16:08:26.307 [PVM:In-Memory STWorkProcessor:5] INFO
c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 2.
Message = This is message number 2
16:08:26.311 [PVM:In-Memory STWorkProcessor:1] INFO
c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 3.
Message = This is message number 3
```

```

16:08:26.315 [PVM:In-Memory STWorkProcessor:3] INFO
c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 4.
Message = This is message number 4
16:08:26.318 [PVM:In-Memory STWorkProcessor:5] INFO
c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 5.
Message = This is message number 5

```

## Understanding the Configuration

The **JMS Send Message** activity sends the messages with the message style set to Queue.

The **Get JMS Queue Message** activity receives the JMS messages from a Queue in a loop. The activity is placed in a group and the group action is set to Repeat.

The condition loops five times and receives five messages.

### JMS Connection for EMS SSL

For this sample the EMS is configured with SSL, and uses a SSL Client Resource that in turn uses a KeyStore provider Resource. The certificate **truststore.jks** used by KeyStore Provider Resource is packaged as part of the application.

### EMS SSL URL

Property Configurations		
System		
Groups / Properties	Type	Value
SSL_QUEUE	String	emssl.queue
EMS_URL	String	ssl://154.60.120.1175:7243

### JMS Connection with SSL Client

**Basic Configuration**  
 Connection Factory Type:   
 Messaging Style:   
 Provider URL:    
 Click test connection  
[Click here to set preferences](#)

---

**Security**  
**Advance Configuration**  
 Auto-generate Client ID:   
 Client ID:

---

**SSL**  
 Confidentiality   
 SSL Client:

**Keystore**  
 Provider:   
 URL:     
 Password:    
 Type:   
 Refresh Interval:    **Milliseconds**

## Testing the EMS Secure Sample

### Before you begin

- The Cloud Foundry environment updated with TIBCO BusinessWorks Container Edition buildpack.
- User provided service for EMS SSL. Read the procedure for creating an user provided service from the Cloud Foundry documentation. For more information, see <https://docs.cloudfoundry.org/>.

## Creating User Provided Service Instance (CUPS) for the TIBCO EMS Server

Cloud Foundry has commands for creating and updating user-provided service instances (cups). Once created, user-provided service instances can be bound to an application. To interact with the Cloud Foundry instance, the command line interface, cf CLI is used.

### Procedure

1. Log into your Cloud Foundry environment using the cf CLI.
2. Using the cf CLI, run

```
cf cups emssql -p "url"
```

3. When prompted, specify the value of the TIBCO EMS Server.

For example,

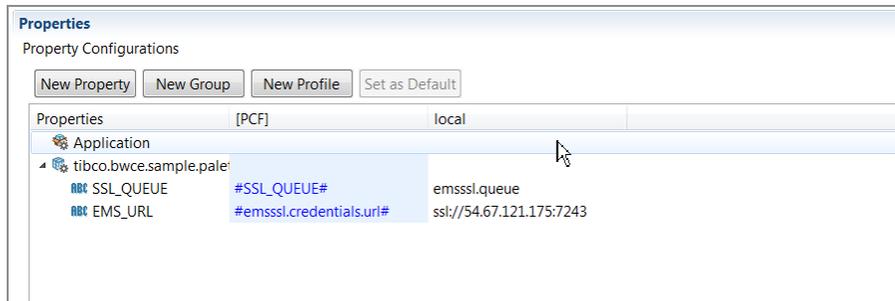
```
url> ssl://152.188.28.123.7227
```

## Setting the Default Application Profile

### Procedure

1. From the Project Explorer expand the **tibco.bwce.sample.palette.jms.SSL.application**.
2. Expand the **Package Units** folder and click **Properties**.
3. Select the **PCF** profile and **Set As Default**.

If you do not have a profile named **PCF**, click **New Profile** to create a profile and name it **PCF**.



## Generating an Application Archive File

### Procedure

1. Expand the Package Unit and select **Overview** .
2. In the **Overview** window select **Export Application for Deployment**.
3. Enter the location of your EAR file.

## Deploying the EMS Secure Application

The following steps describe how to deploy the application to your Cloud Foundry environment using the cf command line interface tool.

### Procedure

1. Log into your Cloud Foundry environment using the cf command-line interface.
2. Copy the manifest file from the samples directory to the location where you placed your EAR file in the previous section.
3. Edit the **path** variable in the `manifest.yml` file. Set the path to the location of the EAR file and
4. **Save** the file.

As the manifest file contains the service name, the **emssl** service automatically binds to this application at deployment.

5. In your Cloud Foundry environment change to the directory where you placed your EAR file.

## 6. Run the command

```
cf push -f manifest.yml
```

To see the log output use the command `cf logs <application name> --recent`

```
cf logs tibco.bwce.sample.palette.jms.SSL.application --recent
```

If the application deploys successfully, you can see a similar output in the console log.

```
16:08:26.311 [PVM:In-Memory STWorkProcessor:1] INFO
c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 3.
Message = This is message number 3
16:08:26.315 [PVM:In-Memory STWorkProcessor:3] INFO
c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 4.
Message = This is message number 4
16:08:26.318 [PVM:In-Memory STWorkProcessor:5] INFO
c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 5.
Message = This is message number 5
```

# JMS Basic With Spring Cloud Config Server

JMS Basic with Spring Cloud Config sample uses the JMS Queue Receiver and Reply to JMS Message activities to receive and respond to JMS messages. The JMS Queue Receiver activity is triggered by a JMS Request Reply activity from another process. The main intention of the sample is to demonstrate how to attach your TIBCO BusinessWorks Container Edition application running in your Cloud Foundry environment to an application configuration management server like Spring Cloud Config Server to externalize configuration management properties.

## Before you begin

- TIBCO Enterprise Message Service must be running.

- Using TIBCO Enterprise Message Service administration, create `springconfig.queue` before executing the sample.

## Testing in TIBCO Business Studio for BusinessWorks

### Procedure

1. From the **File Explorer**, navigate to the samples directory, select **Container > cloudfoundry > palette > jms > SpringCloudConfigtibco.bwce.sample.palette.jms.SpringCloudConfig**. For more information, see [Accessing Samples](#).
2. In Project Explorer expand the **tibco.bwce.sample.palette.jms.SpringCloudConfig** project.
3. Verify your TIBCO Enterprise Message Service connection.
  - a. Expand the **Resources** directory.
  - b. Double-click **JMSConnectionResource.jmsConnResource**.
  - c. In the Basic Configuration dialog, click the **Test Connection** button to verify the connection.
4. Expand the **Processes** directory and check **QueueReceiver** and **RequestReply** processes.
5. Click **Run > Debug**.
6. From the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bwce.sample.palette.jms.SpringCloudConfig.application**.
8. Click **Debug**.

The sample now runs in the debug mode.

### Result

The console window shows messages similar to:

```

00:48:05.773 INFO [Framework Event Dispatcher: Equinox Container:
70dfaf13-6384-0015-159e-c83afd447683] com.tibco.thor.frwk.Application -
TIBCO-THOR-FRWK-300021: All Application dependencies are resolved for
Application [tibco.bwce.sample.palette.jms.Basic:1.0]
00:48:07.518 INFO [Thread-43] com.tibco.thor.frwk.Application - TIBCO-
THOR-FRWK-300006: Started BW Application
[tibco.bwce.sample.palette.jms.Basic:1.0]
00:48:07.523 INFO [Framework Event Dispatcher: Equinox Container:
70dfaf13-6384-0015-159e-c83afd447683] com.tibco.thor.frwk.Application -
Started by BusinessStudio, ignoring .enabled settings.
00:48:07.621 INFO [bwEngThread:In-Memory Process Worker-1]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogBeforeReq - Sending The Request
00:48:07.645 INFO [bwEngThread:In-Memory Process Worker-2]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogAfterReply - ***** JMSRequestReply Received
the Response ***** Sending A Reply Back For Request Message Received
*****
00:48:07.645 INFO [bwEngThread:In-Memory Process Worker-3]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogReceiveMessage - Received a Queue Message.
Request = Sending a Request JMS Message. JMSMessageID = ID:EMS-
SERVER.95A15639F97F35:1

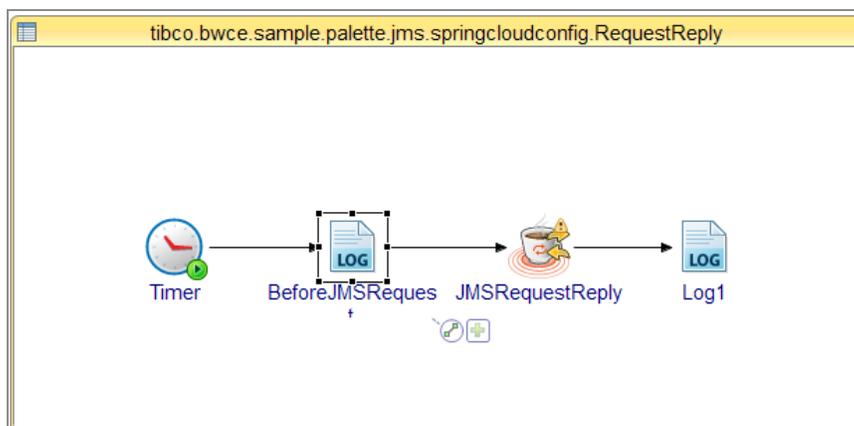
```

## Understanding the Configuration

The project uses two processes

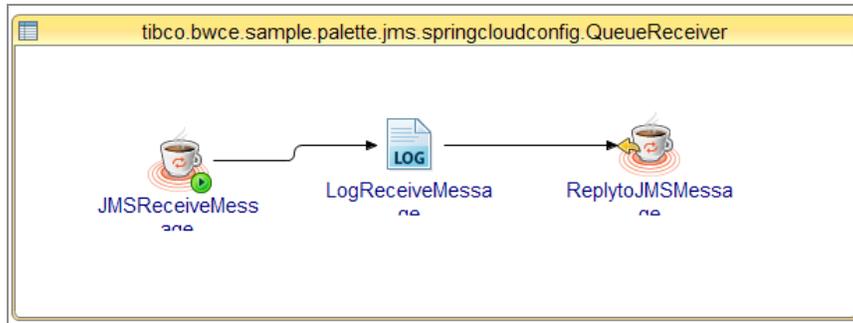
- The JMS Request Reply Process (**RequestReply.bwp**)

The JMS Request Reply activity is used to send a message to a queue and wait for the response by setting a replyTo destination to the message.



- The JMS QueueReceiver activity (**QueueReceiver.bwp**)

The JMS QueueReceiver activity in the second process receives the message from the queue and replies back by setting the correlation ID and replyTo from the incoming message.



## Testing the JMS Basic Sample with Spring Cloud Config Server

### Before you begin

- The Cloud Foundry environment updated with TIBCO BusinessWorks Container Edition buildpack.
- User provided service for TIBCO Enterprise Message Service.
- User provided service for Spring Cloud Config or a managed Spring Cloud Config service. For more information, see <https://docs.pivotal.io/spring-cloud-services/config-server/>.
- Read the procedure for creating an user provided service from the Cloud Foundry documentation. For more information, see <https://docs.cloudfoundry.org/>.
- The cf command line interface tool installed on your machine.

Config Server for Cloud Foundry is an externalized application configuration service which gives you a central place to manage an application's external properties across all environments. Spring Cloud Config Server is used for this purpose, that provides HTTP, resource-based API for external configuration (name-value pairs, or equivalent YAML content). The default backend storage repository is a Git Repo that stores the application configuration files. For running this sample, its assumed that the spring cloud config server is running with Git Repo backend and that the configuration files can be retrieved as a

client.

In this example, `foo` maps to the application name and `default` maps to your Application Profile or Environment.

```
curl http://54.67.121.175:8888/foo/default
{"name":"jmsspringcloud","profiles":
["default"],"label":null,"version":"0fd3f399e2fb5de96aa07a130d59fbe5e032
32ed","propertySources":[{"name":"https://github.com/TIBCOSoftware/bwcf-
spring-config-server-data/jmsspringcloud-default.properties","source":
{"EMS_QUEUE":"springconfig.queue","MESSAGE":"This message is from GIT
repo","EMS_URL":"tcp://localhost:7222","REPLY_
QUEUE":"springconfigreply.queue"}]}}
```

## Creating User Provided Service Instance (CUPS) for Spring Cloud Config

Cloud Foundry has commands for creating and updating user-provided service instances (cups). Once created, user-provided service instances can be bound to an application. To interact with the Cloud Foundry instance, the command line interface, cf CLI is used.

### Procedure

1. Log into your Cloud Foundry environment using the cf CLI.
2. Using the cf CLI, run `cf cups`.

```
cf cups spring-cloud-config-server -p "url"
```

3. When prompted, specify values appropriate for your environment.

For example,

```
url> tcp://localhost:8888
```

## Creating User Provided Service Instance (CUPS) for TIBCO EMS Server

Cloud Foundry has commands for creating and updating user-provided service instances (cups). Once created, user-provided service instances can be bound to an application. To interact with the Cloud Foundry instance, the command line interface, cf CLI is used. For more information on cf CLI, refer to <https://docs.cloudfoundry.org/cf-cli/>

### Procedure

1. Log into your Cloud Foundry environment using the cf CLI.
2. In your cf CLI, run

```
cf cups ems -p "url"
```

3. When prompted, specify values appropriate for your environment.

For example,

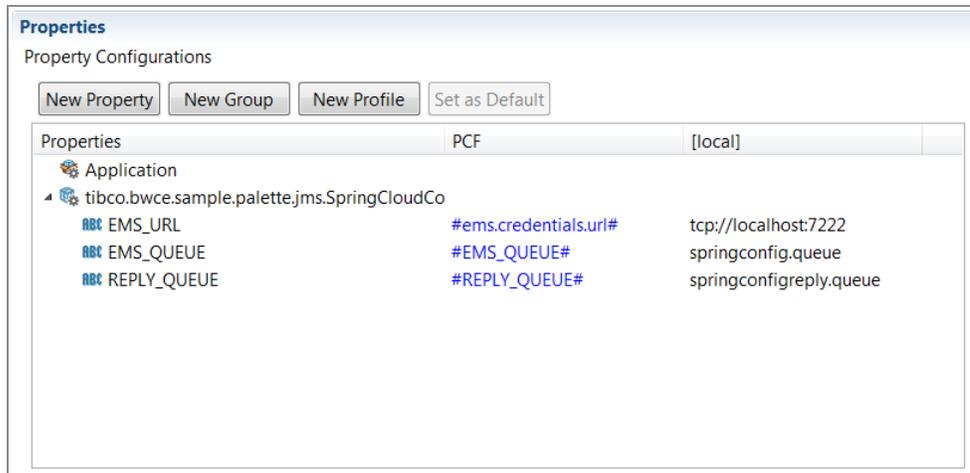
```
url> tcp://155.188.28.123.7222
```

## Setting the Default Application Profile

### Procedure

1. From the Project Explorer expand the **tibco.bwce.sample.palette.jms.SpringCloudConfig.application**.
2. Expand the **Package Units** folder and click **Properties**.
3. Select the **PCF** profile and **Set As Default**.

If you do not have a profile named **PCF**, click **New Profile** to create a profile and name it **PCF**.



## Generating an Application Archive File

### Procedure

1. Expand the Package Unit and select **Overview** .
2. In the **Overview** window select **Export Application for Deployment**.
3. Enter the location of your EAR file.

## Deploying the Application

The following steps describe how to deploy the application to your Cloud Foundry environment using the cf command line interface tool.

### Procedure

1. Log into your Cloud Foundry environment using the cf command-line interface.
2. Copy the manifest file from the samples directory to the location where you placed your EAR file in the previous section.
3. Edit the **path** variable in the `manifest.yml` file. Set the path to the location of the EAR file and Save the file.

As the manifest file contains the **service** name, the **spring-cloud-config-server**

service automatically binds to this application at deployment.

4. In your Cloud Foundry environment change to the directory where you placed your EAR file.
5. Run the following command:

```
cf push -f manifest.yml
```

To see the log output use the command `cf logs <application name> --recent`. For example,

```
cf logs tibco.bwce.sample.paLETTE.jms.SpringCloudConfig.application  
--recent
```

If the application deploys successfully, you can see a similar output in the console log.

```
10:11:19.026 INFO [bwEngThread:In-Memory Process Worker-1]  
c.t.b.p.g.L.t.b.s.p.j.S.BeforeJMSRequest - Sending a JMS Request  
Message  
10:11:19.456 INFO [bwEngThread:In-Memory Process Worker-3]  
c.t.b.p.g.L.t.b.s.p.j.S.LogReceiveMessage - JMSReceiver Received a  
Message  
10:11:19.716 INFO [bwEngThread:In-Memory Process Worker-2]  
c.t.b.p.g.L.t.b.s.p.j.S.Log1 - Received a Reply From JMSReceiver  
===*** Sending A Reply Back For Request Message Received *****
```

## Collecting Process Instance, Activity Instance, and Transition Statistics

In this sample, you will use the `tibco.bwce.sample.application.execution.event.subscribe` sample to collect statistics for process instances, activity instances, and transitions in the `tibco.bwce.sample.paLETTE.http.RequestResponse` sample project.

When working this sample, any of the following application statistics can be collected.

### Process Instance Statistics

<b>Statistic</b>	<b>Description</b>
Application Name	Name of the application.
Application Version	Version of the application.
Module Name	Name of the TIBCO BusinessWorks Container Edition module.
Module Version	Version of the TIBCO BusinessWorks Container Edition module.
Component Process Name	Name of process configured to a component. If the process is a non in-lined sub process, this could be empty.
Job ID	Job ID of the process.
Parent Process Name	If the process is an in-lined sub process, the name of the parent process.
Parent Process ID	If the process is an in-lined sub process, the instance ID of the parent process.
Process Name	Name fo the process.
Process Instance ID	Instance ID of the process.
Start Time	Process instance start time.
End Time	Process instance end time.
Elapsed Time	Elapsed time for a process is the total time taken by the process, including the elapsed time for all the activities executed for the process.
Eval Time	The Eval Time for a process instance is the total evaluation time (in milliseconds) for all the activities executed for the process instance.
Status	Status of process instance, for example: Completed or Faulted.

## Activity Instance Statistics

Statistic	Description
Application Name	Name of the application.
Application Version	Version of the application.
Module Name	Name of the TIBCO BusinessWorks Container Edition module.
Module Version	Version of the TIBCO BusinessWorks Container Edition module.
Activity Name	Name of the activity.
Process Name	Name of the process.
Process Instance ID	Instance ID of the process.
Start Time	When the activity instance started.
End Time	When the activity instance ended.
Eval Time	The time between the beginning and end of the evaluation period for the activity. If the activity completes in one step, the evalTime and elapsedTime would be the same. However, some activities, such as <b>Request</b> , <b>Reply</b> or <b>Wait for...</b> activities typically do not complete in one step.
Elapsed Time	Elapsed time of an activity is the time difference (in milliseconds) between start time and end time of the activity. Between the start and end time, control may get switched with other activities from other jobs. This is the time taken to run an activity plus all the delays in acquiring resources like engine threads, JDBC connections, network, and so on. The elapsed time is Eval Time plus the time taken for evaluating all the forward transitions from that particular activity.
Status	Status of activity, for example: Completed, Faulted or Canceled.

## Transition Statistics

Statistic	Description
Application Name	Name of the application.
Application Version	Version of the application.
Module Name	Name of the TIBCO BusinessWorks Container Edition module.
Module Version	Version of the TIBCO BusinessWorks Container Edition module.
Transition Name	Name of the transition
Process Name	Name of the process.
Process Instance ID	Instance ID of the process.
Component Process Name	Name of process configured to a component . If the process is a non in-lined subprocess, this could be empty.
Target Activity Name	Name of the activity the transition targets.

## Procedure

1. Import the sample into TIBCO Business Studio for BusinessWorks by right-clicking in the Project Explorer pane, and selecting **Import > Existing Studio Projects into Workspace**.
2. In the Import Projects window, ensure the option **Select root directory** field is selected, and specify the location of the **tibco.bwce.sample.application.execution.event.subscribe** sample. The sample is located at `BWCE_HOME/samples/Container/cloudfoundry/source/event-subscriber/tibco.bwce.sample.application.execution.event.subscriber`.
3. Click **Finish** to import the sample project.
4. In the **samples** directory, select **palette > http > RequestResponse** and double-click

**tibco.bwce.sample.palette.http.RequestResponse.zip.**

5. In **Project Explorer** expand the **tibco.bwce.sample.palette.http.RequestResponse** project.
6. Fully expand the **Processes** directory and double-click **HTTP\_Request\_Response\_Example.bwp**.
7. Click **Run > Debug Configurations**.
8. At the left hand tree of **Debug Configuration** wizard, expand **BusinessWorks Application** and select **BWApplication**.
9. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bwce.sample.palette.http.RequestResponse.application**.
10. Click the **Bundles** tab and ensure the following bundles in your workspace are selected:
  - **tibco.bwce.sample.application.execution.event.subscriber (1.0.0.qualifer)**
  - **tibco.bwce.sample.palette.http.RequestResponse (1.0.0.qualifer)**
  - **tibco.bwce.sample.palette.http.RequestResponse.application (1.0.0.qualifer)**
11. In the **Arguments** tab, add the "- Dbw.frwk.event.subscriber.instrumentation.enabled=TRUE" property in the **VM arguments** field.
12. Click **Debug**.  
This runs the sample in Debug mode.
13. Run lendpoints at the prompt in the **Console** tab to obtain the endpoint for the application.
14. Copy the endpoint URL of the application.
15. Open a browser window, and paste the endpoint URL into the address bar.
16. Click the **Terminate**  icon to stop the process.

**Result**

Details about process instances, activity instances, and transitions in **tibco.bwce.sample.palette.http.RequestResponse** are displayed on the **Console** tab in TIBCO Business Studio for BusinessWorks.

```
<>@BWEclipseAppNode>
ProcessInstance Auditing Event {
  Application
  Name:tibco.bwce.sample.palette.http.RequestResponse.application
  Application Version:1.0
  Module Name:tibco.bwce.sample.palette.http.RequestResponse
  Module Version:1.0.0.qualifier
  ProcessInstanceId:bw0a100
  Process Name:tibco.bwce.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
  State:SCHEDULED
}

ProcessInstance Auditing Event {
  Application
  Name:tibco.bwce.sample.palette.http.RequestResponse.application
  Application Version:1.0
  Module Name:tibco.bwce.sample.palette.http.RequestResponse
  Module Version:1.0.0.qualifier
  ProcessInstanceId:bw0a100
  Process Name:tibco.bwce.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
  State:STARTED
}

Activity Auditing Event {
  Application
  Name:tibco.bwce.sample.palette.http.RequestResponse.application
  Application Version:1.0
  Module Name:tibco.bwce.sample.palette.http.RequestResponse
  Module Version:1.0.0.qualifier
  ProcessInstanceId:bw0a100
  Process Name:tibco.bwce.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
  Activity Name:Incoming_HTTP_Request
  State:STARTED
}

Transition Auditing Event {
  Application
  Name:tibco.bwce.sample.palette.http.RequestResponse.application
  Application Version:1.0
  Module Name:tibco.bwce.sample.palette.http.RequestResponse
  Module Version:1.0.0.qualifier
  ProcessInstanceId:bw0a100
  Process Name:tibco.bwce.sample.palette.http.requestresponse.HTTP_
```

## Request\_Response\_Example

```

Activity Auditing Event {
  Application
  Name:tibco.bwce.sample.palette.http.RequestResponse.application
  Application Version:1.0
  Module Name:tibco.bwce.sample.palette.http.RequestResponse
  Module Version:1.0.0.qualifier
  ProcessInstanceId:bw0a100
  Process Name:tibco.bwce.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
  Activity Name:Incoming_HTTP_Request
  State:COMPLETED
}

```

```

Activity Auditing Event {
  Application
  Name:tibco.bwce.sample.palette.http.RequestResponse.application
  Application Version:1.0
  Module Name:tibco.bwce.sample.palette.http.RequestResponse
  Module Version:1.0.0.qualifier
  ProcessInstanceId:bw0a100
  Process Name:tibco.bwce.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
  Activity Name:Log1
  State:STARTED
}

```

```

22:25:19.463 INFO [bwEngThread:In-Memory Process Worker-1]
c.t.b.p.g.L.t.b.s.p.h.R.Log1 - No matching 'NEWS' source found.

```

```

Activity Auditing Event {
  Application
  Name:tibco.bwce.sample.palette.http.RequestResponse.application
  Application Version:1.0
  Module Name:tibco.bwce.sample.palette.http.RequestResponse
  Module Version:1.0.0.qualifier
  ProcessInstanceId:bw0a100
  Process Name:tibco.bwce.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
  Activity Name:Log1
  State:COMPLETED
}

```

```

ProcessInstance Auditing Event {
  Application

```

```
Name:tibco.bwce.sample.palette.http.RequestResponse.application
Application Version:1.0
Module Name:tibco.bwce.sample.palette.http.RequestResponse
Module Version:1.0.0.qualifier
ProcessInstanceId:bw0a100
Process Name:tibco.bwce.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
State:COMPLETED
```

✔ **Tip:** You can use this sample to build your own application statistics collection tool. Follow these steps to do this:

1. From the Project Explorer tab, select **tibco.bwce.sample.application.execution.event.subscriber > src > tibco.bwce.sample.application.execution.event.subscriber > BWEventSubscriber.java**.
2. Update `handleEvent(Event event)` method based on your use case.
3. Save your changes to the project.
4. Export the project as a plug-in by right clicking on **tibco.bwce.sample.application.execution.event.subscriber** and selecting **export > Export > Plug-in Development > Deployable plug-ins and fragments**.
5. In the Export wizard, ensure the `tibco.bwce.sample.application.execution.event.subscriber` project is selected, and specify a location to export the plug-in.
6. After the project has been exported as a plugin to the location you specified, locate the JAR file in the plugin folder, and copy paste the JAR to your buildpack. Refer to the section "The TIBCO BusinessWorks™ Container Edition Buildpack" topic of *Application Development* guide.

Your application statistics collection tool has been added to your run time environment. You can see the application statistics after you run the TIBCO BusinessWorks Container Edition application with this buildpacks.

# Samples for Docker

---

The following samples are available for your Docker environment:

- [REST Bookstore](#)
- [REST BookStore Consumption](#)
- [Implementing a JMS Service that Returns Information Based on Zip Codes](#)
- [Resilience4j](#)
- [Service Discovery](#)
- [HTTP Request Response](#)
- [JDBC Basic](#)
- [JMS Basic](#)
- [JMS Basic with Consul](#)
- [Sending and Receiving Messages Using EMS-SSL](#)
- [SOAP over HTTP](#)
- [Collecting Process Instance, Activity Instance, and Transition Statistics](#)

## REST Bookstore

This sample shows how to use the RESTful service to add, delete, update, and retrieve books from a bookstore. The sample uses the following HTTP methods for the Books REST resource:

- POST: Add books to the bookstore.
- GET: Get books from the bookstore.
- PUT: Update books in the bookstore.
- DELETE: Delete books from the bookstore.

# Testing in TIBCO Business Studio for BusinessWorks

## Before you begin

- Access to a locally running PostgreSQL database.
- A web browser.

## Procedure

1. From the **File Explorer**, navigate to the **samples > Container > docker > binding > rest > BookStore** folder and then double-click **tibco.bwce.sample.binding.rest.BookStore**.
2. From the **Project Explorer** expand the **tibco.bwce.sample.binding.rest.BookStore.application** project.
3. Expand **tibco.bwce.sample.binding.rest.Bookstore.application > Package Unit** folder. Provide valid values for the application properties. Provide a valid dbUserName, dbPassword and dbURL to connect to your locally running PostgreSQL database.
4. Set the **ApplicationProfile** to **local**. For more information, see [Setting the Default Application](#).
5. Verify your JDBC connection.
  - a. Expand the **Resources** directory.
  - b. Double-click **JDBCConnectionResource.jdbcResource**.
  - c. Click the **Test Connection** button to verify the connection.
6. Click **File > Save**.
7. From the **Project Explorer** expand the **Processes** folder and the **tibco.bwce.sample.binding.rest.BookStore** package within that folder.
8. Click **Run > Debug**.
9. From the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
10. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications.
11. Select the checkbox next to **tibco.bwce.sample.binding.rest.BookStore**.

12. Click **Debug**.

The sample now runs in the debug mode.

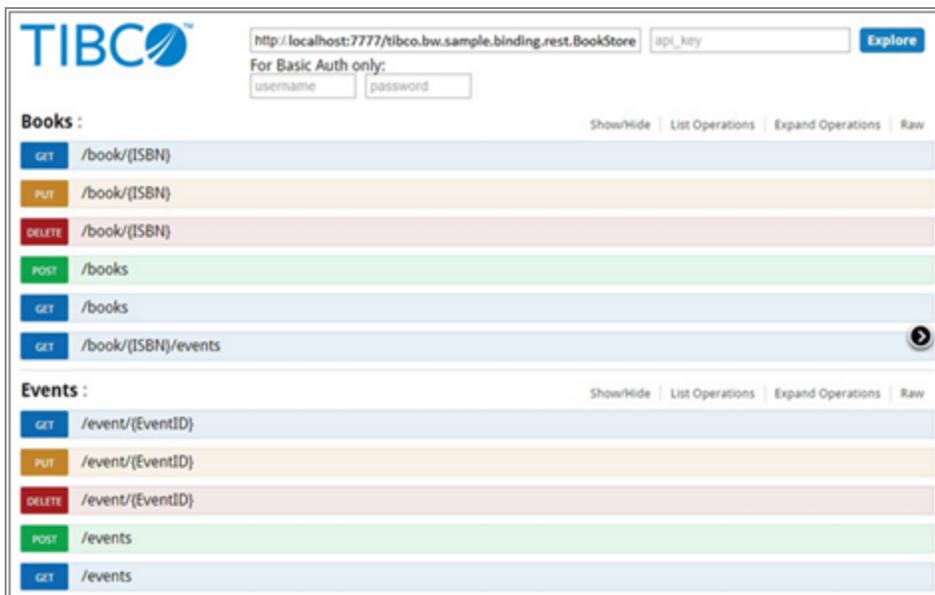
The console window shows engine messages similar to:

```
Started BW Application [tibco.bwce.sample.binding.rest.BookStore.
[tibco.bwce.sample.binding.rest.BookStore.application:1.0].
The OSGI command to list REST and Swagger URLs is lrestdoc, which
lists the discovery URL:[Application Name]:
tibco.bwce.sample.binding.rest.BookStore.application [Discovery
Url]:http://localhost:7777/tibco.bwce.sample.binding.rest.BookStore
.application
```

## 13. Launch the browser and open

`http://localhost:7777/tibco.bwce.sample.binding.rest.BookStore.application`.

The following page is displayed



14. Click any of the listed operations GET, PUT, PUT, or DELETE.

15. Click the Terminate  icon when you have completed using the listed operations.

## Result

The web page lists the following operations for Books and Events:

### Books

- POST books
- GET books
- GET book by ISBN
- PUT book by ISBN
- DELETE book by ISBN

## Events

- POST Events
- GET Events
- GET Event by EventID
- PUT Event by EventID
- DELETE Event by EventID

**GET books** returns an output similar to the following:

```
{
  "Book": [
    {
      "isbn": "0061122416",
      "name": "The Alchemist",
      "description": "Every few decades a book is published that changes
the lives of its readers forever. The Alchemist is such a book",
      "authorName": "Paul Coelho",
      "releaseDate": "2006-04-25",
      "vintage": true,
      "signed": true,
      "price": 11.9
    },
    {
      "isbn": "0071450149",
      "name": "The Power to Predict",
      "description": "How Real Time Businesses Anticipate Customer
Needs, Create Opportunities, and Beat the Competition",
      "authorName": "Vivek Ranadive",
      "releaseDate": "2006-01-26",
      "vintage": false,
      "signed": true,
      "price": 15.999
    }
  ]
}
```

```
]
}
```

**GET books** by ISBN returns an output similar to the following for the ISBN 0061122416:

```
{
  "isbn": "0061122416",
  "name": "The Alchemist",
  "description": "Every few decades a book is published that changes
the lives of its readers forever. The Alchemist is such a book",
  "authorName": "Paul Coelho",
  "releaseDate": "2006-04-25",
  "vintage": true,
  "signed": true,
  "price": 11.9
}
```

The books.log file is generated with the following information:

```
POST Books----->{"Book":[{"isbn":"1451648537","name":"Steve
Jobs","description":"Biography of Apple Co-Founder Steve
Jobs","authorName":"Walter Isaacson","releaseDate":"2012-10-
24","vintage":false,"signed":false,"price":21},
{"isbn":"0385537859","name":"Inferno","description":"Robert Langdon
returns in Dan Brown's latest fast paced action
thriller","authorName":"Dan Brown","releaseDate":"2013-05-
14","vintage":false,"signed":true,"price":14.09},
{"isbn":"0399103421","name":"The Godfather","description":"The Godfather
is an epic story of a New York's top mafia family, loyalty, and how men
of honor live in their own world, and die by their own
laws.","authorName":"Mario Puzo","releaseDate":"196903-
10","vintage":true,"signed":true,"price":50}]}
*****
GET Books----->{"Book":[{"isbn":"1451648537","name":"Steve
Jobs","description":"Biography of Apple Co-Founder Steve
Jobs","authorName":"Walter Isaacson","releaseDate":"2012-10-
24+05:30","vintage":false,"signed":false,"price":21},
{"isbn":"0385537859","name":"Inferno","description":"Robert Langdon
returns in Dan Brown's latest fast paced action
thriller","authorName":"Dan Brown","releaseDate":"2013-05-
14+05:30","vintage":false,"signed":true,"price":14.09},
{"isbn":"0399103421","name":"The Godfather","description":"The Godfather
is an epic story of a New York's top mafia family, loyalty, and how men
of honor live in their own world, and die by their own
laws.","authorName":"Mario Puzo","releaseDate":"1969-03-
```

```

10+05:30","vintage":true,"signed":true,"price":50}}]
*****
GET Book By ISBN----->{"isbn":"1451648537","name":"Steve
Jobs","description":"Biography of Apple Co-Founder Steve
Jobs","authorName":"Walter Isaacson","releaseDate":"2012-10-
24+05:30","vintage":false,"signed":false,"price":21}
*****
DELETE Book By ISBN----->"Deleted book with ISBN -
145164853
7"*****
GET Events By ISBN---->
{}*****

```

## Understanding the Configuration

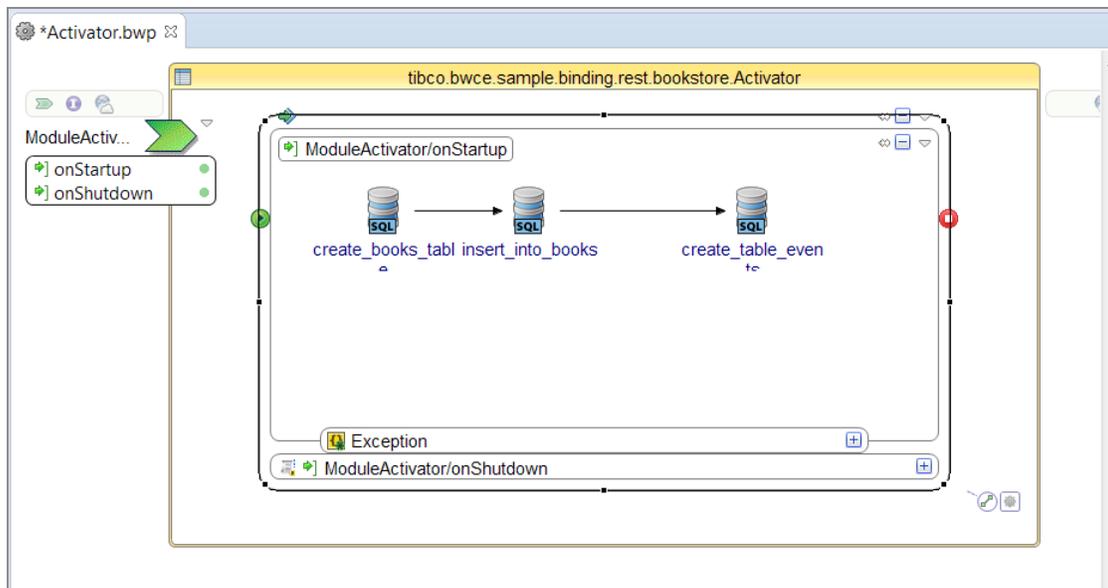
The project contains three processes (Activator, Books, and Events) and two subprocesses (BooksDB and EventsDB).

- The Activator Process (**Activator.bwp**)

The BW Activator process is used to perform pre-processing and post- processing tasks when the application is started and stopped respectively.

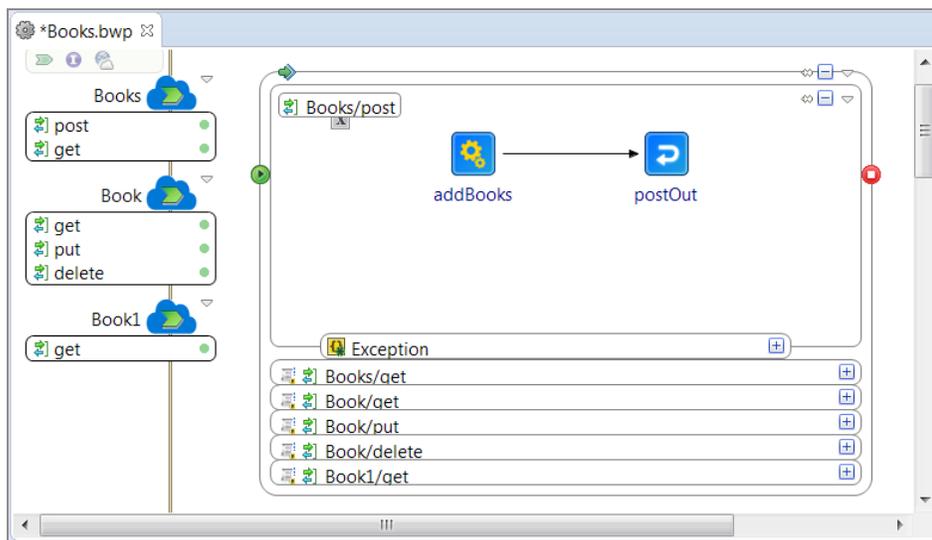
This contains a process service with two operations: OnStartup and OnShutDown.

The OnStartup process creates the Books and Events tables and inserts data into Books.



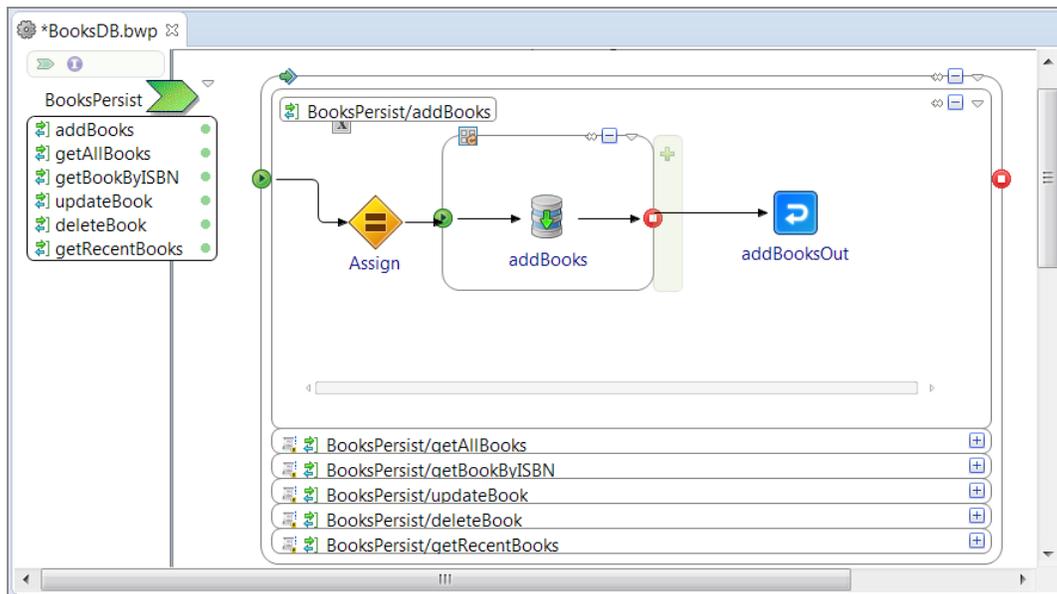
- The Books Process (**Books.bwp**)

In this process, the Book REST service performs the post, get, put, delete operations. It posts one or more books to the bookstore and retrieves all the books from the bookstore. For the post operation, a sub process is called to add books to the database. The addBooks operation is implemented in the BooksDB subprocess. The Book REST service performs three operations: getting a book from the bookstore by ISBN, updating a book in the bookstore using ISBN, and deleting a specific book from the bookstore using ISBN. ISBN is passed as a parameter to these operations.



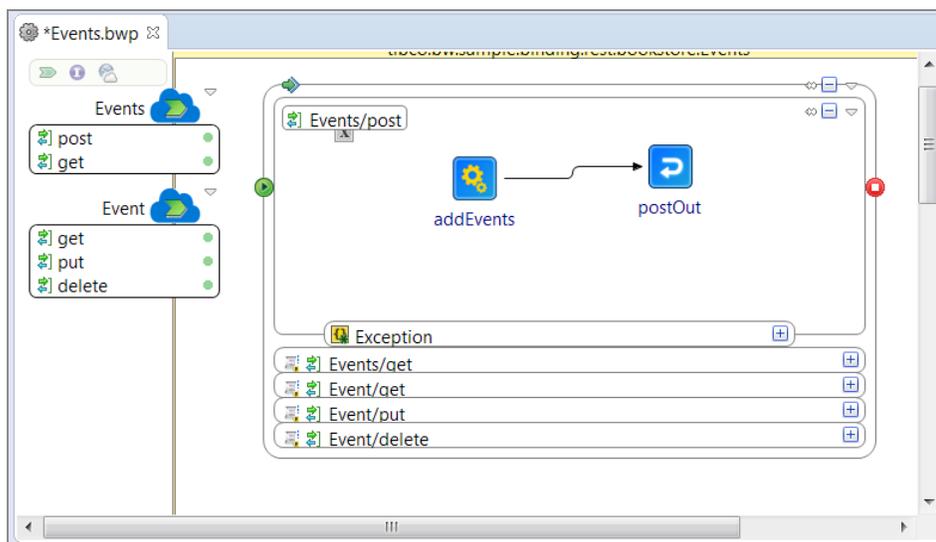
- The BooksDB subprocess (**BooksDB.bwp**)

The subprocess handles deletes, updates, retrievals of books from the database. The BooksPersist service implements following operations: adding books to database, getting all books from database, getting a book by ISBN, updating a book by ISBN, deleting a book by ISBN. Using BooksPersist/addBooks, books are added to the database.



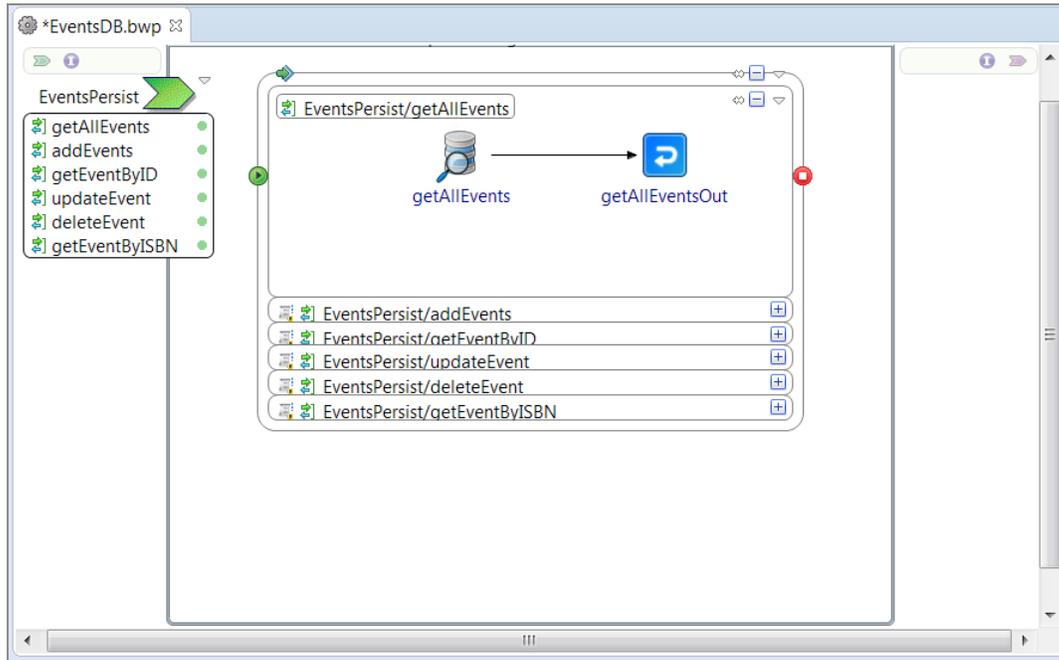
- The Events Process (**Events.bwp**)

This process is similar to the Books process and is used to add, delete, update, and retrieve events.



- The EventsDB subprocess (**EventsDB.bwp**)

This process is similar to the BooksDB process and is used to add, delete, update, and retrieve events from the database.



## Testing the REST Sample

This section describes the process of testing this sample in your Docker environment.

### Before you begin

- A web browser.
- TIBCO BusinessWorks Container Edition runtime base image.

Follow instructions in [Creating the TIBCO BusinessWorksTrademark\(TM\) Container Edition Application Docker Image](#) to create the runtime base image.

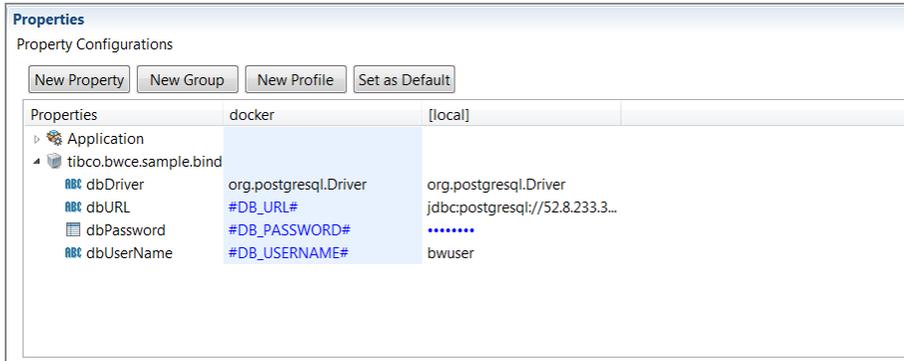
## Setting the Default Application Profile

### Procedure

1. From the Project Explorer expand the **tibco.bwce.sample.binding.rest.BookStore.application**.

2. Expand the **Package Units** folder and click **Properties**.
3. Select the **docker** profile and **Set As Default**.

If you do not have a profile named **docker**, click **New Profile** to create a profile and name it **docker**.



## Generating an Application Archive File

Follow these steps to generate the .EAR file:

### Procedure

1. Expand the Package Unit and select **Overview**.
2. In the **Overview** window select **Export Application for Deployment**.
3. Enter the location of your EAR file.

## Building the Application Image

### Procedure

1. Copy the Dockerfile from the samples directory to the location where you placed the EAR file.
2. From the docker terminal, navigate to the folder where the EAR and Dockerfile are stored.
3. In the Dockerfile, make sure the base image points to the TIBCO BusinessWorks

Container Edition runtime base image.

Also make sure the ear file path and name is correct.

```
FROM tibco/bwce:latest
MAINTAINER Tibco
ADD bwce-rest-bookstore-app.ear /
EXPOSE 8080
```

4. Type in below command on the docker terminal to generate the application image.

```
docker build -t bwce-rest-bookstore-app .
```

## Testing the Application Locally in a Docker Setup

### Procedure

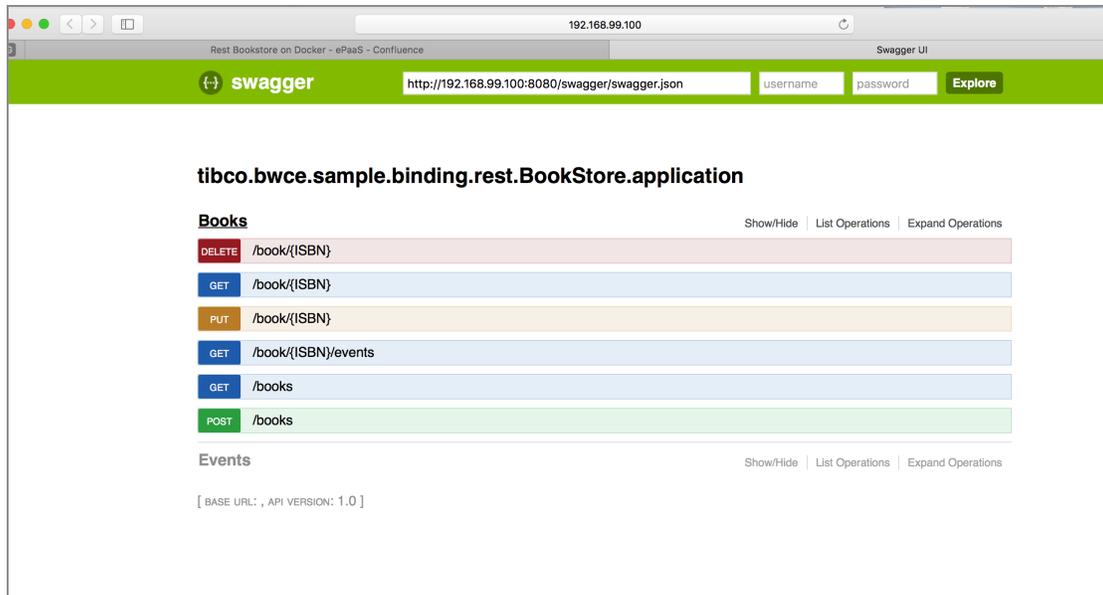
1. Run the following command on a docker terminal to run the application image:

```
docker run -i -p 8080:8080 -e "DB_
URL=jdbc:postgresql://52.18.253.34:5433/
postgres" -e "DB_USERNAME=bwuser" -e "DB_PASSWORD=password"
tibco/bwce-restbookstore-
app
```

2. Run the following command to view the logs to make sure that the application has started successfully.

```
docker logs <container name>
```

3. Get the docker machine IP.



## Testing the REST Service

### Procedure

1. Append `swagger` to the routable url to access the Swagger doc to test the REST Service.

For example, `docker machine ip:8080/swagger`

2. Expand the **Books** and **Events** headers to test the operations.

Click any of the operations, such as POST, GET, PUT, DELETE, and click **Try it out!**.

The screenshot shows a REST client interface with the following sections:

- Method:** GET
- Path:** /books
- Response Class (Status 200):** Model Schema
- Model Schema:**

```

{
  "isbn": "string",
  "name": "string",
  "description": "string",
  "authorName": "string",
  "releaseDate": "string",
  "vintage": true,
  "signed": true,
  "price": 0
}

```
- Response Content Type:** application/json
- Buttons:** Try it out!, Hide Response
- Curl:**

```

curl -X GET --header "Accept: application/json" "http://bookstorenewapp.tibcoqa.com:80/books"

```
- Request URL:** http://bookstorenewapp.tibcoqa.com:80/books
- Response Body:**

```

{
  "Book": [
    {
      "isbn": "1451648537",
      "name": "Cloud Foundry",
      "description": "Biography of Apple Co-Founder Steve Jobs",
      "authorName": "Walter Isaacson",
      "releaseDate": "2012-10-24Z",
      "vintage": false,
    }
  ]
}

```

3. Click the **Terminate** icon  to stop the process after you have completed using the operations on the page.

## Test Your Application in the Kubernetes Setup on the Google Cloud Platform

### Before you begin

- Google cloud account with a project and cluster.

- Google Cloud SDK
- Kubectl

## Procedure

1. From a terminal, follow these steps:

```
gcloud auth login
gcloud config set project <your project name>
gcloud config set container/cluster <your cluster name>
gcloud container clusters get-credentials <your cluster name> --zone <your
cluster zone name>
kubectl get nodes
```

2. Tag the application image created in the previous step.

```
docker tag bwce-rest-bookstore-app gcr.io/<your project name>/bwce-rest-
bookstore-app
```

3. Push your application image to Google Container Registry.

```
gcloud docker push gcr.io/<your project name>/bwce-rest-bookstore-app
```

4. Confirm that the image is present in the Google Container Registry.
5. Open `manifest.yml` file and update application image name. Ensure the image name follows the format:

```
gcr.io/<your gcloud project name>/<image name>
```

6. Run the following command to create the service:

```
kubectl create -f manifest.yml
```

7. To check that application has started successfully, run the command

```
kubectl logs pod-name
```

8. After few minutes, the external IP is available. Check it using the following command:

```
kubectl get service rest-bookstore-app
```

9. Retrieve the swagger doc using `http://<external ip>/swagger`.
10. Click any of the operations, such as POST, GET, PUT, DELETE as displayed on the web page and click **Try it out!**.
11. Expand the Books and Events headers and test the operations

## REST BookStore Consumption

This sample walks you through the process of creating an application in TIBCO BusinessWorks Container Edition to consume a RESTful webservice running in your Docker environment.

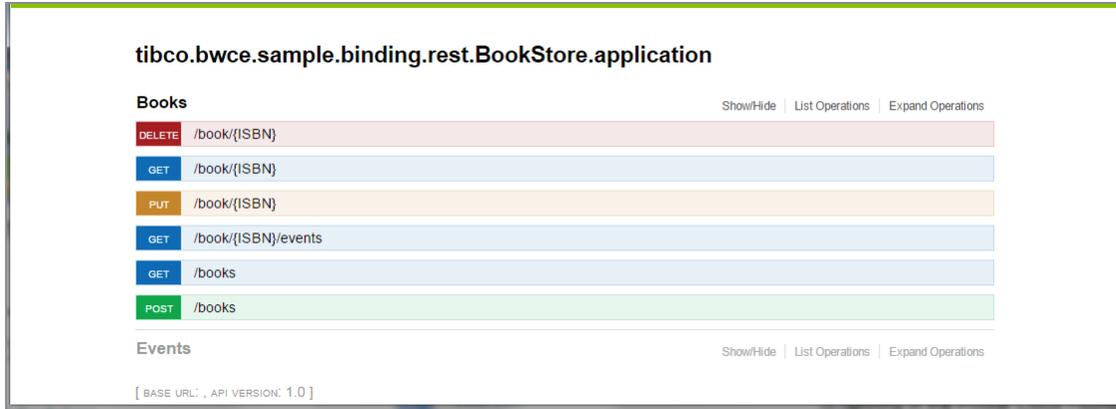
## Consuming a BookStore REST Endpoint in TIBCO Business Studio for BusinessWorks

### Importing the REST API document

Swagger 2.0-compliant REST API documents must be imported into the TIBCO Business Studio for BusinessWorks Service Descriptors folder of the project. This gives you the ability to expand and collapse endpoints, operations, parameters and response codes in the Project Explorer view.

### Accessing the BookStore Swagger in the Docker Environment

For the BookStore sample running in the Docker environment, the swagger URL should be accessible using the Docker external URL. in the format `http://docker-external-ip/swagger`.



## Retrieving the Swagger Document

You can now retrieve the swagger document by accessed using the URL

```
http://<docker external ip>/swagger/swagger.json
```

or using the curl command:

```
curl http://<docker external ip>/swagger/swagger.json -o bookstore.json
```

## Accessing the BookStore Swagger in the K8S Environment

If your BookStore sample is hosted in K8S environment, the swagger URL can be accessed using the external IP of the K8S service.

For example the following command will give you the external IP of the running Docker service

```
kubectl get svc rest-bookstore-app
```

## Importing the Swagger Document in Service Descriptors Folder

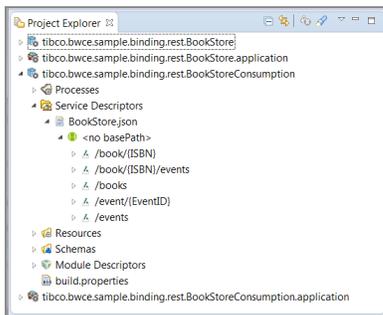
For configuring the design time, import the **bookstore.json** file in your workspace project under the **Service Descriptors** folder.

# Creating a REST Reference Binding

## Procedure

1. From the **File Explorer**, navigate to the samples directory and select **samples > container > cloudfoundry > binding > rest > BookStoreConsumption** and double-click **tibco.bwce.sample.binding.rest.BookStoreConsumption**.
2. From the Project Explorer, navigate to **tibco.bwce.sample.binding.rest.BookStoreConsumption > Service Descriptors > bookstore.json** to view the endpoints,

This will create a cloud shaped icon with a right facing arrow. The cloud is an indication that it is a REST Reference whereas the arrow within the cloud indicates that it is a binding. Since the binding is within a cloud, it is an indication that it is a REST binding.

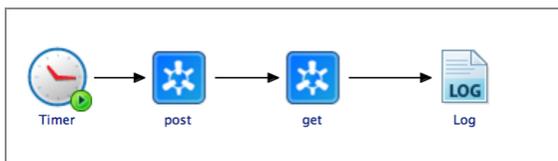


3. Drag and drop an endpoint ( **/books**) on the right side of the canvas to create a REST Reference Binding.

This creates an Invoke activity which is pre-configured to invoke the operation. It also creates an HTTP Client Shared Resource with the routable URL as the default host. The configuration for these entities is copied from the Swagger document from which you created the Reference Binding. The Reference consists of the name of the API as well as the operations it supports.

4. Test the configured process using the TIBCO Business Studio Debugger

The process will first try to run a POST a book and then get all the books from the BookStore Service.



The output of the GET operation can be seen as

```
09:05:42.933 INFO [bwEngThread:In-Memory Process Worker-3]
c.t.b.p.g.L.t.b.s.b.s.r.B.module.Log - First Book
Returned:Biography of Apple Co-Founder Steve Jobs
```

## Implementing a JMS Service that Returns Information Based on Zip Codes

This sample uses the JMS service to return information about a city given a zip code. The sample also provides the distance between two cities as defined by their zip codes.

### Before you begin

- TIBCO Enterprise Message Service must be running.

### Procedure

1. In the samples directory, select **binding > soap > JMS > ZipCodeLookup** and double-click **tibco.bw.sample.binding.soap.jms.ZipCodeLookup**.  
For more information, see [Accessing Samples](#).
2. In **Project Explorer** expand the **tibco.bw.sample.binding.soap.jms.ZipCodeLookup project**.
3. Fully expand the **Processes** directory and double-click **ZipInfoService.bwp**.
4. Verify your TIBCO Enterprise Message Service connection.
  - a. Fully expand the **Resources** directory.
  - b. Double-click **JMSConnectionResource.jmsConnResource**.
  - c. In the **Basic Configuration** dialog, click the **Test Connection** button to verify the connection.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of **Debug Configuration** wizard, expand **BusinessWorks Application** and select **BWApplication**.

7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.soap.jms.ZipCodeLookup.application**.
8. Click **Debug**.  
This runs the sample in Debug mode.
9. In TIBCO Business Studio for BusinessWorks, click the Terminate icon  to stop the process.

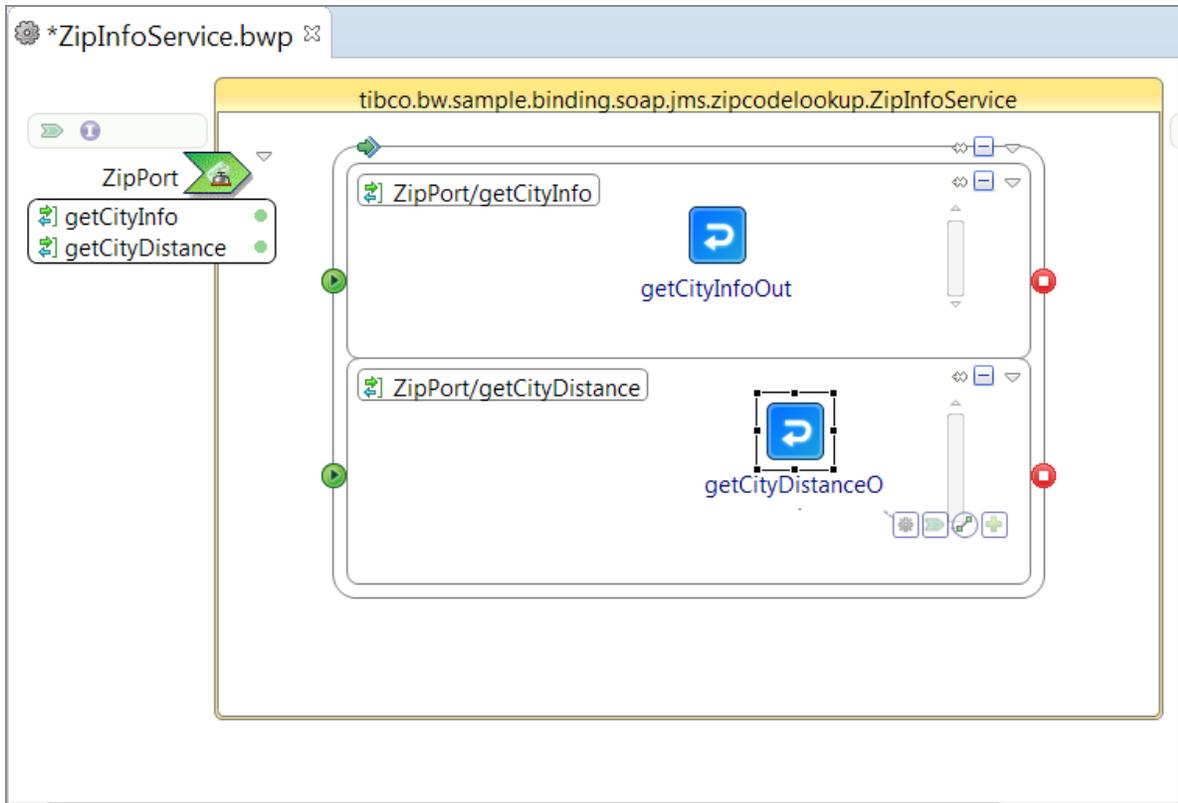
## Result

For the `getCityInfo` service, information corresponding to the zip code defined in the zip field is returned. The default value is 61801, which returns information about Urbana, IL.

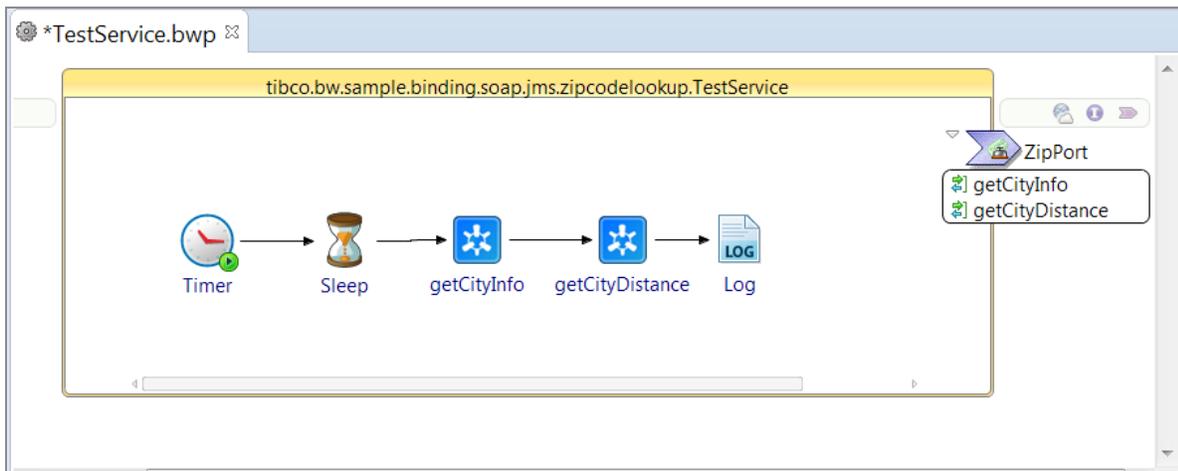
For the `getCityDistance` service, information corresponding to the distance between two zip codes is returned. The default values are 61801 for Urbana IL and 61820 for Champaign IL.

## Understanding the Configuration

The project contains the `ZipInfoService` process that provides city information about a zip code. JMS is used for the transport. The project represents a simple JMS based service using SOAP. The `getCityDistance` and `getCityInfo` operations are implemented in the `ZipInfoServiceprocess`.



The sample also includes a test client process called `TestService.bwp`, that invokes the `ZipInfoService` process.



# Testing the Sample

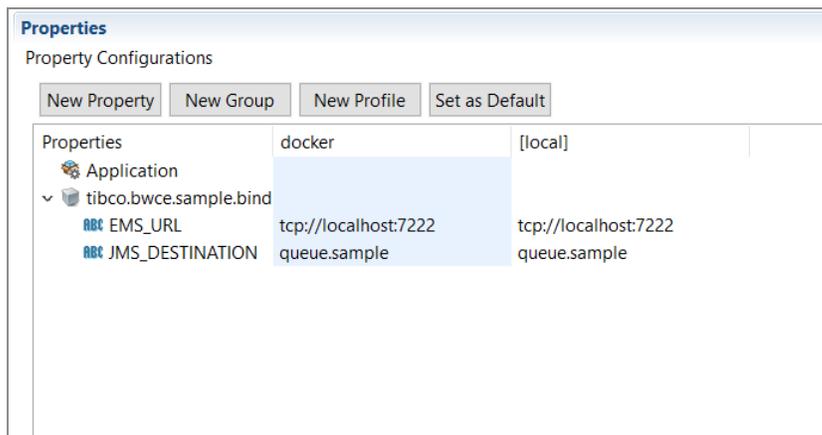
## Before you begin

- TIBCO BusinessWorks Container Edition runtime base image.  
Follow instructions in [Creating the TIBCO BusinessWorksTrademark\(TM\) Container Edition Application Docker Image](#) to create the runtime base image.
- Connection to EMS
- For Kubernetes
  - Google Cloud SDK installed on your machine.
  - Kubernetes tool installed on your machine.
  - Kubernetes project with cluster should be setup.

## Setting the Default Application Profile

### Procedure

1. From the Project Explorer expand **tibco.bwce.sample.binding.soap.jms.ZipcodeLookup.application**.
2. Expand the **Package Units** folder and click **Properties**.
3. Select the docker profile and set as default. If you do not have a profile named docker, click **New Profile** to create a profile and name it **docker**.



# Generating an Application Archive File

## Procedure

1. Expand the Package Unit and select **Overview** .
2. In the **Overview** window select **Export Application for Deployment**.
3. Enter the location of your EAR file.

# Test your Application Locally in a Docker Setup

## Procedure

1. Run the following command in a docker terminal:

```
docker run -ti -e "EMS_URL=tcp://54.67.133.122:7222" bwce-soap-jms-app
```

2. Run the following command to views the logs to make sure that the application has successfully started:

```
docker logs <container name>
```

3. Check the logs to make sure application has run successfully.

You should see some message like below:

```
13:49:06.733 INFO [Framework Event Dispatcher: Equinox Container:
608aaa95-52d7-0015-1f90-d3fd61937bf8]
com.tibco.thor.frwk.Application - Started by BusinessStudio,
ignoring .enabled settings.
13:49:11.962 INFO [bwEngThread:In-Memory Process Worker-6]
c.t.b.p.g.L.t.b.s.b.s.j.Z.Log - Output of getCityInfo : 61801
Urbana Illinois Urbana, Illinois, United States 40.11 88.207
Output of getCityDistance : Distance between two cities is : 4
miles. 61801 61820 Urbana Champaign Illinois Illinois Urbana,
Illinois, United States Champaign, Illinois, United States
```

# Test your Application in the Kubernetes Setup on the Google Cloud Platform

## Before you begin

- Google Cloud account with a project and cluster
- Google Cloud SDK
- Kubectl

## Procedure

1. From a terminal, run these commands:

```
gcloud auth login
gcloud config set project <your project name>
gcloud config set container/cluster <your cluster name>
gcloud container clusters get-credentials <your cluster name> --zone <your cluster zone name>
kubectl get nodes
```

2. Tag the application image created in the previous step.

```
docker tag bwce-soap-jms-app gcr.io/<your project name>/bwce-soap-jms-app
```

3. Push your application image to Google Container Registry.

```
gcloud docker push gcr.io/<your project name>/bwce-soap-jms-app
```

4. Confirm that the image is present in the Google Container Registry
5. Open the `manifest.yml` file and update the application image name. Ensure the image name follows the format:

```
gcr.io/<your gcloud project name>/<image name>
```

6. Navigate to the samples directory where the manifest file is present and run the following command to create the service:

```
kubectl create -f manifest.yml
```

7. To check that application is started successfully, run the following command

```
kubectl logs <pod name>
```

## Result

If the application deploys successfully, you can see a similar output in the console log.

```
13:49:06.733 INFO [Framework Event Dispatcher: Equinox Container:
608aaa95-52d7-0015-1f90-d3fd61937bf8] com.tibco.thor.frwk.Application -
Started by BusinessStudio, ignoring .enabled settings.
```

```
13:49:11.962 INFO [bwEngThread:In-Memory Process Worker-6]
c.t.b.p.g.L.t.b.s.b.s.j.Z.Log - Output of getCityInfo : 61801 Urbana
Illinois Urbana, Illinois, United States 40.11 88.207
```

```
Output of getCityDistance : Distance between two cities is : 4 miles.
61801 61820 Urbana Champaign Illinois Illinois Urbana, Illinois, United
States Champaign, Illinois, United States
```

## Resilience4j

This sample describes the Resilience4j functionality using HTTP Request Response in TIBCO BusinessWorks Container Edition.

# Testing in TIBCO Business Studio for BusinessWorks

## Before you begin

- Your computer must be connected to the Internet.
- Setup Prometheus to monitor events and Grafana dashboard provided by resilience4j for visualization.

## Procedure

1. From the **File Explorer**, navigate to the **samples > Container > docker > core > Resilience4j** folder and then double-click **tibco.bwce.sample.core.Resilience4j**.
2. From the **Project Explorer** expand the **tibco.bwce.sample.core.Resilience4j** project.
3. Expand the **Processes** folder and double-click **HTTP\_Request\_Response\_Example.bwp**.
4. Click **Run > Debug Configurations**.
5. From the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bwce.sample.palette.http.RequestResponse.application**.
7. Click **Debug**.  
The sample now runs in the debug mode.
8. Open the `request_news.html` file found in **samples > docker > core > Resilience4j**.
9. Click the **Get News from Wiki!** button to request headlines from the associated web page.

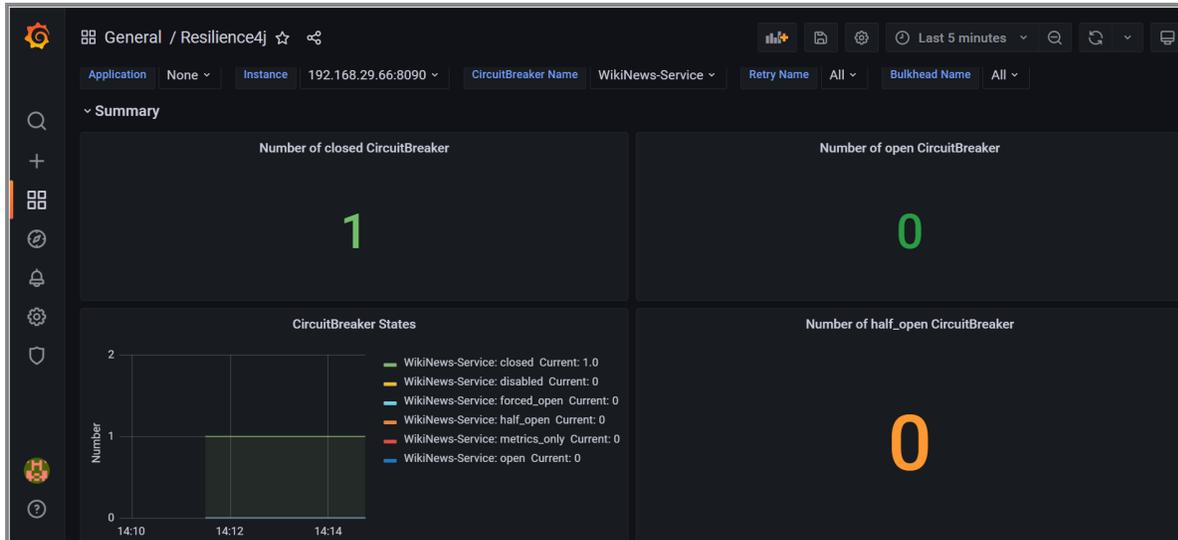
## Result

The console window shows engine messages similar to: The Wiki web page displays in your default browser. The message `Response sent successfully!!` is printed on the TIBCO Business Studio for BusinessWorks console.

In order to view the resilience4j metrics, you need a resilience4j stream URL which is as follows:

```
http://localhost:8090/resilience4j_metrics
```

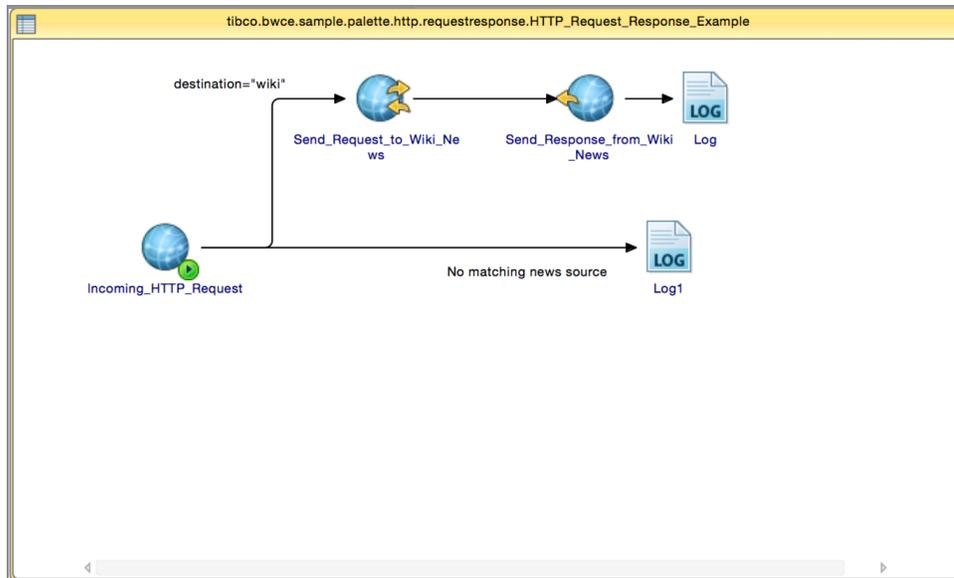
You can see the Grafana dashboard locally and the specified command name is displayed as shown below.



## Understanding the Configuration

This project contains the following:

- The Incoming HTTP Request process starter listens on the connection specified in `ListeningHTTPConnection.httpClientResource`. The `request_news.html` file contains a form and clicking the **Get News from Wiki!** button sends the corresponding text string to the Incoming HTTP Request activity.
- The conditional transition routes the request to the Send HTTP Request activity, which sends the request to the host using the `ListeningHTTPConnection`. After receiving a response from the remote site, the Send HTTP Response activity closes the HTTP Connection established by the Incoming HTTP Request process starter.
- An internet connection is required for the sample to connect to Wiki News. The `ListeningHTTPConnection` listens on `BW.CLOUD.PORT` (default value 8080).



- The HTTP client shared resource contains the Circuit Breaker configuration.

Circuit Breaker Configuration	
<b>Enable Circuit Breaker:</b>	<input checked="" type="checkbox"/>
<b>Circuit Breaker Name:</b>	IBM COMMAND
<b>Circuit Breaker Property</b>	
Sliding Window Type:	COUNT_BASED
Failure Rate Threshold(%):	50
Minimum Number of Calls:	100
Permitted Number Of Calls In HalfOpen State:	10
Wait Duration In Open State (ms):	60000
Slow Call Rate Threshold (%):	100
Slow Call Duration Threshold(ms):	60000
Automatic Transition From Open To HalfOpen Enabled:	<input type="checkbox"/>
Sliding Window Size:	100
Max Wait Duration In HalfOpen State (ms):	6000

## Testing the Resilience4j Sample

### Before you begin

- TIBCO BusinessWorks Container Edition runtime base image.  
Follow instructions in [Creating the TIBCO BusinessWorks Trademark\(TM\) Container Edition Application Docker Image](#) to create the runtime base image.
- A web browser.
- Setup Prometheus to monitor events and Grafana dashboard provided by Resilience4j for visualization.
- For Kubernetes

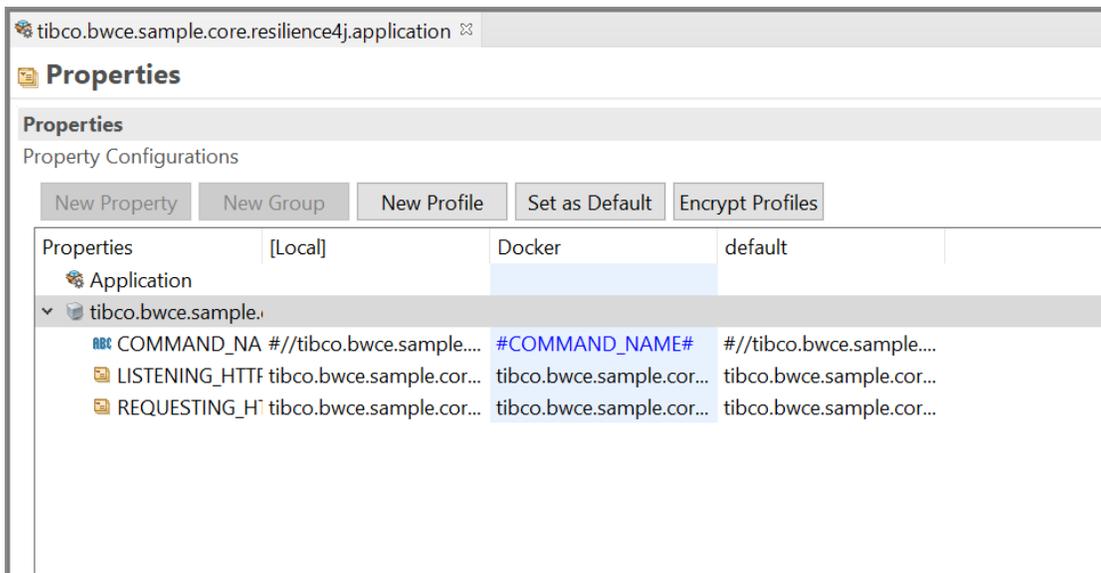
- Google Cloud SDK installed on your machine.
- Kubernetes tool installed on your machine.
- Kubernetes project with cluster should be setup.

## Setting the Default Application Profile

### Procedure

1. From the **Project Explorer** expand the **tibco.bwce.sample.core.Resilience4j.application**.
2. Expand the **Package Units** folder and click **Properties**.
3. Select a profile and **Set As Default**.

If you do not have a profile named **Docker**, click **New Profile** to create a profile and name it **Docker**.



## Generating an Application Archive File

Follow these steps to generate the .EAR file:

### Procedure

1. Expand the Package Unit and select **Overview**.
2. In the **Overview** window select **Export Application for Deployment**.
3. Enter the location of your EAR file.

## Building the Application Image

### Procedure

1. Copy the Dockerfile from the samples directory to the location where you placed the EAR file.
2. From the docker terminal navigate to the folder where the EAR and Dockerfile are stored.
3. In the Dockerfile, make sure the base image points to the TIBCO BusinessWorks Container Edition runtime base image.

Also make sure the ear file path and name is correct.

```
FROM tibco/bwce:latest
MAINTAINER TIBCO Software Inc.
ADD tibco.bwce.sample.core.Resilience4j.application_1.0.0.ear /
EXPOSE 8080
```

4. Run the following command on the docker terminal to generate the application image.

```
docker build -t bwce-Resilience4j.app .
```

## Testing the Application Locally in a Docker Setup

### Before you begin

Docker installed on your machine.

### Procedure

1. Run the following command on a docker terminal to run the application image:

```
docker run -i -d -p 8080:8080 -p 8090:8090 -e COMMAND_
NAME=WikiNews-Service bwce-resilience4j.app
```

2. Run the following command to view the logs to make sure that the application has started successfully.

```
docker logs <container name>
```

3. Get the IP address for the Docker machine.
4. Edit the **request\_news.html** file. Replace the URL "http://127.0.0.1:8080" with the IP address of the Docker machine and the port number.
5. **Save** the file.
6. Open the **request\_news.html** file in a web browser.
7. Click the **Get News from Wiki!** button to request headlines from the associated web page.

## Result

In order to view resilience4j metrics, a resilience4j stream URL is needed as follows:

```
http://<DOCKER-HOST-IP>:8090/resilience4j_metrics
```

# Test Your Application in the Kubernetes Setup on the Google Cloud Platform

## Before you begin

- Google cloud account with a project and cluster.
- Google Cloud SDK
- Kubectl
- Setup Prometheus to monitor events and Grafana dashboard provided by resilience4j for visualization.

## Procedure

1. From a terminal, follow these steps:

```
gcloud auth login
gcloud config set project <your project name>
gcloud config set container/cluster <your cluster name>
gcloud container clusters get-credentials <your cluster name> --zone <your
cluster zone name>
kubectl get nodes
```

2. Tag the application image created in the previous step.

```
docker tag bwce-Resilience4j.app gcr.io/<your project name>/bwce-
Resilience4j.app
```

3. Push your application image to Google Container Registry.

```
gcloud docker push gcr.io/<your project name>/bwce-Resilience4j.app
```

4. Confirm that the image is present in the Google Container Registry.
5. Open `manifest.yml` file and update application image name. Ensure the image name follows the format:

```
gcr.io/<your gcloud project name>/<image name>
```

6. Run the following command to create the service:

```
kubectl create -f manifest.yml
```

7. To check that application has started successfully, run the command

```
kubectl logs pod-name
```

8. The external IP address is available in a few minutes.  
Check for this using the following command:

```
kubectl get services
```

9. Edit the **request\_news.html** page found in **samples > docker > core > Resilience4j** directory.
10. Replace the URL `http://127.0.0.1:8080` with the routable URL of the application.
11. **Save** the file.
12. Open the `request_news.html` file in a web browser
13. Click the **Get News from Wiki!** button to request headlines from the associated web page.

## Result

In order to view the resilience4j metrics, you need a resilience4j stream URL which is as follows:

```
http://<SERVICE-EXTERNAL-IP>:8090/resilience4j_metrics
```

Once the news is displayed in the browser you can view the resilience4j metrics using Grafana dashboard with the command name specified as shown in [Testing in TIBCO Business Studio for BusinessWorks](#).

# Service Discovery

This sample walks you through the process of creating an application in TIBCO BusinessWorks Container Edition for registering a Service and discovering the registered service and deploying it in your Docker environment.

# Testing in TIBCO Business Studio for BusinessWorks

## Before you begin

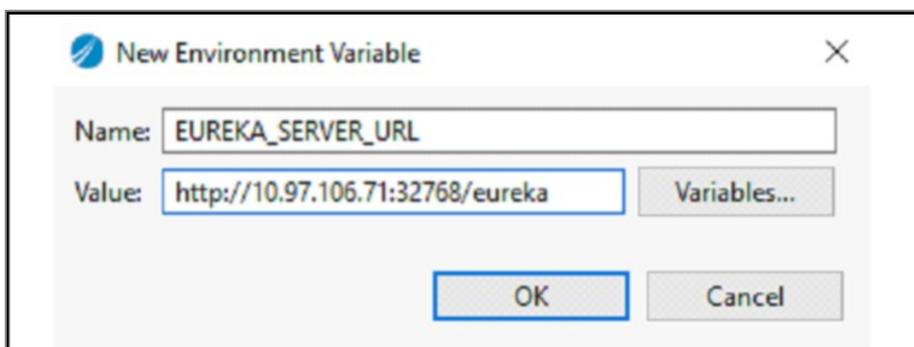
- Access to a local Consul or Eureka server URL used for Service Registration and

Discovery.

- A web browser.

## Procedure

1. From the **File Explorer**, navigate to the samples directory and select **Container > docker > core > ServiceDiscovery** and double-click `tibco.bwce.sample.servicediscovery.Service`.
2. From the **Project Explorer** expand the `tibco.bwce.sample.servicediscovery.Service.application` project.
3. Set the default **Application Profile** to **Local**. For more information, see [Setting the Default Application Profile](#).
4. From the **Project Explorer** expand the `tibco.bwce.sample.servicediscovery.Client` project.
5. Set the default **Application Profile** to **Local**. For more information, see [Setting the Default Application Profile](#).
6. Click **File > Save** to save the project.
7. From the **Project Explorer** expand the `tibco.bwce.sample.servicediscovery.Service` project.
8. Expand the **Processes** folder and the **com.tibco.sample.servicediscovery.client** package within that folder.
9. Click **Run > Debug Configuration**.
10. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
11. Select the Environment tab to add an environment variable.



12. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications.
13. Select the checkbox next to **tibco.bwce.sample.core.servicediscovery.client** and **tibco.bwce.sample.core.servicediscovery.Service**.
14. Click **Debug**.  
The sample now runs in the debug mode.
15. Trigger the application by accessing:

```
http://<hostname>:8080
```

16. Click the **Terminate**  icon to stop the process.

## Result

The console window shows messages similar to:

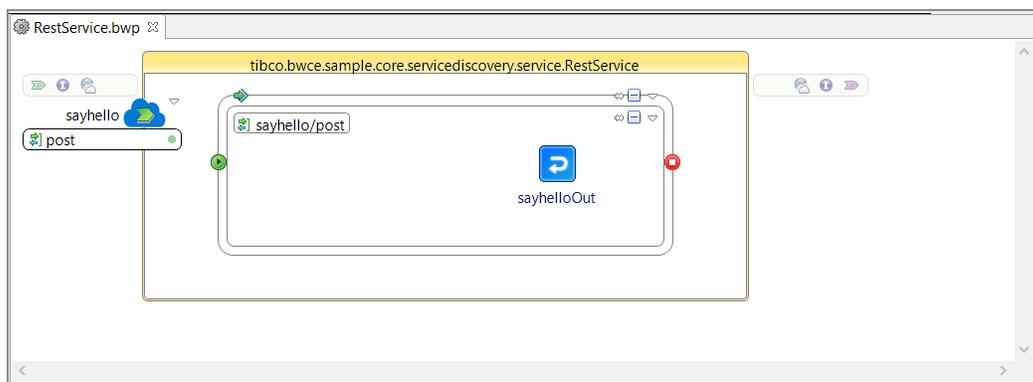
```
17:35:39.320 INFO [bwEngThread:In-Memory Process Worker-2]
c.t.b.p.g.L.t.b.s.S.Hystrix.Log - "Hello from TIBCO"
```

## Understanding the Configuration

The Service project contains the following:

- The RestService Process (**RestService.bwp**)

This process contains a process service with one operation sayhello.



- To discover your service, enable service discovery and Circuit Breaker in the HTTP

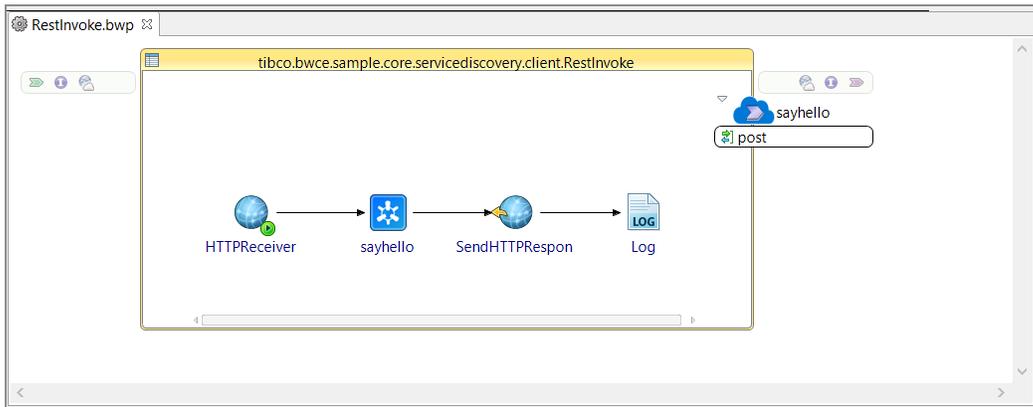
connector shared resource of your Service project.



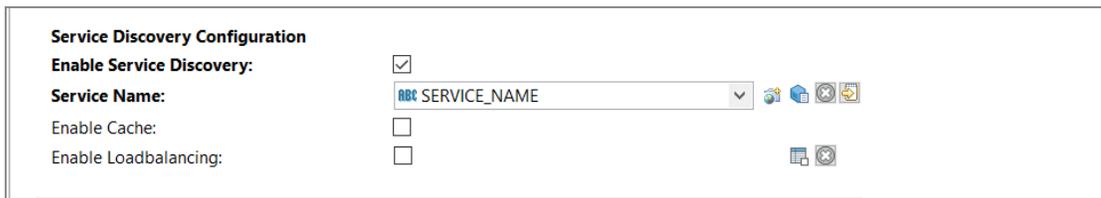
The Client project contains the following:

- The RestInvoke Process (**RestInvoke.bwp**)

This process contains an Invoke operation which invokes the sayhello operation. The HTTP Receiver is the process starter which triggers the process when it receives the request on the listening port.



- In the client project, you have to enable the service discovery configuration in HTTP client shared resource.



## Testing the Service Discovery Sample

### Before you begin

- A web browser.
- TIBCO BusinessWorks Container Edition runtime base image.

For more information, see [Creating the TIBCO BusinessWorksTrademark\(TM\)](#)

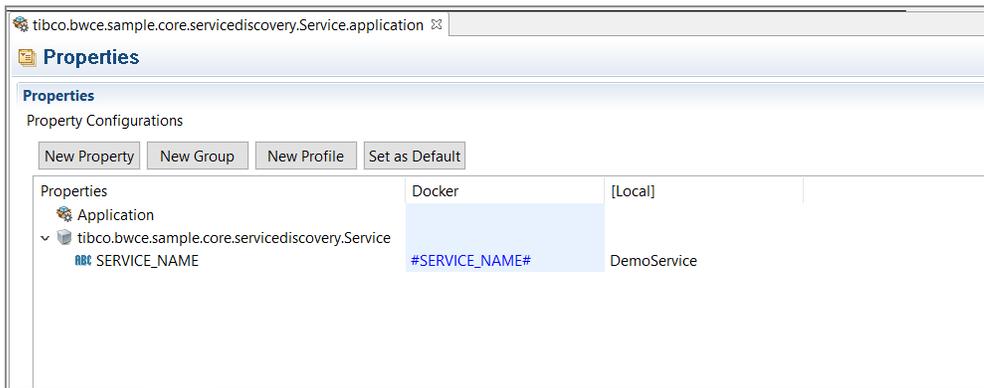
[Container Edition Application Docker Image](#) to create the runtime base image.

## Setting the Default Application Profile

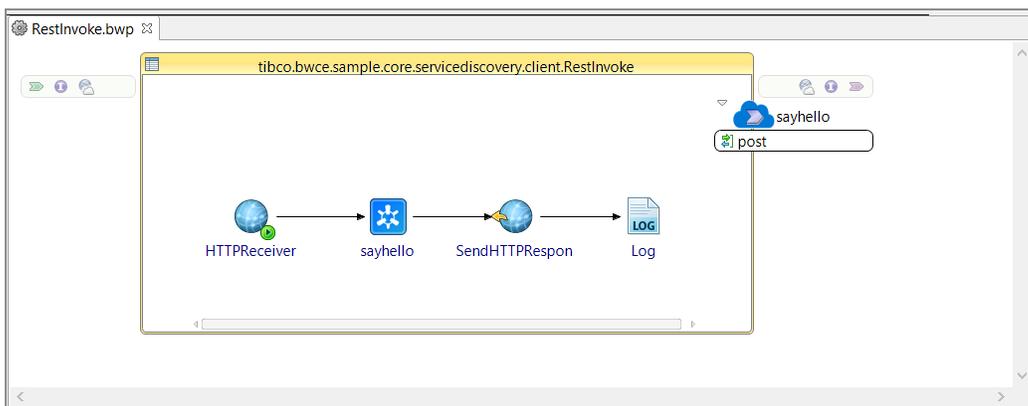
### Procedure

1. From the **Project Explorer** expand the **tibco.bwce.sample.core.servicediscovery.Service.application**.
2. Expand the **Package Units** folder and click **Properties**.
3. Select a profile and **Set As Default**.

If you do not have a profile named **Docker**, click **New Profile** to create a profile and name it **Docker**.



4. Similarly set the default application profile for **tibco.bwce.sample.core.servicediscovery.Client.application**.



## Generating an Application Archive File

Follow these steps to generate the .EAR file:

### Procedure

1. Expand the Package Unit and select **Overview**.
2. In the **Overview** window select **Export Application for Deployment**.
3. Enter the location of your EAR file.

## Building the Application Image for the Service Application

### Procedure

1. Copy the Dockerfile from the samples directory to the location where you placed the EAR file.
2. From the docker terminal, navigate to the folder where the EAR and Dockerfile are stored.
3. In the Dockerfile, make sure the base image points to the TIBCO BusinessWorks Container Edition runtime base image.

Also make sure the ear file path and name is correct.

```
FROM tibco/bwce:latest
MAINTAINER TIBCO Software Inc.
ADD tibco.bwce.sample.core.servicediscovery.Service.application_
1.0.0.ear /
EXPOSE 8080
```

4. Run the following command on the docker terminal to generate the application image.

```
docker build -t bwce-servicediscovery.service.app .
```

# Building the Application Image for the Client Application

## Procedure

1. Copy the Dockerfile from the samples directory to the location where you placed the EAR file.
2. From the docker terminal, navigate to the folder where the EAR and Dockerfile are stored.
3. In the Dockerfile, make sure the base image points to the TIBCO BusinessWorks Container Edition runtime base image.

Also make sure the ear file path and name is correct.

```
FROM tibco/bwce:latest
MAINTAINER TIBCO Software Inc.
ADD tibco.bwce.sample.core.servicediscovery.Client.application_
1.0.0.ear /
EXPOSE 8080
```

4. Run the following command on the docker terminal to generate the application image.

```
docker build -t bwce-servicediscovery.client.app .
```

# Testing the Service and Client Applications Locally in a Docker Setup

## Before you begin

- Docker must be installed on your machine.
- Access to a local Consul server.

## Procedure

1. Run the following command on a Docker terminal to run the service application

image:

```
docker run -e CONSUL_SERVER_URL=consul-server-url
-e DOCKER_LOCAL_HOST_IP=Docker-machine-ip
-e DOCKER_LOCAL_HOST_PORT=Docker-machine-port -p 18086:8080
-e SERVICE_NAME=BWCE-HELLOWORLD-SERVICE bwce-
servicediscovery.service-app
```

**i Note: DOCKER\_LOCAL\_HOST\_IP:** While registering service, the variable value becomes the value for IP address configured with consul service.

**i Note: DOCKER\_LOCAL\_HOST\_PORT:** While registering service, the variable value becomes the value for PORT configured with consul service.

2. Run the following command on a Docker terminal to run the service application image:

```
docker run -e CONSUL_SERVER_URL=consul-server-url
-p 18087:8080
-e SERVICE_NAME=BWCE-HELLOWORLD-SERVICE bwce-
servicediscovery.client.app
```

3. Run the following command to view the logs to make sure that the application has started successfully.

```
docker logs <container name>
```

4. Check the logs for the service and client applications to make sure they have started successfully.
5. Get the IP address for the Docker machine.
6. Access the following URL from a browser:

```
http://<docker-machine-ip>:18087
```

## Result

The following output should be seen in the browser:

```
Hello from TIBCO"
```

# Test Your Application in the Kubernetes Setup on the Google Cloud Platform

## Before you begin

- Google cloud account with a project and cluster.
- Google Cloud SDK
- Kubectl

## Procedure

1. From a terminal, follow these steps:

```
gcloud auth login
gcloud config set project <your project name>
gcloud config set container/cluster <your cluster name>
gcloud container clusters get-credentials <your cluster name> --zone <your
cluster zone name>
kubectl get nodes
```

2. Tag the service application image created in the previous step.

```
docker tag bwce-service-discovery-service-app gcr.io/<your project
name>/bwce-service-discovery-service-app
```

3. Push your application image to Google Container Registry.

```
gcloud docker push gcr.io/<your project name>/bwce-service-discovery-
service-app
```

4. Confirm that the image is present in the Google Container Registry.
5. Open `manifest-service.yml` file and update application image name, the Consul server URL, and the service name. Ensure the image name follows the format:

```
gcr.io/<your gcloud project name>/<image name>
```

6. Run the following command to create the service:

```
kubectl create -f manifest-service.yml
```

7. Tag the service application image created in the previous step.

```
docker tag bwce-service-discovery-client-app gcr.io/<your project name>/bwce-service-discovery-client-app
```

8. Push your application image to Google Container Registry.

```
gcloud docker push gcr.io/<your project name>/bwce-service-discovery-client-app
```

9. Confirm that the image is present in the Google Container Registry.
10. Open `manifest-client.yml` file and update application image name, Consul server URL, and the service name. Ensure the image name follows the format:

```
gcr.io/<your gcloud project name>/<image name>
```

11. Run the following command to create the service:

```
kubectl create -f manifest-client.yml
```

12. To check that the service and client applications have started successfully, run the command:

```
kubectl logs pod-name
```

13. The external IP address is available in a few minutes.  
Check for this using the following command:

```
kubectl get services
```

14. Using a browser, send a request to the client application by using its external IP address.

```
http://<EXTERNAL-IP>:80
```

## Result

The following output should be seen in the browser:

```
Hello from TIBCO"
```

# HTTP Request Response

This sample demonstrates how to use the HTTP palette activities to configure requests to a web server and manage the responses.

## Before you begin

- Your computer must be connected to the Internet.

# Testing in TIBCO Business Studio for BusinessWorks

## Procedure

1. In the samples directory, select **samples > Container > docker > palette > http > RequestResponse** and double-click **tibco.bwce.sample.palette.http.RequestResponse**. For more information, see [Accessing Samples](#).
2. From the **Project Explorer** expand the **tibco.bwce.sample.palette.http.RequestResponse** project.
3. Click **Run > Debug Configurations**.

4. From the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
5. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications.
6. Select the checkbox next to **tibco.bwce.sample.palette.http.RequestResponse.application**.
7. Click **Debug**.  
The sample now runs in the debug mode.
8. Open the `request_news.html` file found in **samples > docker > palette > http > RequestResponse**.
9. Click the **Get News from Wiki!** button to request headlines from the associated web page.

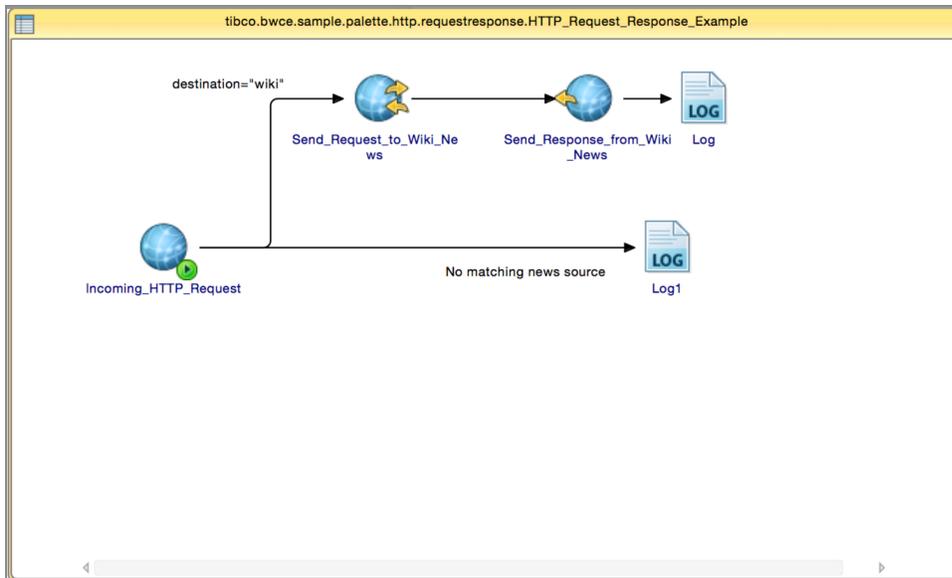
## Result

The Wiki web page displays in your default browser. The message `Response sent successfully!!` is printed on the TIBCO Business Studio for BusinessWorks console.

## Understanding the Configuration

The following shared resources are defined in the project:

- The Incoming HTTP Request process starter listens on the connection specified in `ListeningHTTPConnection.httpConnResource`. The `request_news.html` file contains a form and clicking the **Get News from Wiki!** button sends the corresponding text string to the Incoming HTTP Request activity.
- The conditional transition routes the request to the `Send HTTP Request` activity, which sends the request to the host using the `ListeningHTTPConnection`. After receiving a response from the remote site, the `Send HTTP Response` activity closes the HTTP Connection established by the Incoming HTTP Request process starter.
- An internet connection is required for the sample to connect to Wiki News. The `ListeningHTTPConnection` listens on `BW.CLOUD.PORT` (default value 8080).



## Testing the HTTP Request Response Sample

### Before you begin

- TIBCO BusinessWorks Container Edition runtime base image.  
For more information, see [Creating the TIBCO BusinessWorksTrademark\(TM\) Container Edition Application Docker Image](#) to create the runtime base image.
- A web browser.
- For Kubernetes
  - Google Cloud SDK installed on your machine.
  - Kubernetes tool installed on your machine.
  - Kubernetes project with cluster should be setup.

# Creating the TIBCO BusinessWorks Container Edition Base Docker Image for Linux Containers

## Before you begin

- Download the TIBCO BusinessWorks Container Edition runtime zip file, `bwce-runtime-<version>.zip`, from <http://edelivery.tibco.com>.

To download this file,

1. Select **Container** from the **Operating Systems** drop down list.
  2. Read and Accept the **TIBCO End User License Agreement**.
  3. Select the radio button for **Individual file Download**.
  4. Click the **+** sign to view the individual components and select `bwce-runtime-<version>.zip`.
- An installation of Docker.

**i Note:** TIBCO ActiveMatrix BusinessWorks™ does not ship docker folder or scripts with it, because the scripts can be pulled directly from GitHub without installing TIBCO BusinessWorks Container Edition separately.

## Procedure

1. Navigate to the `TIBCO_HOME/bwce/<version>/docker` directory.
2. Copy the `bwce-runtime-<version>.zip` file to the `TIBCO_HOME/bwce/<version>/docker/resources/bwce-runtime` folder.
3. Open a command terminal and run the following command from the `TIBCO_HOME/bwce/<version>/docker` folder:

```
docker build -t TAG-NAME .
```

For example,

```
docker build -t tibco/bwce:latest .
```

By default, the `debian:bookworm-slim` image is used to create base docker image for

TIBCO BusinessWorks Container Edition. The following is the dockerfile content.

- Dockerfile content for debian:bookworm-slim:

```
FROM debian:bookworm-slim
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 755 /scripts/*.sh && apt-get update && apt-get --no-
install-recommends -y install unzip ssh net-tools && apt-get -
y install xsltproc && apt-get clean && rm -rf
/var/lib/apt/lists/*
RUN groupadd -g 2001 bwce \
&& useradd -m -d /home/bwce -r -u 2001 -g bwce bwce
USER bwce
ENV LANG C.UTF-8
ENV LC_ALL C.UTF-8
ENTRYPOINT ["/scripts/start.sh"]
```

You can also change the image with the following dockerfile content.

- Dockerfile content for openSUSE:

```
FROM opensuse/leap
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 755 /scripts/*.sh && zypper -n update && zypper -n
refresh && \
zypper -n in unzip openssh net-tools
RUN groupadd -g 2001 bwce \
&& useradd -m -d /home/bwce -r -u 2001 -g bwce bwce
USER bwce
ENTRYPOINT ["/scripts/start.sh"]
```

- Dockerfile content for CentOS 7:

```
FROM centos:7
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 755 /scripts/*.sh && yum -y update && yum -y install
unzip ssh net-tools
ENTRYPOINT ["/scripts/start.sh"]
```

- Dockerfile content for CentOS 9:

```
FROM quay.io/centos/centos:stream9
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 755 /scripts/*.sh && yum -y update && yum -y install
unzip ssh net-tools
ENTRYPOINT ["/scripts/start.sh"]
```

- Dockerfile content for eclipse-temurin:11-jre-alpine

```
FROM eclipse-temurin:11-jre-alpine
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 755 /scripts/*.sh && apk update && apk add unzip
openssh net-tools
RUN apk add --no--cache bash
RUN addgroup -S bwcegroup && adduser -S bwce -G bwcegroup
USER bwce
ENTRYPOINT ["/scripts/start.sh"]
```

**i Note:** To use OpenJDK11 and remove TIBCO JRE from eclipse-temurin:11-jre-alpine based container image, navigate to <https://github.com/TIBCOSoftware/bwce-docker/tree/openjdk-alpine> and clone the 'eclipse-alpine' repository to your local machine. For more information to create the base Docker image for Linux container, see "[Creating the TIBCO BusinessWorks Container Edition Base Docker Image for Linux Containers](#)"

- Dockerfile content for ubi8:

```
FROM registry.access.redhat.com/ubi8/ubi:latest
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 777 scripts/*.sh && yum install -y unzip net-tools
&& \
yum update -y; yum clean all
ENTRYPOINT ["/scripts/start.sh"]
```

- Dockerfile content for rhel7-minimal

```
FROM registry.access.redhat.com/rhel7-minimal
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 777 scripts/*.sh && microdnf install unzip net-tools
--enablerepo=rhel-7-server-rpms && \
microdnf update; microdnf clean all
ENTRYPOINT ["/scripts/start.sh"]
```



**Note:** Ensure that you have valid Red Hat subscription for using **rhel7-minimal**.

- Dockerfile content for Redhat Standard OS:

```
FROM registry.access.redhat.com/rhel7/rhel
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 777 scripts/*.sh && yum install -y unzip ssh net-
tools && \
yum update -y; yum clean all
ENTRYPOINT ["/scripts/start.sh"]
```

- Dockerfile content for AmazonLinux 2 OS:

```
FROM amazonlinux:2
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 777 scripts/*.sh && yum install -y unzip ssh net-
tools && \
yum update -y; yum clean all
RUN groupadd -g 2001 bwce \
&& useradd -m -d /home/bwce -r -u 2001 -g bwce bwce
USER bwce
ENV LANG C.UTF-8
ENV LC_ALL C.UTF-8
ENTRYPOINT ["/scripts/start.sh"]
```

- Dockerfile content for AmazonLinux 2023 OS:

```

FROM amazonlinux:2023
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 777 scripts/*.sh && yum update -y && yum install -y
  unzip openssl-clients.x86_64 net-tools.x86_64
  findutils && yum clean all
ENV LANG C.UTF-8
ENV LC_ALL C.UTF-8
ENTRYPOINT ["/scripts/start.sh"]

```

## Result

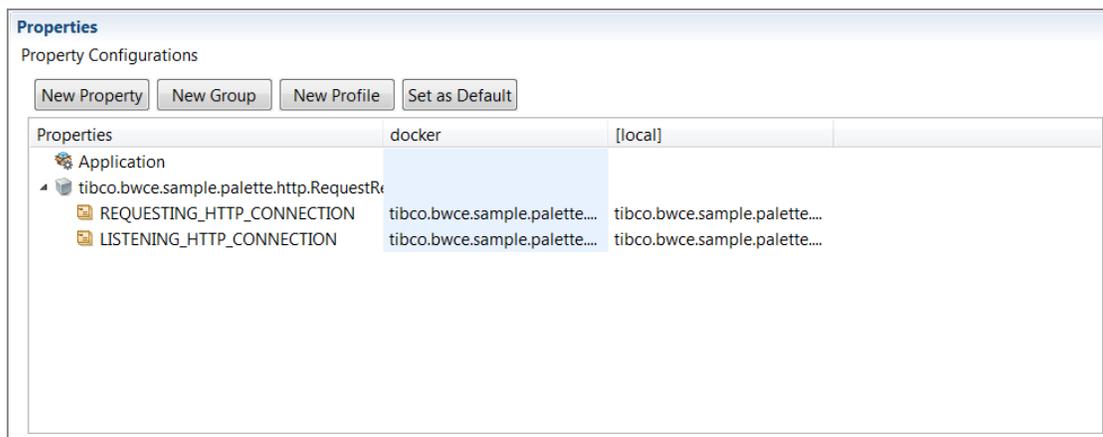
The TIBCO BusinessWorks Container Edition base docker image is now created. This base docker image can now be used to create docker application images.

## Setting the Default Application Profile

### Procedure

1. From the Project Explorer expand the **tibco.bwce.sample.palette.http.RequestResponse.application**.
2. Expand the **Package Units** folder and click **Properties**.
3. Select the **docker** profile and **Set As Default**.

If you do not have a profile named **docker**, click **New Profile** to create a profile and name it **docker**.



## Generating an Application Archive File

Follow these steps to generate the .EAR file:

### Procedure

1. Expand the Package Unit and select **Overview**.
2. In the **Overview** window select **Export Application for Deployment**.
3. Enter the location of your EAR file.

## Building an Application Image

### Procedure

1. Copy the Dockerfile from the samples directory to the location where you placed the EAR file.
2. From the Docker terminal, navigate to the folder where the EAR and Dockerfile are stored.
3. In the Dockerfile, make sure the base image points to the TIBCO BusinessWorks Container Edition runtime base image.
4. Make sure the EAR file name and the path is correct.
5. Run the following command to generate the application image:

```
docker build -t bwce-http-reqresp-app .
```

## Test your Application Locally in a Docker Setup

### Before you begin

- Docker should be installed on your machine.

### Procedure

1. Run the command in a docker terminal.

```
docker run -i -d -p 8080:8080 bwce-http-reqresp-app
```

2. Run the following command to view the logs to make sure that the application has successfully started.

```
docker logs <container name>
```

3. Get the IP address for the Docker machine.
4. Edit the **request\_news.html** file. Replace the URL "http://127.0.0.1:8080" with the IP address of the Docker machine.
5. Save the file.
6. Open the `request_news.html` file in a web browser.
7. Click the **Get News from Wiki!** button to request headlines from the associated web page.

## Test your Application in the Kubernetes Setup on the Google Cloud Platform

### Before you begin

- Google Cloud account with a project and cluster
- Google Cloud SDK
- Kubectl

### Procedure

1. From a terminal run the following commands:

```
gcloud auth login
```

```
gcloud config set project <your project name>
gcloud config set container/cluster <your cluster name>
gcloud container clusters get-credentials <your cluster name> --zone <your
cluster zone name name>
kubectl get nodes
```

2. Tag the application image created in the previous step.

```
docker tag bwce-http-reqresp-app gcr.io/<your project name>/bwce-http-
reqresp-app
```

3. Push your application image to Google Container Registry.

```
gcloud docker push gcr.io/<your project name>/bwce-http-reqresp-app
```

4. Confirm that the image is present in the Google Container Registry.
5. Open the `manifest.yml` file and update the application image name. Ensure the image name follows the format

```
gcr.io/<your gcloud project name>/<image name>
```

6. Navigate to the `samples` directory where the manifest file is present and run the following command to create the service:

```
kubectl create -f manifest.yml
```

7. To check that application has started successfully, type in below command

```
kubectl logs <pod name>
```

8. After few minutes, the external IP is available. Check it using the following command:

```
kubectl get service bwce-http-requestreply-service
```

9. Edit the `request_news.html` file, replace the URL `http://127.0.0.1:8080` with the external IP address, and save the file
10. Open the `request_news.html` file in a web browser
11. Click the **Get News from Wiki!** button to request headlines from the associated web page.

## JDBC Basic

This sample establishes a JDBC connection with a database and runs the SELECT and UPDATE queries on the tables. The intention of the sample is to show how to use DataDirect or native third party jars in the TIBCO BusinessWorks Container Edition runtime and deploy it in the Docker and Kubernetes environment.

## Testing in TIBCO Business Studio for BusinessWorks

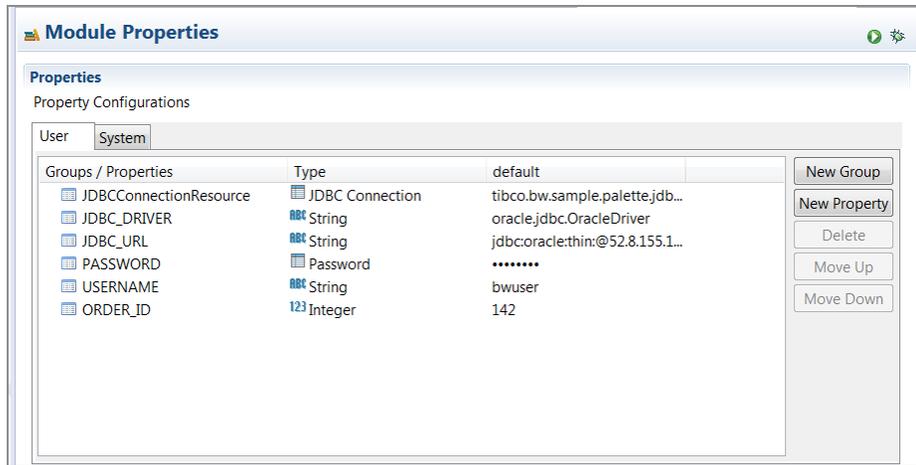
This section describes how to setup a JDBC Connection to Query and Update tables.

### Before you begin

- Access to a database.

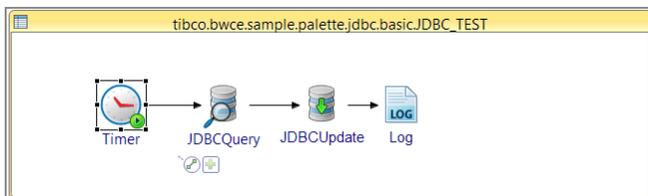
### Procedure

1. From the **File Explorer**, navigate to the samples directory and select **samples > Container > docker > palette > jdbc > Basic** and then double-click **tibco.bwce.sample.palette.jdbc.Basic**.
2. From the **Project Explorer** expand the `tibco.bwce.sample.palette.jdbc.Basic` project.
3. Expand the **Module Descriptors** folder and double-click **Module Properties**.



The JDBC properties defined for the application are defined in the dialog. Provide a valid username, password, and database URL to connect to your database.

4. Update the values under **Module properties** and save your project.
5. Expand the Processes directory and double-click **JDBC\_TEST.bwp**.



6. Verify your JDBC connection.
  - a. Expand the **Resource** directory.
  - b. Double-click **JDBCConnection\_Oracle.jdbcResource**
  - c. Select the TIBCO DataDirect Driver and provide valid access credential.
  - d. Click the **Test Connection** button to verify the connection.
7. Click **File > Save** to save the project.
8. Click **Run > Debug Configurations**.
9. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
10. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bwce.sample.palette.jdbc.Basic.application**.

11. Click **Debug**.

This runs the sample in Debug mode.

12. Click the **Terminate**  icon to stop the process.

## Understanding the Configuration

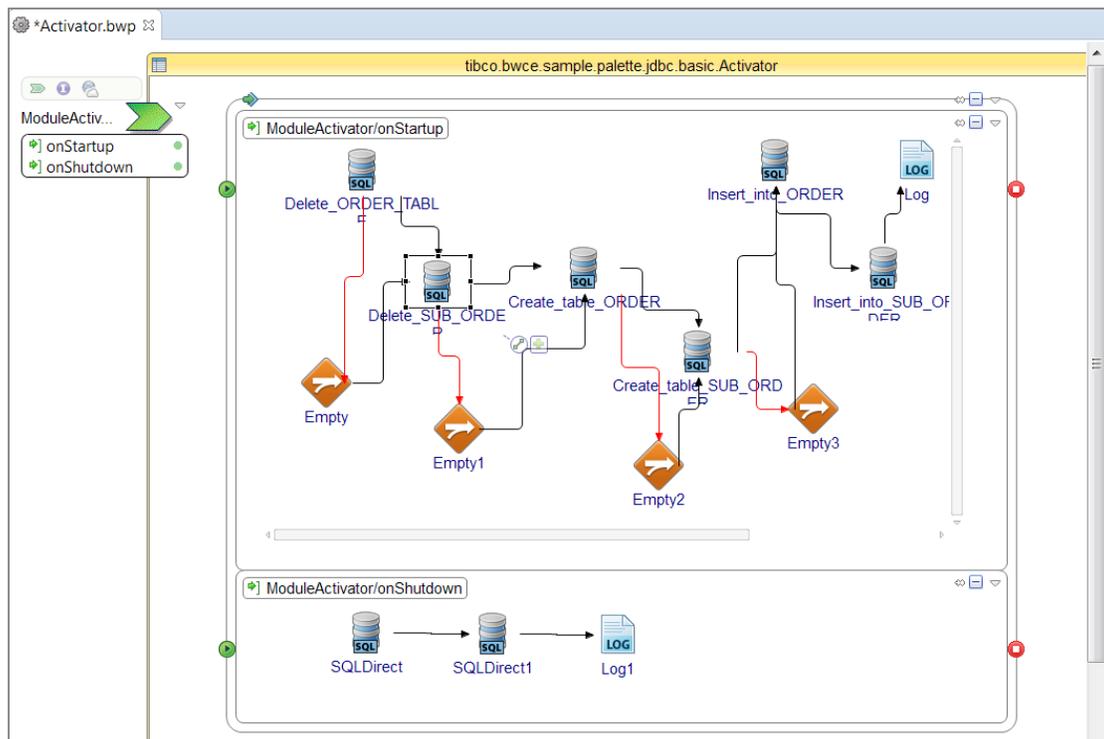
This project contains the following processes:

- The Activator Process (**Activator.bwp**)

The BusinessWorks Activator process is used to perform pre-processing and post-processing tasks when the application is started and stopped respectively.

This contains a process service with two operations: **OnStartup** and **OnShutDown**.

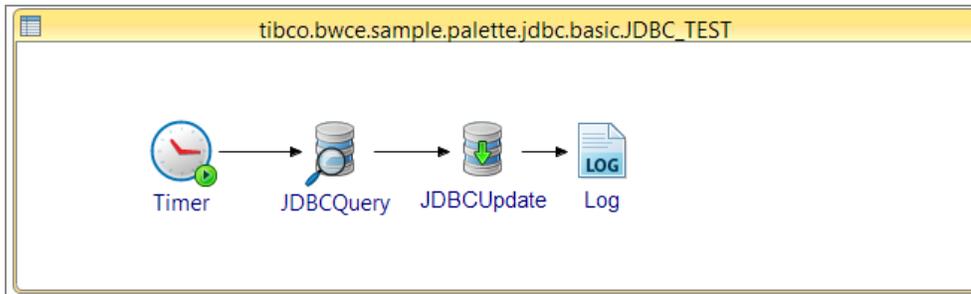
The onStartup operation creates two database tables: order\_table and sub\_order and the onShutdown operation drops both the tables at the end of the application execution.



- The JDBC test process (**JDBC\_TEST.bwp**)

This process uses the JDBC Connection in the Resources folder to establish a

connection to the database and to run SELECT and UPDATE queries. The process first queries the table for a specific record using the JDBC\_Query activity and then it updates another table using the JDBC\_Update activity. Both activities use prepared parameters as input. This shows the ability to run the same statement with multiple values by caching the statement at runtime.



## Testing the JDBC Basic Sample

### Before you begin

- TIBCO BusinessWorks Container Edition runtime base image.

Follow instructions in [Creating the TIBCO BusinessWorks Container Edition Base Docker Image for Linux Containers](#) to create the runtime base image.

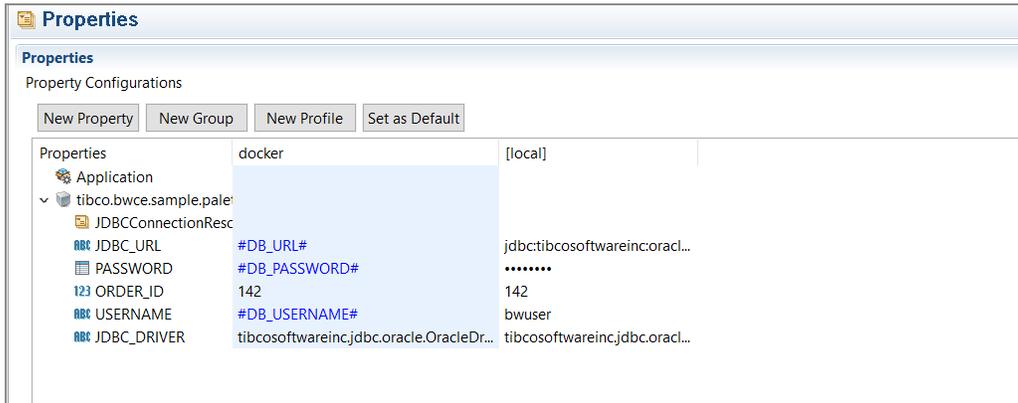
- A web browser.
- For Kubernetes
  - Google Cloud SDK installed on your machine.
  - Kubernetes tool installed on your machine.
  - Kubernetes project with cluster should be setup.

## Setting the Default Application Profile

### Procedure

1. From the Project Explorer expand the **tibco.bwce.binding.jdbc.Basic.application**.
2. Expand the **Package Units** folder and click **Properties**.
3. Select the **docker** profile and **Set As Default**.

If you do not have a profile named **docker**, click **New Profile** to create a profile and name it **docker**.



## Generating an Application Archive File

Follow these steps to generate the .EAR file:

### Procedure

1. Expand the Package Unit and select **Overview**.
2. In the **Overview** window select **Export Application for Deployment**.
3. Enter the location of your EAR file.

## Testing the Application Locally in a Docker Setup

### Procedure

1. Run the command in a docker terminal:

```
docker run -ti -e "DB_
URL=jdbc:tibcosoftwareinc:oracle://54.6.155.176:1521;ServiceName=or
cl" -e "DB_USERNAME=bwuser" -e "DB_PASSWORD=password" bwce-jdbc-
basic-app
```

2. Run the following command to views the logs to make sure that the application has successfully started:

```
docker logs <container name>
```

For example,

```
docker logs jdbc-basic-app
```

3. Check the logs to make sure application has run successfully.

You should see some message like below:

```
13:49:06.733 INFO [Framework Event Dispatcher: Equinox Container:
608aaa95-52d7-0015-1f90-d3fd61937bf8]
com.tibco.thor.frwk.Application - Started by BusinessStudio,
ignoring .enabled settings.
13:49:11.962 INFO [bwEngThread:In-Memory Process Worker-6]
c.t.b.p.g.L.t.b.s.b.s.j.Z.Log - Output of getCityInfo : 61801
Urbana Illinois Urbana, Illinois, United States 40.11 88.207
Output of getCityDistance : Distance between two cities is : 4
miles. 61801 61820 Urbana Champaign Illinois Illinois Urbana,
Illinois, United States Champaign, Illinois, United States
```

## Testing the Application in the Kubernetes Setup on the Google Cloud Platform

### Before you begin

- Google Cloud account with a project and cluster
- Google Cloud SDK
- Kubectl

### Procedure

1. From a terminal, follow these steps:

```
gcloud auth login
gcloud config set project <your project>
gcloud config set container/cluster <your cluster>
gcloud container clusters get-credentials <your cluster> --zone <your cluster
zone>
kubectl get nodes
```

2. Tag the application image created in the previous step.

```
docker tag bwce-jdbc-basic-app gcr.io/<your project>/bwce-jdbc-basic-app
```

3. Push your application image to Google Container Registry.

```
gcloud docker push gcr.io/<your project>/bwce-jdbc-basic-app
```

4. Confirm that the image is present in the Google Container Registry
5. Open the `manifest.yml` file and update the application image name. Ensure the image name follows the format:

```
gcr.io/<your gcloud project name>/<image name>
```

6. Navigate to the samples directory where the manifest file is present and type in the below command to create the service:

```
kubectl create -f manifest.yml
```

7. To check that application is started successfully, type in below command

```
kubectl logs pod-name
```

## Result

If the application deploys successfully, you can see a similar output in the console log.

```
05:16:00.763 INFO [bwEngThread:In-Memory Process Worker-5]
```

```
c.t.b.p.g.L.t.b.s.p.jdbc.Basic.Log - Records Updated By JDBCUpdate  
Activity: 1
```

## JMS Basic

JMS Basic sample uses the JMS Queue Receiver and Reply to JMS Message activities to receive and respond to JMS messages. The JMS Queue Receiver activity is triggered by a JMS Request Reply activity from another process. This sample demonstrates how TIBCO BusinessWorks Container Edition applications can send and receive messages from JMS destinations using TIBCO EMS.

### Before you begin

- TIBCO Enterprise Message Server must be running.
- Ensure that the queues `jmsbasic.queue` and `reply.queue` have been created.

## Testing in TIBCO Business Studio for BusinessWorks

### Procedure

1. From the **File Explorer** navigate to the samples directory, select **Container > docker > palette > jms > Basic** and double-click **tibco.bwce.sample.palette.jms.Basic**. For more information, see [Accessing Samples](#).
2. From the **Project Explorer** expand the **tibco.bwce.sample.palette.jms.Basic** project.
3. Verify your TIBCO Enterprise Message Service connection.
  - a. Completely expand the **Resources** directory.
  - b. Double-click **JMSConnectionResource.jmsConnResource**.
  - c. Click the **Test Connection** button to verify the connection.
4. From the **Project Explorer** expand the **Processes** folder and the **QueueReceiver.bwp** and **RequestReply.bwp** processes within that folder.

5. From the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications.
7. Select the checkbox next to **tibco.bwce.sample.palette.jms.Basic.application**.
8. Click **Debug**.

The sample now runs in the debug mode.

The console window shows engine messages similar to:

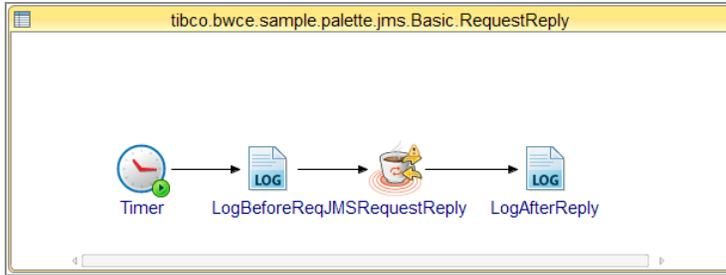
```
00:48:05.773 INFO [Framework Event Dispatcher: Equinox Container:
70dfaf13-6384-0015-159e-c83afd447683]
com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300021: All
Application dependencies are resolved for Application
[tibco.bwce.sample.palette.jms.Basic:1.0]
00:48:07.518 INFO [Thread-43] com.tibco.thor.frwk.Application -
TIBCO-THOR-FRWK-300006: Started BW Application
[tibco.bwce.sample.palette.jms.Basic:1.0]
00:48:07.523 INFO [Framework Event Dispatcher: Equinox Container:
70dfaf13-6384-0015-159e-c83afd447683]
com.tibco.thor.frwk.Application - Started by BusinessStudio,
ignoring .enabled settings.
00:48:07.621 INFO [bwEngThread:In-Memory Process Worker-1]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogBeforeReq - Sending The Request
00:48:07.645 INFO [bwEngThread:In-Memory Process Worker-2]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogAfterReply - ***** JMSRequestReply
Received the Response ***** Sending A Reply Back For Request
Message Received *****
00:48:07.645 INFO [bwEngThread:In-Memory Process Worker-3]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogReceiveMessage - Received a Queue
Message. Request = Sending a Request JMS Message. JMSMessageID =
ID:EMS-SERVER.95A15639F97F35:1
```

## Understanding the Configuration

The project uses the following two processes:

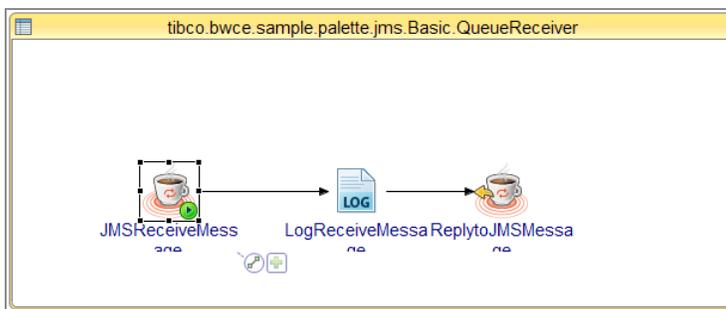
- The Request Reply Process (**RequestReply.bwp**)

The JMS Request Reply activity sends a message to a queue and waits for the response by setting a **replyTo** destination to the message.



- The JMS QueueReceiver Process (**QueueReceiver.bwp**)

The JMS QueueReceiver activity in the second process receives the message from the queue and replies by setting the **correlation ID** and **replyTo** from the incoming message.



## Testing the JMS Basic Sample

### Before you begin

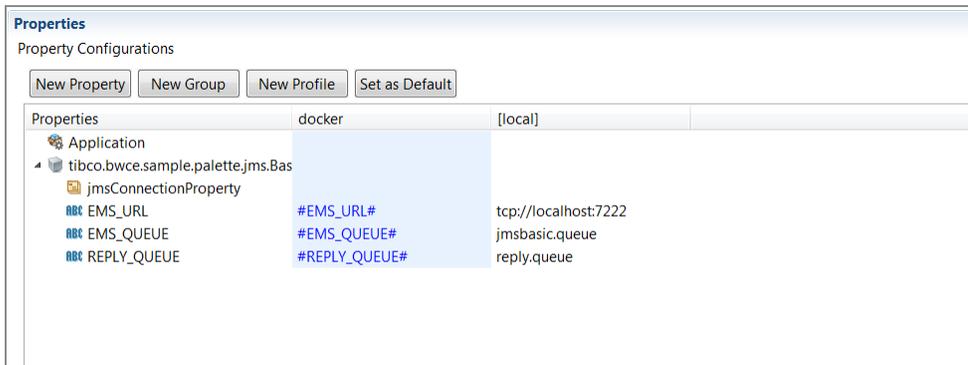
- The Cloud Foundry environment updated with TIBCO BusinessWorks Container Edition buildpack.
- A user provided service instance for TIBCO EMS Server. Read the procedure for creating a user provided service from the Cloud Foundry documentation. For more information, see <https://docs.cloudfoundry.org/>.
- The cf command line interface tool installed on your machine.

## Setting the Default Application Profile

### Procedure

1. From the Project Explorer expand the **tibco.bwce.sample.palette.jms.Basic.application**.
2. Expand the **Package Units** folder and click **Properties**.
3. Select the **docker** profile and **Set As Default**.

If you do not have a profile named **docker**, click **New Profile** to create a profile and name it **docker**.



## Generating an Application Archive File

### Procedure

1. Expand the Package Unit and select **Overview**.
2. In the **Overview** window select **Export Application for Deployment**.
3. Enter the location of your EAR file.

## Building an Application Image

### Procedure

1. Copy the Dockerfile from the samples directory to the location where you placed the EAR file.
2. From the Docker terminal, navigate to the folder where the EAR and Dockerfile are stored.
3. In the Dockerfile, make sure the base image points to the TIBCO BusinessWorks

Container Edition runtime base image.

4. Make sure the EAR file name and the path is correct.
5. Run the following command to generate the application image:

```
docker build -t bwce-jms-basic-app .
```

## Test your Application Locally in a Docker Setup

### Before you begin

- Docker should be installed on your machine.

### Procedure

1. Run the command in a docker terminal.

```
docker run -ti -e "EMS_URL=tcp://52.8.233.34:7222" -e "EMS_
QUEUE=jmsbasic.queue" -e "REPLY_QUEUE=reply.queue" -e "BW_
PROFILE=docker" bwce-jms-basic-app
```

2. Run the following command to views the logs to make sure that the application has successfully started.

```
docker logs <container name>
```

3. Check the logs to make sure application has run successfully.

You should see some messages such as:

```
00:48:07.621 INFO [bwEngThread:In-Memory Process Worker-1]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogBeforeReq - Sending The Request
00:48:07.645 INFO [bwEngThread:In-Memory Process Worker-2]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogAfterReply - ***** JMSRequestReply
Received the Response *****
Sending A Reply Back For Request Message Received *****
```

```
00:48:07.645 INFO [bwEngThread:In-Memory Process Worker-3]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogReceiveMessage - Received a Queue
Message.
Request = Sending a Request JMS Message. JMSMessageID = ID:EMS-
SERVER.95A15639F97F35:1
```

## Test Your Application in a Kubernetes Setup on the Google Cloud Platform

### Before you begin

- Google Cloud account with a project and cluster
- Google Cloud SDK
- Kubectl

### Procedure

1. Run these commands from a terminal window:

```
gcloud auth login
gcloud config set project <your project name>
gcloud config set container/cluster <your cluster name>
gcloud container clusters get-credentials <your cluster name> --zone <your
cluster zone name>
kubectl get nodes
```

2. Tag the application image created in the previous step.

```
docker tag bwce-jms-basic-app gcr.io/<your project name>/bwce-jms-
basic-app
```

3. Push your application image to Google Container Registry.

```
gcloud docker push gcr.io/<your project name>/bwce-jms-basic-app
```

4. Confirm that the image is present in the Google Container Registry.
5. Open the `manifest.yml` file and update the application image name. Ensure the image name follows the format

```
gcr.io/<your project name>/<image name>
```

6. Type in the following command to create the service:

```
kubectl create -f manifest.yml
```

7. To check that application has started successfully, run the following command:

```
kubectl logs <pod name>
```

8. When the application is successfully deployed, the console will display messages such as:

```
00:48:07.621 INFO [bwEngThread:In-Memory Process Worker-1]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogBeforeReq - Sending The Request
00:48:07.645 INFO [bwEngThread:In-Memory Process Worker-2]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogAfterReply - ***** JMSRequestReply
Received the Response *****
Sending A Reply Back For Request Message Received *****
00:48:07.645 INFO [bwEngThread:In-Memory Process Worker-3]
c.t.b.p.g.L.t.b.s.p.j.B.m.LogReceiveMessage - Received a Queue
Message. Request =
Sending a Request JMS Message. JMSMessageID = ID:EMS-
SERVER.95A15639F97F35:1
```

## JMS Basic with Consul

The JMS Basic with Consul sample uses the JMS Queue Receiver and Reply to JMS Message activities to receive and respond to JMS messages. The JMS Queue Receiver activity is triggered by a JMS Request Reply activity from another process. The main intention of the sample is to demonstrate how to attach your TIBCO BusinessWorks Container Edition application to an Application config management Server like Consul to externalize config management properties.

### Before you begin

- TIBCO Enterprise Message Service must be running.
- Consul image

## Testing in TIBCO Business Studio for BusinessWorks

This sample uses the JMS service to return information about a city given a zip code. The sample also provides the distance between two cities as defined by their zip codes.

### Before you begin

- TIBCO Enterprise Message Service must be running.

### Procedure

1. In the samples directory, select **palette > jms > Consul** and double-click **tibco.bwce.sample.palette.jms.Consul**.  
For more information, see [Accessing Samples](#).
2. In **Project Explorer** expand the **tibco.bwce.sample.palette.jms.Consul**.
3. Fully expand the **Processes** directory and double-click **ZipInfoService.bwp**.
4. Verify your TIBCO Enterprise Message Service connection.
  - a. Fully expand the **Resources** directory.
  - b. Double-click **JMSConnectionResource.jmsConnResource**.
  - c. In the **Basic Configuration** dialog, click the **Test Connection** button to verify the connection.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of **Debug Configuration** wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bwce.sample.palette.jms.Consul.application**.
8. Click **Debug**.

This runs the sample in Debug mode.

- In **TIBCO Business Studio for BusinessWorks**, click the **Terminate** icon  to stop the process.

## Result

After the application has successfully started, you can see the output similar to:

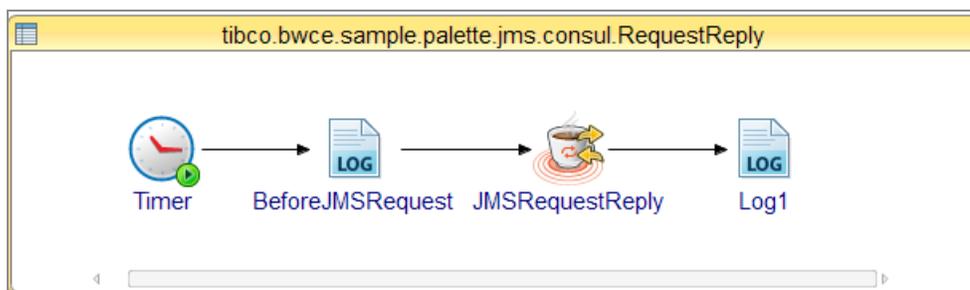
```
22:16:36.973 INFO [bwEngThread:In-Memory Process Worker-1]
c.t.b.p.g.L.t.b.s.p.j.C.BeforeJMSRequest - Sending a JMS Request Message
to queue consul.queue
22:16:37.004 INFO [bwEngThread:In-Memory Process Worker-2]
c.t.b.p.g.L.t.b.s.p.j.C.LogReceiveMessage - JMSReceiver Received a
Message
22:16:37.028 INFO [bwEngThread:In-Memory Process Worker-3]
c.t.b.p.g.L.t.b.s.p.jms.Consul.Log1 - Received a Reply From JMSReceiver
==*** Sending A Reply Back For Request Message Received *****
```

## Understanding the Configuration

The project uses two processes

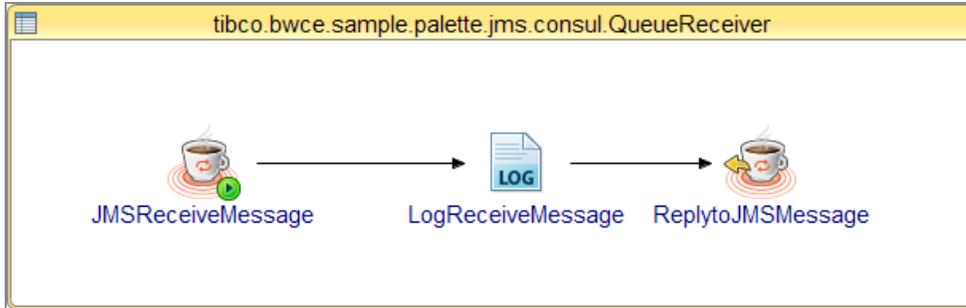
- The JMS Request Reply activity (**RequestReply.bwp**)

The JMS Request Reply activity sends a message to a queue and waits for the response by setting a replyTo destination to the message.



- The JMS QueueReceiver activity (**QueueReceiver.bwp**)

The JMS QueueReceiver activity in the second process receives the message from the queue and replies by setting the correlation ID and replyTo from the incoming message.



## Testing the Sample

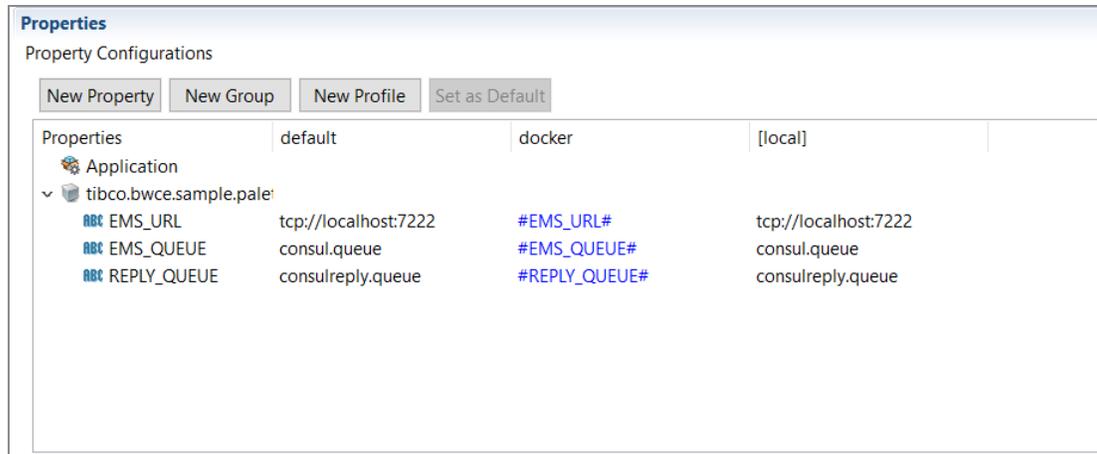
### Before you begin

- TIBCO BusinessWorks Container Edition runtime base image.  
For more information to create the runtime base image, see [Creating the TIBCO BusinessWorksTrademark\(TM\) Container Edition Application Docker Image](#).
- Connection to EMS
- Consul server should be running. For more information, see [Setting up the Consul Server](#) for more information.
- For Kubernetes
  - Google Cloud SDK installed on your machine.
  - Kubernetes tool installed on your machine.
  - Kubernetes project with cluster should be setup.

## Setting the Default Application Profile

### Procedure

1. From the Project Explorer expand the **tibco.bwce.sample.palette.jms.Consul.application** .
2. Expand the **Package Units** folder and click **Properties**.
3. Select the **docker** profile and set as default. If you do not have a profile named **docker**, click **New Profile** to create a profile and name it **docker**.



## Generating an Application Archive File

Follow these steps to generate the .EAR file:

### Procedure

1. Expand the Package Unit and select **Overview**.
2. In the **Overview** window select **Export Application for Deployment**.
3. Enter the location of your EAR file.

## Setting up the Consul Server

### Procedure

1. Pull the Consul image.

```
docker pull progrium/consul
```

2. Tag and push this image to Google Container Registry using the following commands:

```
docker tag progrium/consul gcr.io/<project name>/progrium/consul
gcloud docker tag gcr.io/<gcloud project name>/progrium/consul
```

### 3. Configure the yaml file as below

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: consul-server
labels:
  app: consul-service
spec:
  replicas: 1
  selector:
    app: consul-server
  template:
    metadata:
      name: consul-server
      labels:
        app: consul-server
    spec:
      containers:
        - name: consul-server
          image: your image name
          args:
            - '-server'
            - '-bootstrap'
          imagePullPolicy: Always
          ports:
            - containerPort: 8500
            - containerPort: 8400
            - containerPort: 53
            protocol: UDP
```

### 4. Run the command:

```
kubectl create -f <yaml file>
```

### 5. Get the external IP address of the Consul service and from the browser launch Consul UI using

```
http://External IP
```

6. Configure the Key/Value pair for your application as following:

```
<BWCE application name>/<Profile Name>/<Key Name>
```

For example,

```
tibco.bwce.sample.palette.jms.Consul.application/dev/MESSAGE
```

or

```
tibco.bwce.sample.palette.jms.Consul.application/qa/MESSAGE
```

## Test your Application Locally in a Docker Setup

### Procedure

1. Run the command in a docker terminal.

```
docker run -it -e "CONSUL_SERVER_URL=consul-server-external-ip" -e "APP_
CONFIG_PROFILE=
                                profile-name given in consul server" -e "BW_PROFILE=docker" bwce
app
```

2. Run the following command to views the logs to make sure that the application has successfully started.

```
docker logs container name
```

3. Check the logs to make sure application has run successfully.

You should see some message like below:

```
22:16:36.973 INFO [bwEngThread:In-Memory Process Worker-1]
```

```
c.t.b.p.g.L.t.b.s.p.j.C.BeforeJMSRequest - Sending a JMS Request
Message to queue consul.queue
22:16:37.004 INFO [bwEngThread:In-Memory Process Worker-2]
c.t.b.p.g.L.t.b.s.p.j.C.LogReceiveMessage - JMSReceiver Received a
Message
22:16:37.028 INFO [bwEngThread:In-Memory Process Worker-3]
c.t.b.p.g.L.t.b.s.p.jms.Consul.Log1 - Received a Reply From
JMSReceiver ==** Sending A Reply Back For Request Message
Received *****
```

## Test your Application in the Kubernetes Setup on the Google Cloud Platform

### Before you begin

- Google Cloud account with a project and cluster
- Google Cloud SDK
- Kubectl

### Procedure

1. From a terminal, run the following commands:

```
gcloud auth login
gcloud config set project <your project name>
gcloud config set container/cluster <your cluster name>
gcloud container clusters get-credentials <your cluster name> --zone <your
cluster zone name>
kubectl get nodes
```

2. Tag the application image created in the previous step.

```
docker tag bwce-jms-consul-app gcr.io/<your project name>/bwce-jms-
consul-app
```

3. Push your application image to Google Container Registry.

```
gcloud docker push gcr.io/<your project name>/bwce-jms-consul-app
```

4. Confirm that the image is present in the Google Container Registry
5. Open the `manifest.yml` file and update the application image name. Ensure the image name follows the format

```
gcr.io/<your gcloud project name>/<image name>
```

6. Navigate to the `samples` directory where the manifest file is present and run the following command to create the service:

```
kubectl create -f manifest.yml
```

7. To check that application is started successfully, run the command:

```
kubectl logs <pod name>
```

## Result

If the application deploys successfully, you can see a similar output in the console log.

```
22:16:36.973 INFO [bwEngThread:In-Memory Process Worker-1]
c.t.b.p.g.L.t.b.s.p.j.C.BeforeJMSRequest - Sending a JMS Request Message
to queue consul.queue
22:16:37.004 INFO [bwEngThread:In-Memory Process Worker-2]
c.t.b.p.g.L.t.b.s.p.j.C.LogReceiveMessage - JMSReceiver Received a
Message
22:16:37.028 INFO [bwEngThread:In-Memory Process Worker-3]
c.t.b.p.g.L.t.b.s.p.jms.Consul.Log1 - Received a Reply From JMSReceiver
===** Sending A Reply Back For Request Message Received **===
```

# Sending and Receiving Messages Using EMS-SSL

This sample demonstrates how to design and run a TIBCO BusinessWorks Container Edition application that connects to an EMS Server via SSL in your Docker and Kubernetes environments.

## Before you begin

- TIBCO Enterprise Message Service configured with SSL must be running.
- Ensure the queue `emssl.queue` has been created.

## Testing in TIBCO Business Studio for BusinessWorks

### Procedure

1. From the **File Explorer**, navigate to the samples directory and select **docker > Container > palette > jms > SSL** and double-click **tibco.bwce.sample.palette.jms.SSL**.
2. From the **Project Explorer** expand the **tibco.bwce.sample.palette.jms.SSL.application** project.
3. Verify your JDBC connection.
  - a. Expand the **Resources** directory.
  - b. Double-click **JMSConnectionResource.jmsConnResource**.
  - c. Click the **Test Connection** button to verify the connection.
4. From the **Project Explorer** expand the **Processes** folder and the **GetQueueMessage.bwp** project within that folder.
5. From the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications.

7. Select the checkbox next to **tibco.bwce.sample.palette.jms.SSL.application.application**.
8. Click **Debug**.

The sample now runs in the debug mode.

The console window shows engine messages similar to:

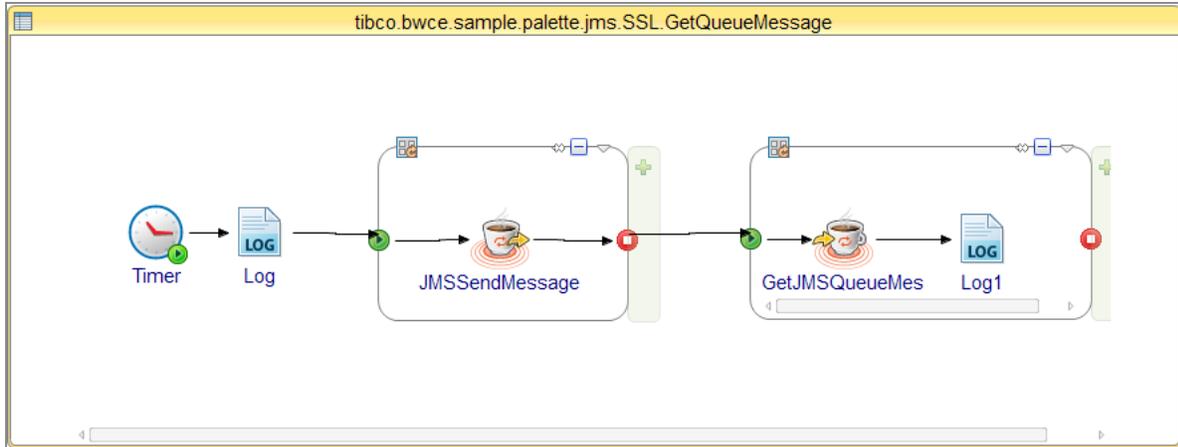
```
16:08:26.242 [PVM:In-Memory STWorkProcessor:1] INFO
c.t.b.p.g.L.t.b.p.j.G.Log - Message=Sending 5 Queue Messages
16:08:26.292 [PVM:In-Memory STWorkProcessor:1] INFO
c.t.b.p.g.L.t.b.p.j.G.Log1 - Message=Finished sending 5 Queue
messages. Getting 5 Queue messages...
16:08:26.303 [PVM:In-Memory STWorkProcessor:3] INFO
c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 1.
Message = This is message number 1
16:08:26.307 [PVM:In-Memory STWorkProcessor:5] INFO
c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 2.
Message = This is message number 2
16:08:26.311 [PVM:In-Memory STWorkProcessor:1] INFO
c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 3.
Message = This is message number 3
16:08:26.315 [PVM:In-Memory STWorkProcessor:3] INFO
c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 4.
Message = This is message number 4
16:08:26.318 [PVM:In-Memory STWorkProcessor:5] INFO
c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 5.
Message = This is message number 5
```

## Understanding the Configuration

The **JMS Send Message** activity sends the messages with the message style set to Queue.

The **Get JMS Queue Message** activity receives the JMS messages from a Queue in a loop. The activity is placed in a group and the group action is set to Repeat.

The condition loops five times and receives five messages.



### JMS Connection for EMS SSL

For this sample the EMS is configured with SSL, and uses a SSL Client Resource that in turn uses a KeyStore provider Resource. The certificate **truststore.jks** used by KeyStore Provider Resource is packaged as part of the application.

### EMS SSL URL

**Properties**  
Property Configurations

User System

Groups / Properties	Type	default
SSL_QUEUE	String	emsssl.queue
EMS_URL	String	ssl://54.67.121.175:72...

### JMS Connection with SSL Client

**Basic Configuration**

Connection Factory Type: Direct

Messaging Style: Queue/Topic

Provider URL: `ems://EMS_URL`

Use UFO Connection Factory:

Click test connection

[Click here to set preferences](#)

**Security**

**Advance Configuration**

Auto-generate Client ID:

Client ID:

**SSL**

Confidentiality

SSL Client: `tibco.bwce.sample.palette.jms.SSL.SSLClientResource`

## KeyStore Provider and Certificate

**Keystore**

Provider:

URL: `Resources/tibco/bwce/sample/palette/jms/SSL/truststore.jks`

Password:

Type: JKS

Refresh Interval: `3600000` Milliseconds

# Testing the Sample

## Before you begin

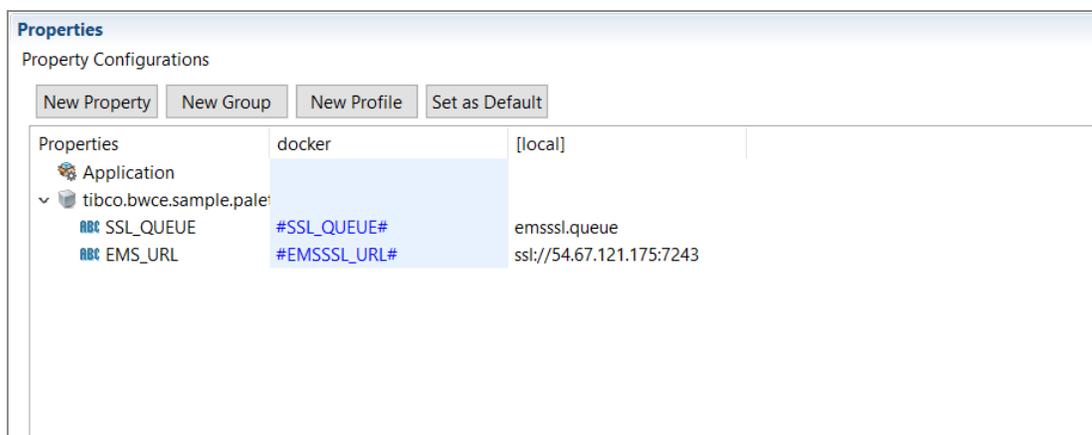
- TIBCO BusinessWorks Container Edition runtime base image.  
For more information to create the runtime base image, see [Creating the TIBCO BusinessWorksTrademark\(TM\) Container Edition Application Docker Image](#).
- Connection to the EMS server
- For Kubernetes
  - Google Cloud SDK installed on your machine.
  - Kubernetes tool installed on your machine.
  - Kubernetes project with cluster should be setup.

## Setting the Default Application Profile

### Procedure

1. From the Project Explorer expand the **tibco.bwce.sample.palette.jms.SSL.application**.
2. Expand the **Package Units** folder and click **Properties**.
3. Select the **docker** profile and **Set As Default**.

If you do not have a profile named **docker**, click **New Profile** to create a profile and name it **docker**.



## Generating an Application Archive File

Follow these steps to generate the .EAR file:

### Procedure

1. Expand the Package Unit and select **Overview**.
2. In the **Overview** window select **Export Application for Deployment**.
3. Enter the location of your EAR file.

# Creating the TIBCO BusinessWorks Container Edition Base Docker Image for Linux Containers

## Before you begin

- Download the TIBCO BusinessWorks Container Edition runtime zip file, `bwce-runtime-<version>.zip`, from <http://edelivery.tibco.com>.

To download this file,

1. Select **Container** from the **Operating Systems** drop down list.
  2. Read and Accept the **TIBCO End User License Agreement**.
  3. Select the radio button for **Individual file Download**.
  4. Click the **+** sign to view the individual components and select `bwce-runtime-<version>.zip`.
- An installation of Docker.

**i Note:** TIBCO ActiveMatrix BusinessWorks™ does not ship docker folder or scripts with it, because the scripts can be pulled directly from GitHub without installing TIBCO BusinessWorks Container Edition separately.

## Procedure

1. Navigate to the `TIBCO_HOME/bwce/<version>/docker` directory.
2. Copy the `bwce-runtime-<version>.zip` file to the `TIBCO_HOME/bwce/<version>/docker/resources/bwce-runtime` folder.
3. Open a command terminal and run the following command from the `TIBCO_HOME/bwce/<version>/docker` folder:

```
docker build -t TAG-NAME .
```

For example,

```
docker build -t tibco/bwce:latest .
```

By default, the `debian:bookworm-slim` image is used to create base docker image for TIBCO BusinessWorks Container Edition. The following is the dockerfile content.

- Dockerfile content for `debian:bookworm-slim`:

```
FROM debian:bookworm-slim
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 755 /scripts/*.sh && apt-get update && apt-get --no-
install-recommends -y install unzip ssh net-tools && apt-get -
y install xsltproc && apt-get clean && rm -rf
/var/lib/apt/lists/*
RUN groupadd -g 2001 bwce \
&& useradd -m -d /home/bwce -r -u 2001 -g bwce bwce
USER bwce
ENV LANG C.UTF-8
ENV LC_ALL C.UTF-8
ENTRYPOINT ["/scripts/start.sh"]
```

You can also change the image with the following dockerfile content.

- Dockerfile content for `openSUSE`:

```
FROM opensuse/leap
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 755 /scripts/*.sh && zypper -n update && zypper -n
refresh && \
zypper -n in unzip openssh net-tools
RUN groupadd -g 2001 bwce \
&& useradd -m -d /home/bwce -r -u 2001 -g bwce bwce
USER bwce
ENTRYPOINT ["/scripts/start.sh"]
```

- Dockerfile content for `CentOS 7`:

```
FROM centos:7
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 755 /scripts/*.sh && yum -y update && yum -y install
unzip ssh net-tools
```

```
ENTRYPOINT ["/scripts/start.sh"]
```

- Dockerfile content for CentOS 9:

```
FROM quay.io/centos/centos:stream9
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 755 /scripts/*.sh && yum -y update && yum -y install
unzip ssh net-tools
ENTRYPOINT ["/scripts/start.sh"]
```

- Dockerfile content for eclipse-temurin:11-jre-alpine

```
FROM eclipse-temurin:11-jre-alpine
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 755 /scripts/*.sh && apk update && apk add unzip
openssh net-tools
RUN apk add --no--cache bash
RUN addgroup -S bwcegroup && adduser -S bwce -G bwcegroup
USER bwce
ENTRYPOINT ["/scripts/start.sh"]
```

**i Note:** To use OpenJDK11 and remove TIBCO JRE from eclipse-temurin:11-jre-alpine based container image, navigate to <https://github.com/TIBCOSoftware/bwce-docker/tree/openjdk-alpine> and clone the 'eclipse-alpine' repository to your local machine. For more information to create the base Docker image for Linux container, see "[Creating the TIBCO BusinessWorks Container Edition Base Docker Image for Linux Containers](#)"

- Dockerfile content for ubi8:

```
FROM registry.access.redhat.com/ubi8/ubi:latest
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 777 scripts/*.sh && yum install -y unzip net-tools
&& \
yum update -y; yum clean all
```

```
ENTRYPOINT ["/scripts/start.sh"]
```

- Dockerfile content for rhel7-minimal

```
FROM registry.access.redhat.com/rhel7-minimal
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 777 scripts/*.sh && microdnf install unzip net-tools
--enablerepo=rhel-7-server-rpms && \
microdnf update; microdnf clean all
ENTRYPOINT ["/scripts/start.sh"]
```



**Note:** Ensure that you have valid Red Hat subscription for using **rhel7-minimal**.

- Dockerfile content for Redhat Standard OS:

```
FROM registry.access.redhat.com/rhel7/rhel
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 777 scripts/*.sh && yum install -y unzip ssh net-
tools && \
yum update -y; yum clean all
ENTRYPOINT ["/scripts/start.sh"]
```

- Dockerfile content for AmazonLinux 2 OS:

```
FROM amazonlinux:2
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 777 scripts/*.sh && yum install -y unzip ssh net-
tools && \
yum update -y; yum clean all
RUN groupadd -g 2001 bwce \
&& useradd -m -d /home/bwce -r -u 2001 -g bwce bwce
USER bwce
ENV LANG C.UTF-8
ENV LC_ALL C.UTF-8
ENTRYPOINT ["/scripts/start.sh"]
```

- Dockerfile content for AmazonLinux 2023 OS:

```
FROM amazonlinux:2023
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 777 scripts/*.sh && yum update -y && yum install -y
unzip openssh-clients.x86_64 net-tools.x86_64
findutils && yum clean all
ENV LANG C.UTF-8
ENV LC_ALL C.UTF-8
ENTRYPOINT ["/scripts/start.sh"]
```

## Result

The TIBCO BusinessWorks Container Edition base docker image is now created. This base docker image can now be used to create docker application images.

# Test your Application Locally in a Docker Setup

## Procedure

1. Run the command in a docker terminal.

```
docker run -ti -e "EMSSSL_URL=ssl://54.167.122.175:7243" -e "SSL_
QUEUE=emssl.queue" bwce-jms-ssl-app
```

2. Run the following command to views the logs to make sure that the application has successfully started.

```
docker logs <container name>
```

3. Check the logs to make sure application has run successfully.

You should see some message like below:

```
16:19:48.00023:19:48.225 INFO [Thread-22]
com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300006: Started
```

```
BW Application [tibco.bwce.sample.palette.jms.SSL.application:1.0]
16:19:48.00023:19:48.664 INFO [bwEngThread:In-Memory Process
Worker-1] c.t.b.p.g.L.t.b.s.p.jms.SSL.Log - Sending 5 Queue
Messages
16:19:49.00023:19:49.962 INFO [bwEngThread:In-Memory Process
Worker-2] c.t.b.p.g.L.t.b.s.p.jms.SSL.Log1 - Getting 5 Queue
messages... ==This is message number 1
16:19:50.00023:19:50.222 INFO [bwEngThread:In-Memory Process
Worker-3] c.t.b.p.g.L.t.b.s.p.jms.SSL.Log1 - Getting 5 Queue
messages... ==This is message number 2
16:19:50.00023:19:50.471 INFO [bwEngThread:In-Memory Process
Worker-4] c.t.b.p.g.L.t.b.s.p.jms.SSL.Log1 - Getting 5 Queue
messages... ==This is message number 3
16:19:50.00023:19:50.725 INFO [bwEngThread:In-Memory Process
Worker-5] c.t.b.p.g.L.t.b.s.p.jms.SSL.Log1 - Getting 5 Queue
messages... ==This is message number 4
16:19:50.00023:19:50.975 INFO [bwEngThread:In-Memory Process
Worker-6] c.t.b.p.g.L.t.b.s.p.jms.SSL.Log1 - Getting 5 Queue
messages... ==This is message number 5
```

## Test your Application in the Kubernetes Setup on the Google Cloud Platform

### Before you begin

- Google Cloud account with a project and cluster
- Google Cloud SDK
- Kubectl

### Procedure

1. From a terminal, follow these steps:

```
gcloud auth login
gcloud config set project <your project name>
gcloud config set container/cluster <your cluster name>
gcloud container clusters get-credentials <your cluster> --zone <your cluster>
```

```
zone name>  
kubectl get nodes
```

2. Tag the application image created in the previous step.

```
docker tag bwce-jms-ssl-app gcr.io/<your project name>/bwce-jms-ssl-app
```

3. Push your application image to Google Container Registry.

```
gcloud docker push gcr.io/<your project name>/bwce-jms-ssl-app
```

4. Confirm that the image is present in the Google Container Registry
5. Open the `manifest.yml` file and update the application image name. Ensure the image name follows the format:

```
gcr.io/<your gcloud project>/<image name>
```

6. Navigate to the `samples` directory where the manifest file is present and type in the below command to create the service:

```
kubectl create -f manifest.yml
```

7. To check that application is started successfully, run the following command:

```
kubectl logs <pod name>
```

## Result

If the application deploys successfully, you can see a similar output in the console log.

```
16:19:48.00023:19:48.225 INFO [Thread-22]  
com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300006: Started BW  
Application [tibco.bwce.sample.palette.jms.SSL.application:1.0]
```

```
16:19:48.00023:19:48.664 INFO [bwEngThread:In-Memory Process Worker-1]
c.t.b.p.g.L.t.b.s.p.jms.SSL.Log - Sending 5 Queue Messages
16:19:49.00023:19:49.962 INFO [bwEngThread:In-Memory Process Worker-2]
c.t.b.p.g.L.t.b.s.p.jms.SSL.Log1 - Getting 5 Queue messages... ==This is
message number 1
16:19:50.00023:19:50.222 INFO [bwEngThread:In-Memory Process Worker-3]
c.t.b.p.g.L.t.b.s.p.jms.SSL.Log1 - Getting 5 Queue messages... ==This is
message number 2
16:19:50.00023:19:50.471 INFO [bwEngThread:In-Memory Process Worker-4]
c.t.b.p.g.L.t.b.s.p.jms.SSL.Log1 - Getting 5 Queue messages... ==This is
message number 3
16:19:50.00023:19:50.725 INFO [bwEngThread:In-Memory Process Worker-5]
c.t.b.p.g.L.t.b.s.p.jms.SSL.Log1 - Getting 5 Queue messages... ==This is
message number 4
16:19:50.00023:19:50.975 INFO [bwEngThread:In-Memory Process Worker-6]
c.t.b.p.g.L.t.b.s.p.jms.SSL.Log1 - Getting 5 Queue messages... ==This is
message number 5
```

## SOAP over HTTP

This sample shows how the HTTP service returns information about the city having a ZIP code. It also provides the distance between two cities as defined by their ZIP codes.

## Testing in TIBCO Business Studio for BusinessWorks

This section describes how to configure a server and client. It uses the HTTP service to return information about the city that given a ZIP code. The sample also provides the distance between the two cities as defined by their ZIP codes.

### Procedure

#### 1. Configuring Server

- a. From the **File Explorer** tab, navigate to the **samples > Container > docker > binding > soap > http > ZipCodeServiceProvider** folder and then double-click **tibco.bwce.sample.binding.soap.http.ZIPCodeServiceProvider.zip**. For more information, see [Accessing Samples](#).

- b. From the **Project Explorer** tab expand the **tibco.bwce.sample.binding.soap.http.ZIPCodeServiceProvider** project.
- c. Completely expand the **Process** directory and double-click **ZipCodeService.bwp**.
- d. Click **Run > Debug Configuration**.
- e. In the left-hand tree of **Debug Configuration** wizard, expand **BusinessWorks Application** and select **BWApplication**.
- f. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications.
- g. Select the Check box next to **tibco.bwce.sample.binding.soap.http.ZipCodeServiceProvider.application**.
- h. Click **Debug**.

The sample now runs in the debug mode.

## 2. Configuring Client

- a. From the **File Explorer** tab, navigate to the **samples > docker > binding > soap > ZipCodeServiceProvider** folder and then double-click **tibco.bwce.sample.binding.soap.http.ZIPCodeServiceClient.zip**. For more information, see [Accessing Samples](#).
- b. Update the HTTP shared resource configuration and verify the Host and Port values.  
  
If you have updated HTTPConnector.httpConnResource in the project, you must regenerate the concrete WSDL and import it into the **tibco.bwce.sample.binding.soap.http.ZipCodeServiceClient** project.
- c. From the **Project Explorer** tab expand the **tibco.bwce.sample.binding.soap.http.ZIPCodeServiceClient** project.
- d. Completely expand the **Process** directory and double-click **ZipCodeClient.bwp**.
- e. Click **Run > Debug Configuration**.
- f. In the left-hand tree of **Debug Configuration** wizard, expand **BusinessWorks Application** and select **BWApplication**.
- g. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications.

- h. Select the Check box next to **tibco.bwce.sample.binding.soap.http.ZipCodeServiceClient.application**.
- i. Click **Debug**.  
The sample now runs in the debug mode.
- j. Click **Terminate**, when all the listed operations are completely executed.

## Result

When both the applications start successfully, you can see the following message in the console log.

### For server application:

```
17:20:05.800 INFO [Thread-12] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300006: Started BW Application
[tibco.bwce.sample.binding.soap.http.ZipCodeServiceProvider.application:1.0]
```

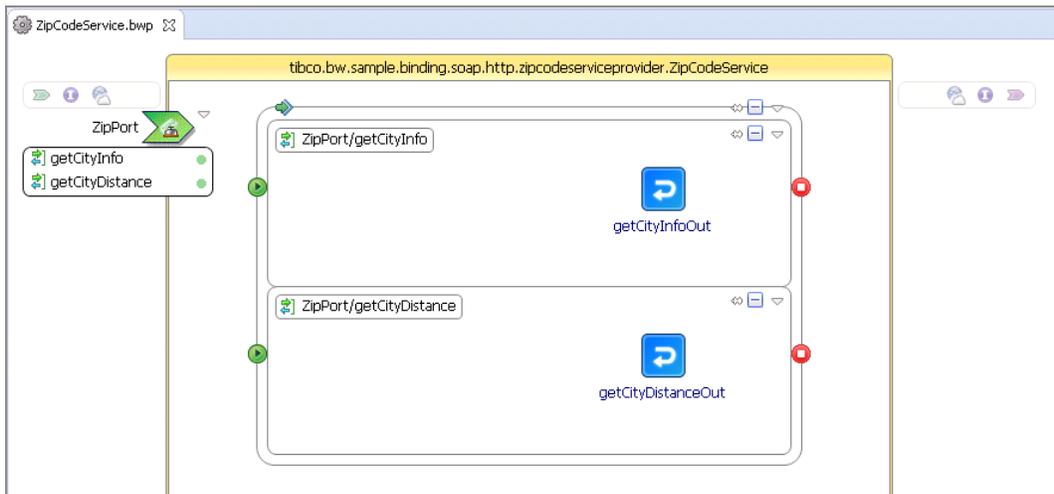
### For client application:

```
15:48:11.905 INFO [Thread-12] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300006: Started BW Application
[tibco.bwce.sample.binding.soap.http.ZipCodeServiceClient.application:1.0]
15:48:14.572 INFO [bwEngThread:In-Memory Process Worker-6]
c.t.b.p.g.L.t.b.s.b.s.h.Z.Log1 - Information of the City with Zip code
61801 is as follows.
Urbana
Illinois
Urbana, Illinois, United States
40.11
88.20761801
15:48:14.576 INFO [bwEngThread:In-Memory Process Worker-6]
c.t.b.p.g.L.t.b.s.b.s.h.Z.Log -
The distance between the Cities Urbana and Champaign is 4 miles.
```

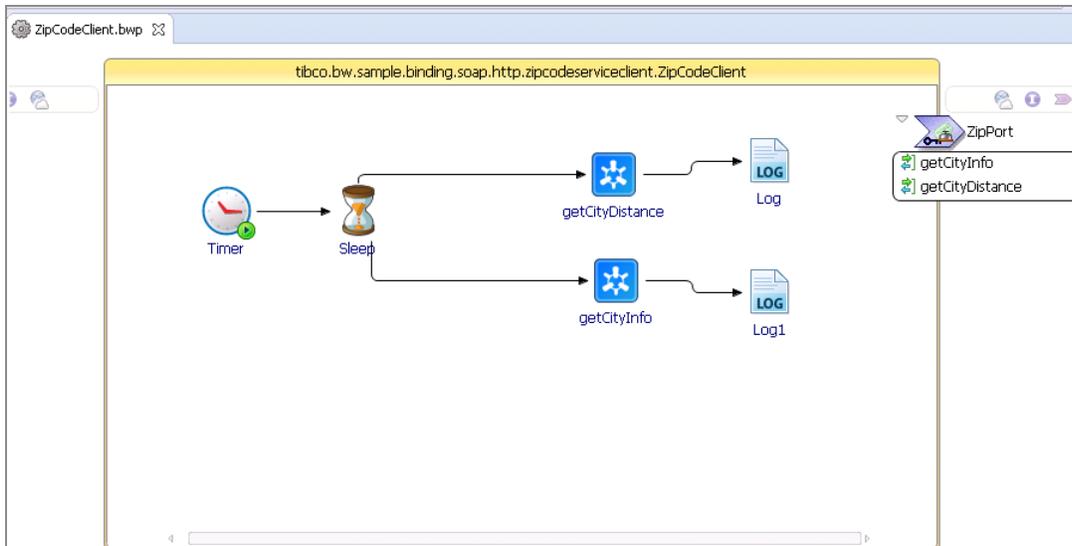
## Understanding Configuration

The project contains the **ZipCodeService** process that provides city information about a ZIP code. HTTP is used for the transport. The project represents a simple HTTP-based service using SOAP.

The **getCityDistance** and **getCityInfo** operations are implemented in the **ZipCodeService** process.



The sample also includes a client process called **ZipCodeClient.bwp**, which invokes the **ZipCodeService** process.



## Testing the Sample

This section describes the process of testing this sample in the Docker environment.

### Before you begin

- TIBCO BusinessWorks Container Edition runtime base image.

For more information to create the runtime base image, see [Creating the TIBCO BusinessWorksTrademark\(TM\) Container Edition Application Docker Image](#).

- For Kubernetes
  - Google Cloud SDK installed on your machine.
  - Kubernetes tool installed on your machine.
  - Kubernetes project with cluster should be setup.

## Generating an Application Archive File

Follow these steps to generate the .AAR file:

### Procedure

1. Click **tibco.bwce.sample.binding.soap.http.ZipCodeServiceProvider.application**
2. Expand **Package Unit** and select **Overview**.
3. In the **Overview** window, select **Export Application for Deployment**.
4. Enter the location of your EAR file.

## Building an Application Image

### Before you begin

Ensure that you have installed Docker on your machine.

### Procedure

1. Copy the Docker file from the samples directory to the location where you have placed the EAR file.
2. From the Docker terminal, navigate to the folder where the EAR and Docker file are stored.

3. In the Dockerfile, ensure that the base image points to the TIBCO BusinessWorks Container Edition on runtime base image. Also, ensure that the EAR file name and the folder path are correct.
4. Run the following command to generate the application image:

```
docker build -t bwce-soap-http-app.
```

## Testing the Application Locally in a Docker Setup

### Procedure

1. Run the following command on a docker terminal to run the application image.

```
docker run -d -p 8080:8080 bwce-soap-http-app
```

2. Run the following command to view the logs to make sure that the application has started successfully.

```
docker logs <container name>
```

3. Check the logs for the server applications to ensure that they have started successfully.

## SOAP Client

The ZipCodeServiceClient is a SOAP client for the ZipCodeServiceProvider to trigger the StockQuote application.

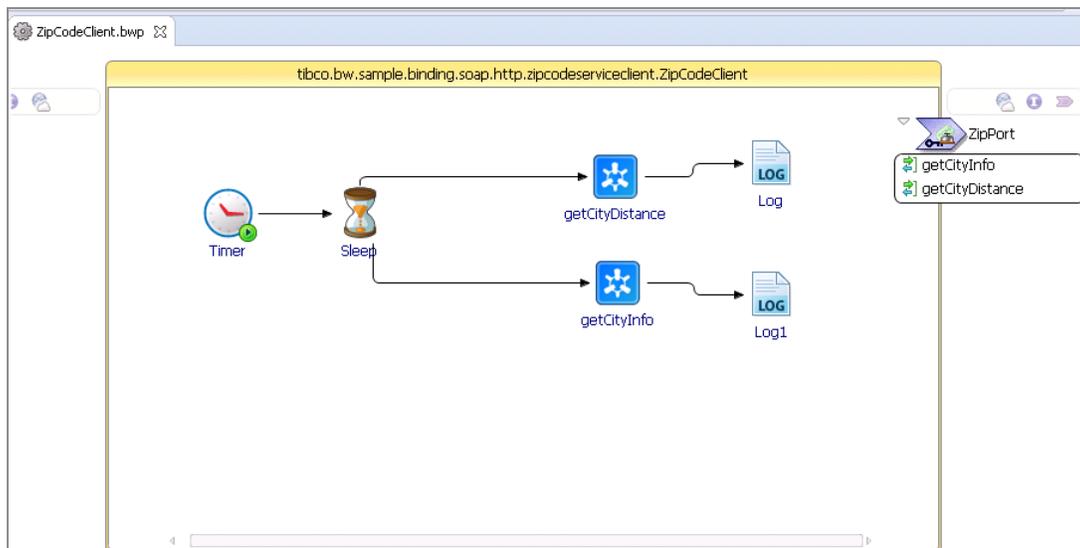
### Procedure

1. From the **File Explorer** tab, navigate to the **samples > docker > binding > soap > http > StockQuoteClient** folder and then double-click **tibco.bwce.sample.binding.soap.http.ZIPCodeServiceClient.zip**. For more information, see [Accessing Samples](#).
2. From the **Project Explorer** tab expand the **tibco.bwce.sample.binding.soap.http.ZIPCodeServiceClient** project.

3. Click **Run > Debug Configuration**.
4. In the left-hand tree of **Debug Configuration** wizard, expand **BusinessWorks Application** and select **BWApplication**.
5. Click the **Applications** tab and, then click the **Deselect All** button if you have multiple applications.
6. Select the checkbox next to **tibco.bwce.sample.binding.soap.http.ZipCodeServiceClient.application**.
7. Click **Debug**.

The sample now runs in the debug mode.

The Process uses an invoke Activity to call out the ZipCodeServiceProvider application. The process can be configured by dragging the concrete WSDL, in the case ZipInfo\_gen1.wsdl onto the workspace canvas and selecting the **Invoke Operation** option.



The reference created is set to type Binding - Reference. Create a SOAP/HTTP reference binding that uses an HTTP Client Resource.

The default host for the HTTP Client Resource should point to your machine IP on which your **bwce-soap-http-app** application is running.

## Result

When client applications start successfully, you can see the following messages in the console log.

```
15:48:11.905 INFO [Thread-12] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300006: Started BW Application
[tibco.bwce.sample.binding.soap.http.ZipCodeServiceClient.application:1.0]
15:48:14.572 INFO [bwEngThread:In-Memory Process Worker-6]
c.t.b.p.g.L.t.b.s.b.s.h.Z.Log1 - Information of the City with Zip code 61801 is as follows.
Urbana
Illinois
Urbana, Illinois, United States
40.11
88.20761801
15:48:14.576 INFO [bwEngThread:In-Memory Process Worker-6]
c.t.b.p.g.L.t.b.s.b.s.h.Z.Log -
The distance between the Cities Urbana and Champaign is 4 miles.
```

## Testing Your Application in Kubernetes Setup on the Google Cloud Platform

### Before you begin

- Google Cloud account with a project and cluster
- Google Cloud SDK
- Kubectl

### Procedure

1. From the secure terminal, run the following command:

```
gcloud auth login
gcloud config set project <your project name>
gcloud config set container/cluster <your cluster name>
gcloud container clusters get-credentials <your cluster name> --
zone <your cluster zone name>
kubectl get nodes
```

2. Tag the application image created in the previous step.

```
docker tag bwce-soap-http-app gcr.io/<your project name>/bwce-soap-http-app
```

3. Push your application image to Google Container Registry.

```
gcloud docker push gcr.io/<your project name>/bwce-soap-http-app
```

4. Confirm that the image is present in the Google Container Registry.
5. Open the `manifest.yml` file and update the application image name. Ensure that the image name follows the following format:

```
gcr.io/<your gcloud project name>/<image name>
```

6. Navigate to the `samples` directory where the manifest file is present and run the following command to create the service:

```
kubectl create -f manifest.yml
```

7. To check that the application has started successfully, run the following command:

```
kubectl logs <pod name>
```

## Result

When the application deploys successfully, you see the following output in the console log:

```
17:20:05.800 INFO [Thread-12] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300006: Started BW Application
[tibco.bwce.sample.binding.soap.http.ZipCodeServiceProvider.application: 1.0]
```

## SOAP Client

The `ZipCodeServiceClient` is a SOAP client for the `ZipCodeServiceProvider` to trigger the `StockQuote` application.

For more information, see [SOAP Client](#) to access the `tibco.bwce.sample.binding.soap.http.ZipCodeServiceClient.zip`

The default host for the HTTP Client Resource should point to your external IP for the **ZipCodeServiceProvider** application is running.

## Result

When client applications start successfully, you can see the following messages in the console log.

```
15:48:11.905 INFO [Thread-12] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300006: Started BW Application
[tibco.bwce.sample.binding.soap.http.ZipCodeServiceClient.application:1.0]
15:48:14.572 INFO [bwEngThread:In-Memory Process Worker-6]
c.t.b.p.g.L.t.b.s.b.s.h.Z.Log1 - Information of the City with Zip code
61801 is as follows.
Urbana
Illinois
Urbana, Illinois, United States
40.11
88.20761801
15:48:14.576 INFO [bwEngThread:In-Memory Process Worker-6]
c.t.b.p.g.L.t.b.s.b.s.h.Z.Log -
The distance between the Cities Urbana and Champaign is 4 miles.
```

# Collecting Process Instance, Activity Instance, and Transition Statistics

In this sample, you will use the `tibco.bwce.sample.application.execution.event.subscribe` sample to collect statistics for process instances, activity instances, and transitions in the `tibco.bwce.sample.palette.http.RequestResponse` sample project.

When working this sample, any of the following application statistics can be collected.

## Process Instance Statistics

Statistic	Description
Application Name	Name of the application.

<b>Statistic</b>	<b>Description</b>
Application Version	Version of the application.
Module Name	Name of the TIBCO BusinessWorks Container Edition module.
Module Version	Version of the TIBCO BusinessWorks Container Edition module.
Component Process Name	Name of process configured to a component. If the process is a non in-lined sub process, this could be empty.
Job ID	Job ID of the process.
Parent Process Name	If the process is an in-lined sub process, the name of the parent process.
Parent Process ID	If the process is an in-lined sub process, the instance ID of the parent process.
Process Name	Name fo the process.
Process Instance ID	Instance ID of the process.
Start Time	Process instance start time.
End Time	Process instance end time.
Elapsed Time	<p>Elapsed time of an activity is the time difference (in milliseconds) between start time and end time of the activity. Between the start and end time, control may get switched with other activities from the other jobs. This is the time taken to run an activity plus all the delays in acquiring resources like engine threads, JDBC connections, network, and so on.</p> <p>The elapsed time includes the execution time plus time taken for evaluating all the forward transitions from that particular activity and getting the next activity ready to run, which includes executing its input mapping if all dependencies are met.</p>

Statistic	Description
Eval Time	The Eval Time for an activity is the actual time (in milliseconds) used by the activity itself to complete while using the engine thread. Asynchronous activities may use other threads not included in this time.
Status	Status of process instance, for example: Completed or Faulted.

### Activity Instance Statistics

Statistic	Description
Application Name	Name of the application.
Application Version	Version of the application.
Module Name	Name of the TIBCO BusinessWorks Container Edition module.
Module Version	Version of the TIBCO BusinessWorks Container Edition module.
Activity Name	Name of the activity.
Process Name	Name of the process.
Process Instance ID	Instance ID of the process.
Start Time	When the activity instance started.
End Time	When the activity instance ended.
Eval Time	The time between the beginning and end of the evaluation period for the activity. If the activity completes in one step, the evalTime and elapsedTime would be the same. However, some activities, such as <b>Request</b> , <b>Reply</b> or <b>Wait for...</b> activities typically do not complete in one step.

<b>Statistic</b>	<b>Description</b>
Elapsed Time	Elapsed time of an activity is the time difference (in milliseconds) between start time and end time of the activity. Between the start and end time, control may get switched with other activities from other jobs. This is the time taken to run an activity plus all the delays in acquiring resources like engine threads, JDBC connections, network, and so on. The elapsed time is Eval Time plus the time taken for evaluating all the forward transitions from that particular activity.
Status	Status of activity, for example: Completed, Faulted or Canceled.

### Transition Statistics

<b>Statistic</b>	<b>Description</b>
Application Name	Name of the application.
Application Version	Version of the application.
Module Name	Name of the TIBCO BusinessWorks Container Edition module.
Module Version	Version of the TIBCO BusinessWorks Container Edition module.
Transition Name	Name of the transition
Process Name	Name of the process.
Process Instance ID	Instance ID of the process.
Component Process Name	Name of process configured to a component . If the process is a non in-lined subprocess, this could be empty.
Target Activity Name	Name of the activity the transition targets.

## Procedure

1. Import the sample into TIBCO Business Studio for BusinessWorks by right-clicking in the Project Explorer pane, and selecting **Import > Existing Studio Projects into Workspace**.
2. In the Import Projects window, ensure the option **Select root directory** field is selected, and specify the location of the **tibco.bwce.sample.application.execution.event.subscribe** sample. The sample is located at `BWCE_HOME/samples/Container/docker/source/event-subscriber/tibco.bwce.sample.application.execution.event.subscriber`.
3. Click **Finish** to import the sample project.
4. In the **samples** directory, select **palette > http > RequestResponse** and double-click **tibco.bwce.sample.palette.http.RequestResponse.zip**.
5. In **Project Explorer** expand the **tibco.bwce.sample.palette.http.RequestResponse** project.
6. Fully expand the **Processes** directory and double-click **HTTP\_Request\_Response\_Example.bwp**.
7. Click **Run > Debug Configurations**.
8. At the left hand tree of **Debug Configuration** wizard, expand **BusinessWorks Application** and select **BWApplication**.
9. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bwce.sample.palette.http.RequestResponse.application**.
10. Click the **Bundles** tab and ensure the following bundles in your workspace are selected:
  - **tibco.bwce.sample.application.execution.event.subscriber (1.0.0.qualifer)**
  - **tibco.bwce.sample.palette.http.RequestResponse (1.0.0.qualifer)**
  - **tibco.bwce.sample.palette.http.RequestResponse.application (1.0.0.qualifer)**
11. In the **Arguments** tab, add the `"-Dbw.frwk.event.subscriber.instrumentation.enabled=TRUE"` property in the **VM arguments** field.

12. Click **Debug**.

This runs the sample in Debug mode.

13. Run endpoints at the prompt in the **Console** tab to obtain the endpoint for the application.

## 14. Copy the endpoint URL of the application.

## 15. Open a browser window, and paste the endpoint URL into the address bar.

16. Click the **Terminate**  icon to stop the process.

## Result

Details about process instances, activity instances, and transitions in **tibco.bwce.sample.palette.http.RequestResponse** are displayed on the **Console** tab in TIBCO Business Studio for BusinessWorks.

```
<>@BWEclipseAppNode>
ProcessInstance Auditing Event {
  Application
  Name:tibco.bw.sample.palette.http.RequestResponse.application
  Application Version:1.0
  Module Name:tibco.bw.sample.palette.http.RequestResponse
  Module Version:1.0.0.qualifier
  ProcessInstanceId:bw0a100
  Process Name:tibco.bw.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
  State:SCHEDULED
}

ProcessInstance Auditing Event {
  Application
  Name:tibco.bw.sample.palette.http.RequestResponse.application
  Application Version:1.0
  Module Name:tibco.bw.sample.palette.http.RequestResponse
  Module Version:1.0.0.qualifier
  ProcessInstanceId:bw0a100
  Process Name:tibco.bw.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
  State:STARTED
}

Activity Auditing Event {
  Application
  Name:tibco.bw.sample.palette.http.RequestResponse.application
  Application Version:1.0
```

```
Module Name:tibco.bw.sample.palette.http.RequestResponse
Module Version:1.0.0.qualifier
ProcessInstanceId:bw0a100
Process Name:tibco.bw.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
Activity Name:Incoming_HTTP_Request
State:STARTED
}

Transition Auditing Event {
  Application
Name:tibco.bw.sample.palette.http.RequestResponse.application
Application Version:1.0
Module Name:tibco.bw.sample.palette.http.RequestResponse
Module Version:1.0.0.qualifier
ProcessInstanceId:bw0a100
Process Name:tibco.bw.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
Activity Auditing Event {
  Application
Name:tibco.bw.sample.palette.http.RequestResponse.application
Application Version:1.0
Module Name:tibco.bw.sample.palette.http.RequestResponse
Module Version:1.0.0.qualifier
ProcessInstanceId:bw0a100
Process Name:tibco.bw.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
Activity Name:Incoming_HTTP_Request
Start Time:2019.01.10 11:20:50.947
End Time:2019.01.10 11:20:51.339
Eval Time:0
Elapsed Time:392
State:COMPLETED
}

Activity Auditing Event {
  Application
Name:tibco.bw.sample.palette.http.RequestResponse.application
Application Version:1.0
Module Name:tibco.bw.sample.palette.http.RequestResponse
Module Version:1.0.0.qualifier
ProcessInstanceId:bw0a100
Process Name:tibco.bw.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
Activity Name:Log1
State:STARTED
```

```
}
11:20:51.361 INFO [bwEngThread:In-Memory Process Worker-2]
com.tibco.bw.palette.generalactivities.Log.tibco.bw.sample.palette.http.
RequestResponse.Log1 - No matching 'NEWS' source found.

Activity Auditing Event {
  Application
Name:tibco.bw.sample.palette.http.RequestResponse.application
  Application Version:1.0
  Module Name:tibco.bw.sample.palette.http.RequestResponse
  Module Version:1.0.0.qualifier
  ProcessInstanceId:bw0a100
  Process Name:tibco.bw.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
  Activity Name:Log1
  Start Time:2019.01.10 11:20:51.350
  End Time:2019.01.10 11:20:51.362
  Eval Time:11
  Elapsed Time:12
  State:COMPLETED
}

ProcessInstance Auditing Event {
  Application
Name:tibco.bw.sample.palette.http.RequestResponse.application
  Application Version:1.0
  Module Name:tibco.bw.sample.palette.http.RequestResponse
  Module Version:1.0.0.qualifier
  ProcessInstanceId:bw0a100
  Process Name:tibco.bw.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
  Start Time:2019.01.10 11:20:50.930
  End Time:2019.01.10 11:20:51.375
  Elapsed Time:445
  Eval Time:11
  State:COMPLETED
}
```

✔ **Tip:** You can use this sample to build your own application statistics collection tool. Follow these steps to do this:

1. From the Project Explorer tab, select **tibco.bwce.sample.application.execution.event.subscriber > src > tibco.bwce.sample.application.execution.event.subscriber > BWEventSubscriber.java**.
2. Update `handleEvent(Event event)` method based on your use case.
3. Save your changes to the project.
4. Export the project as a plug-in by right clicking on **tibco.bwce.sample.application.execution.event.subscriber** and selecting **export > Export > Plug-in Development > Deployable plug-ins and fragments**.
5. In the Export wizard, ensure the `tibco.bwce.sample.application.execution.event.subscriber` project is selected, and specify a location to export the plug-in.
6. After the project has been exported as a plug-in to the location you specified, locate the JAR in the plug-ins folder and copy it to a temporary folder. From the temporary folder use the Docker file given below to copy these JAR into the base Docker image.

```
FROM tibco/bwce:latest
COPY . /resources/addons/jars
```

Your application statistics collection tool has been added to your run time environment. You can see the application statistics after you create bwce application image from the base docker image.

# Unit Testing

---

Unit testing in TIBCO BusinessWorks Container Edition consists of verifying whether individual activities in a process are working as expected. While you can run the unit tests on processes any time during the development cycle, testing the processes before you push the application to the production environment may help you to identify issues earlier.

This section explains how to use the Maven plugin and its features to achieve unit testing of TIBCO BusinessWorks Container Edition projects.

Following are the sample projects provided with the TIBCO BusinessWorks Container Edition application:

- **tibco.bw.UT.sample.MainProcessWithUnitTestDemo.zip:** This unit test sample demonstrates the support for mock inputs to the process starter, along with assertions and mocking for main process activities, as well as for REST and SOAP services.
- **tibco.bw.UT.sample.UnitTestDemoProject.zip:** This unit test sample demonstrates the support of mock inputs for the subprocess starter, along with assertions and mocking for subprocess activities.
- **tibco.bw.UT.sample.Calculator.module.zip:** Sample shows the use case of implementing unit test cases in the Shared Module of your application.

Following are the details of the processes in the sample projects:

## **tibco.bw.UT.sample.MainProcessWithUnitTestDemo:**

- **MainProcessWithProcessStarterMocked:** Shows assertion support in the main process along with mocking support on process starter
- **RETServicBindingeMocked:** Shows mocking support on the REST Service binding
- **SOAPServiceBindingMocked:** Shows mocking support on the SOAP Service binding

## **tibco.bw.UT.sample.UnitTestDemoProject:**

- **SubProcesswithActivityAssertion.bwp:** Shows mocking support and Activity Assertion in a subprocess
- **SubProcessWithPrimitiveAssertion.bwp:** Shows mocking support and Primitive

Assertion in a subprocess

### **tibco.bw.UT.sample.Calculator.module:**

- **Calculator:** This process invokes calculation operations based on input received from the HTTPReceiver. It demonstrates support for mocking the HTTPReceiver and Primitive assertion on the "cube" (a call process activity).

In the Shared Module, you can find the following processes:

- **Division:** This process provides inputs (number 2 and number 0) to trigger a divide-by-zero exception, simulating a mock fault. These values can be modified to test different scenarios.
- **Cube:** This process provides input (number 8) for the cube operation and verifies the result of the "end" activity using an activity assertion.
- **Multiply:** This process provides inputs (number 2 and number 3) using an "input XML file" for multiplication and verifies the result of the "end" activity using an activity assertion with a reference to a gold input file on the "end" activity.
- **Square:** This process provides input (number 2) using an "input XML file" for squaring and verifies the result of the "end" activity using a primitive assertion.

## Maven Test Sample of Main Process

This sample covers mocking for activities in process, service binding, starter activity and types of activity assertions.

### **Before you begin**

- Ensure that Apache Maven is installed on your system.
- Ensure that the Maven plugin is installed correctly in your bw studio by verifying the Maven folder present under `<TIBCO_HOME>\bwce\2.x` directory.

### **Procedure**

1. In the samples directory, select **AppSpace > UnitTesting** and double-click **tibco.bw.UT.sample.MainProcessWithUnitTestDemo.zip**. For more information, see

### Accessing Samples.

2. Mavenize the project by right-clicking **tibco.bw.UT.sample.MainProcessWithUnitTestDemo.application** and selecting the **Generate POM for Application** option. The POM Generation window is displayed.
3. Enter the Parent POM details such as Group Id, Parent Artifact Id, and so on, and click **Finish**.

**i Note:** The workspace is indexed after generating the POM files for the first time and may take some time. You can continue with the following steps and allow the indexing to run in the background.

4. **tibco.bw.UT.sample.MainProcessWithUnitTestDemo.application.parent** is generated and the project is converted to Maven (Eclipse project) nature.
5. Right-click **tibco.bw.UT.sample.MainProcessWithUnitTestDemo.application.parent** and select **Run as > Maven test**.
6. Check the console to see the result.

## Result

The console displays messages similar to the following:

```
[INFO]
Tests for mainprocesswithunittestdemo.MainProcessWithProcessStarterMocked
Tests run : 1    Success : 1    Failure : 2    Errors : 0
Tests for mainprocesswithunittestdemo.RESTServiceBindingMocked
Tests run : 1    Success : 1    Failure : 0    Errors : 0
Tests for mainprocesswithunittestdemo.SOAPServiceBindingMocked
Tests run : 1    Success : 1    Failure : 0    Errors : 0

Results
Success : 3    Failure : 2    Errors : 0
```

**i Note:** Here we have covered 3 positive scenarios and 2 negative scenarios by giving incorrect value to assertions. If you want to see the result in the html form, use Maven “site” goal, while running the application.

**MainProcessWithUnitTestDemo.application**  
Last Published: 2022-04-20 | Version: 1.0.0-SNAPSHOT

### BW Test Report

#### Summary

[Summary] [Package List] [Test Cases] [Failure Details]

Tests	Errors	Failures	Skipped	Success Rate
3	0	1	0	66.67%

#### Package List

[Summary] [Package List] [Test Cases] [Failure Details]

Module	Package	Test	Errors	Failures	Skipped	Success Rate
MainProcessWithUnitTestDemo	mainprocesswithunittestdemo	3	0	1	0	66.67%

Properties Problems BW Help Console JUnit

tibco.bw.UT.sample.mainprocesswithunittestdemo.MainProcessWithProcessStarterMocked

Runs: 5/5    Errors: 0    Failures: 2

- tibco.bw.UT.sample.mainprocesswithunittestdemo.MainProcessWithProcessStarterMocked
  - Mapper
  - Mapper1
  - Mapper2
- tibco.bw.UT.sample.mainprocesswithunittestdemo.RESTServiceBindingMocked
  - Mapper
- tibco.bw.UT.sample.mainprocesswithunittestdemo.SOAPServiceBindingMocked
  - Mapper

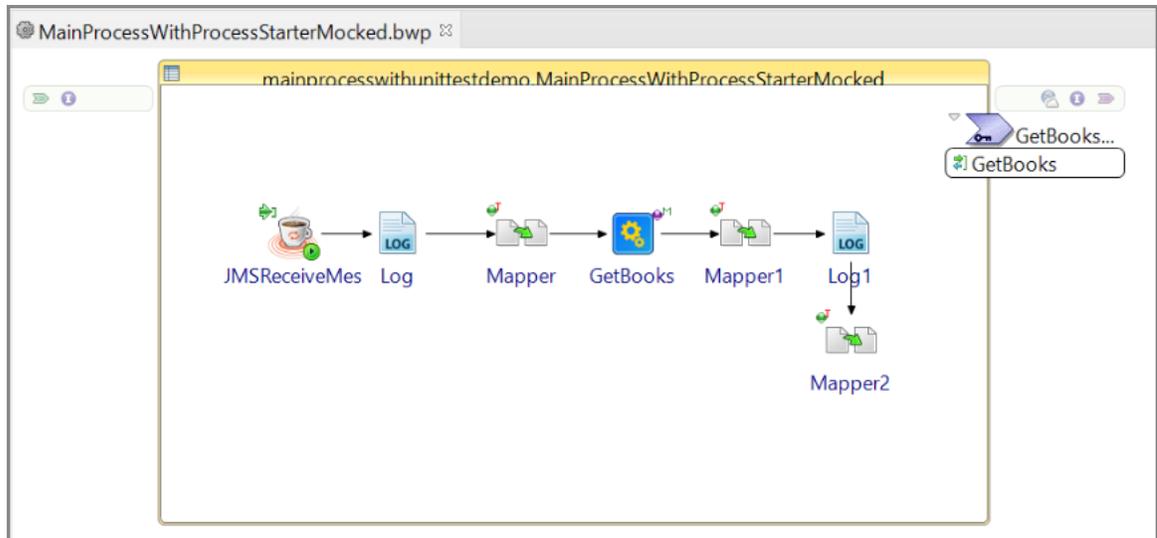
## Understanding the Configuration

The project contains the following processes.

- **MainProcessWithProcessStarterMocked.bwp:**

This process has JMSReceiver as a starter activity. Whenever a message is published on JMS queue, this process gets triggered. From the JMS message content, input for “GetBooks By author” SOAP call is constructed using mapper activity. Later in the process, “GetBooks By Author” SOAP call is made, which returns corresponding book details, and then using a mapper, the output response is converted in required format.

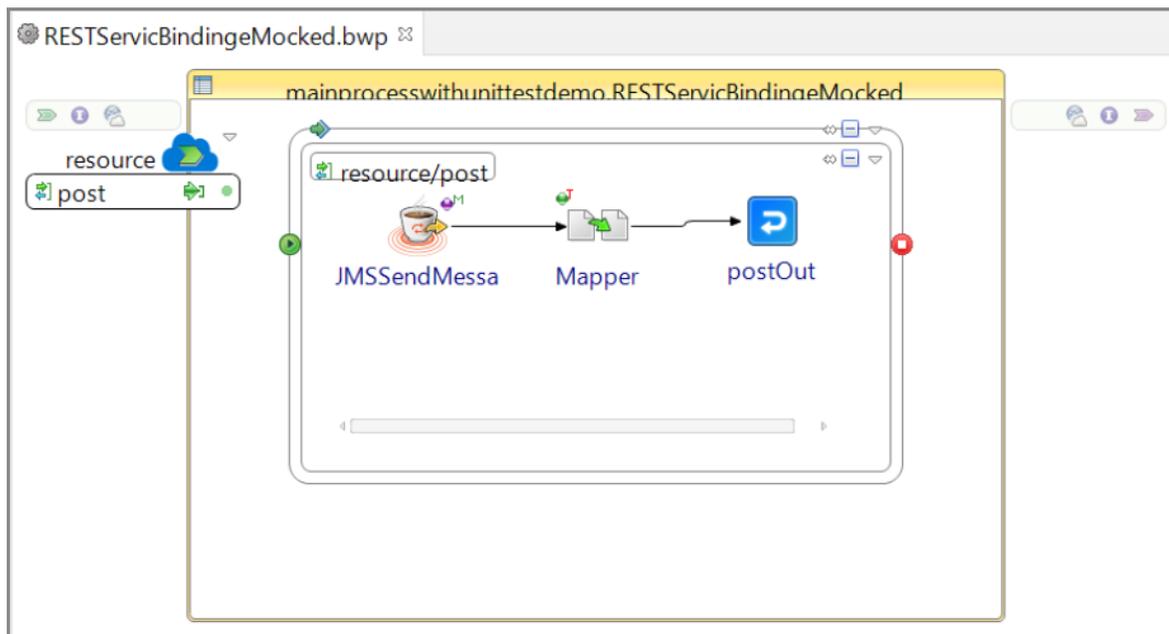
This process shows assertion support in the main process along with mocking support on process starter.



- **RESTServiceBindingeMocked.bwp:**

This process is triggered by the POST endpoint call, which requires book details like isbn, name, description, and author name in the post body. First this process publishes the book name, which it received in the POST body, as JMS message body and author name as dynamic property to a JMS queue. Later using mapper activity output for the REST endpoint is constructed and then using mapper the output is mapped to the postOut (reply activity).

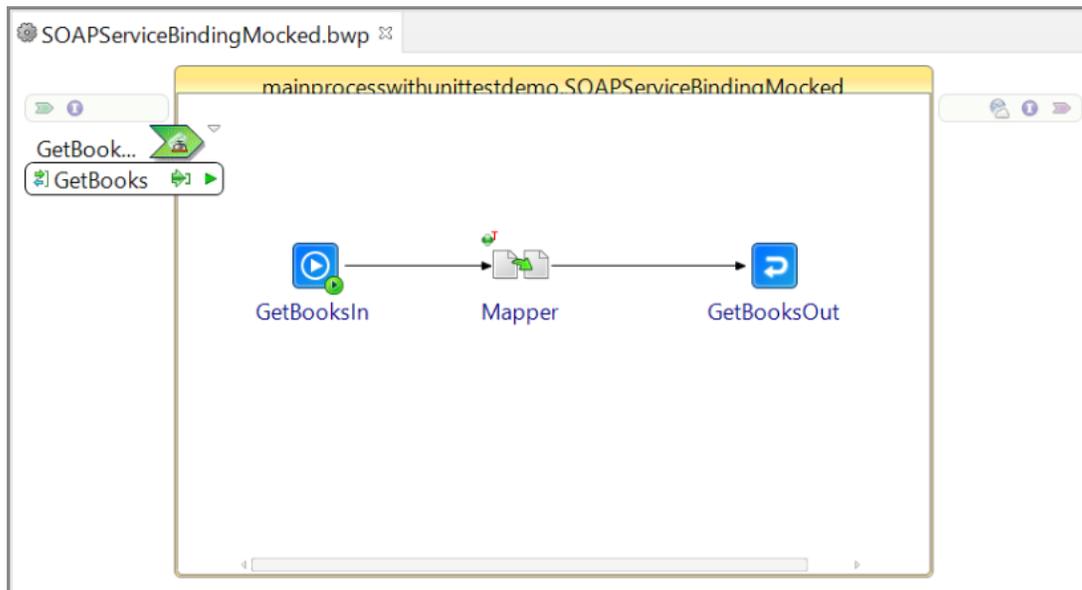
This process shows mocking support on the REST Service binding.



- **SOAPServiceBindingMocked.bwp:**

This process is triggered by the Get Books SOAP call, which receives ‘author’ as an input. After accepting author as input, depending on the author, the required output is constructed using mapper activity and then using GetBooksOut (Reply activity) the response like title, author, date isbn, and publisher is returned in the response.

This process shows mocking support on the SOAP Service binding.



### Troubleshooting

If you find the `application = null` error in the console then check the Maven version by navigating to **Window > Preferences > Maven Defaults**. The BW6 Maven Plugin Version should be the same as the plugin version you installed while installing TIBCO Business Studio for BusinessWorks. Similarly, check Maven version in the `pom.xml` file.

Refer the [Troubleshooting](#) guide for any issues.

## Maven Sample of Unit Test Demo Project

This sample covers mocking for activities in process, subprocess, service binding, starter-end activity, and types of activity assertion.

## Before you begin

- Ensure that Apache Maven is installed in your system.
- Ensure that the Maven plugin is installed correctly in your bw studio by verifying the Maven folder present under <TIBCO\_HOME>\bwce\2.x directory.

## Procedure

1. In the samples directory, select **AppSpace > UnitTesting** and double-click **tibco.bw.UT.sample.MainProcessWithUnitTestDemo.zip**. For more information, see [Accessing Samples](#).
2. In the Project Explorer, right-click **tibco.bw.UT.sample.MainProcessWithUnitTestDemo.application** and select **Generate POM for Application**. The POM Generation window is displayed.
3. Enter the Parent POM details such as Group Id, Parent Artifact Id, and so on, and click **Finish**.

**i Note:** The workspace is indexed after generating the POM files for the first time and may take some time. You can continue with the following steps and allow the indexing to run in the background.

4. **tibco.bw.UT.sample.MainProcessWithUnitTestDemo.application.parent** is generated and the project is converted to Maven (Eclipse project) nature.
5. Right-click **tibco.bw.UT.sample.MainProcessWithUnitTestDemo.application.parent** and select **Run as > Maven test**.
6. Check the console to see the result.

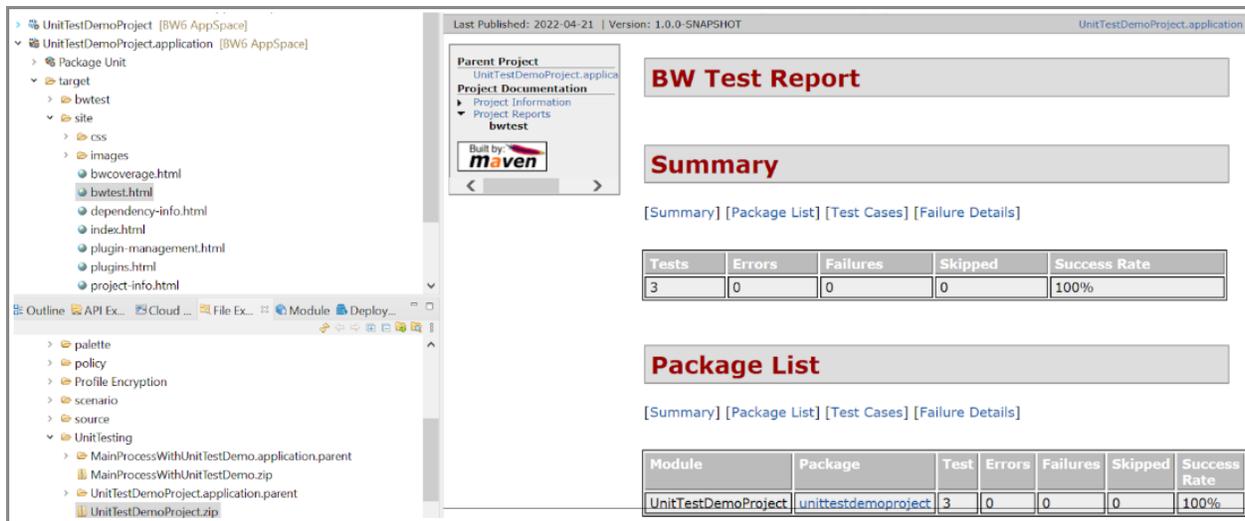
## Result

The console displays messages similar to the following:

```
Tests for unittestdemoproject.SubProcesswithActivityAssertion
Tests run : 2    Success : 2    Failure : 0    Errors : 0
Tests for unittestdemoproject.SubProcesswithPrimitiveAssertion
Tests run : 1    Success : 1    Failure : 0    Errors : 0

Results
Success : 3    Failure : 0    Errors : 0
```

**Note:** Here we have covered 3 positive scenarios. If you want to see the result in the html form, use maven “site” goal, while running the application.



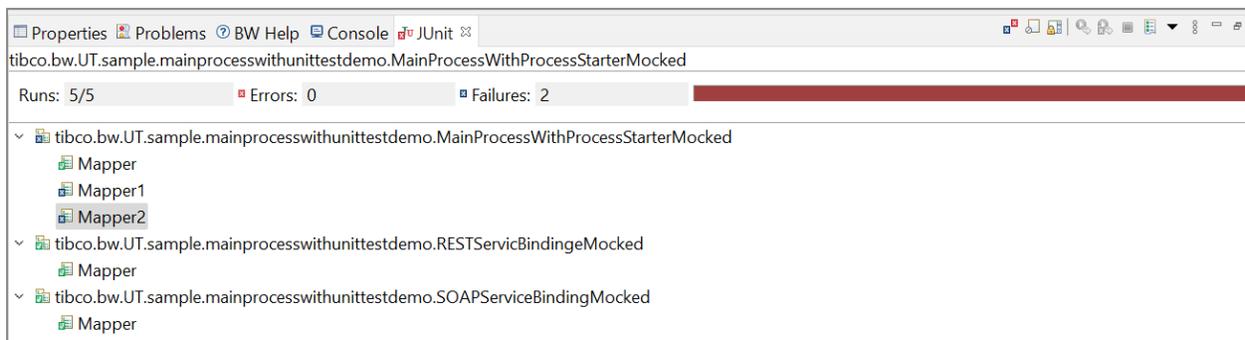
The screenshot shows a web browser displaying a 'BW Test Report' for a Maven project. The report includes a summary table and a package list table.

**Summary Table:**

Tests	Errors	Failures	Skipped	Success Rate
3	0	0	0	100%

**Package List Table:**

Module	Package	Test	Errors	Failures	Skipped	Success Rate
UnitTestDemoProject	unittestdemoproject	3	0	0	0	100%



The screenshot shows the JUnit console output for a test run. The output indicates that 5 tests passed, 0 errors occurred, and 2 failures occurred.

Runs: 5/5    Errors: 0    Failures: 2

The test run includes the following classes:

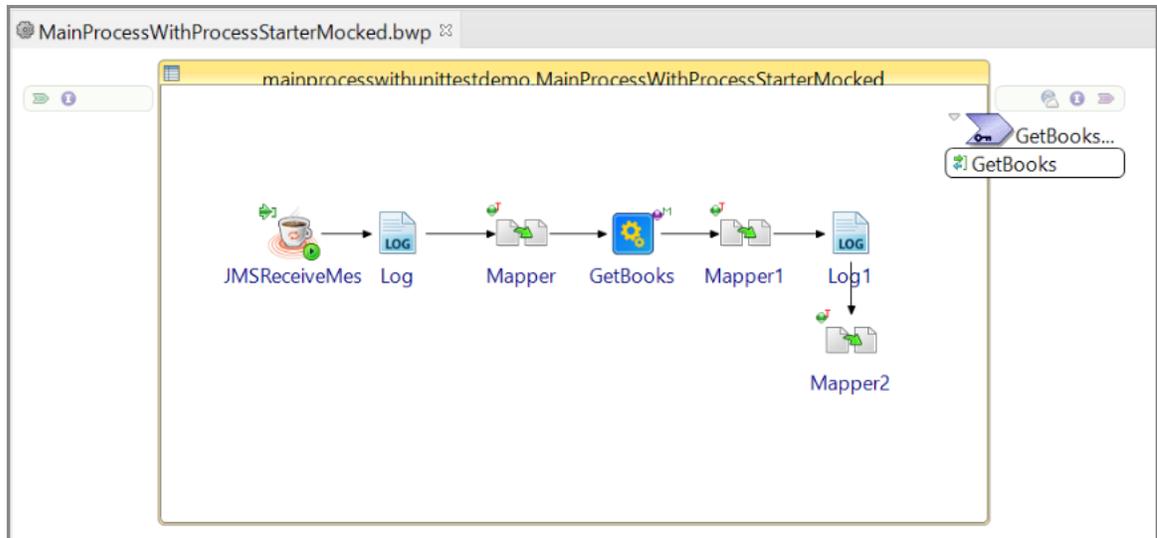
- tibco.bw.UT.sample.mainprocesswithunittestdemo.MainProcessWithProcessStarterMocked
  - Mapper
  - Mapper1
  - Mapper2
- tibco.bw.UT.sample.mainprocesswithunittestdemo.RESTServiceBindingMocked
  - Mapper
- tibco.bw.UT.sample.mainprocesswithunittestdemo.SOAPServiceBindingMocked
  - Mapper

## Understanding the Configuration

The project contains the following processes.

- **MainProcess.bwp:**

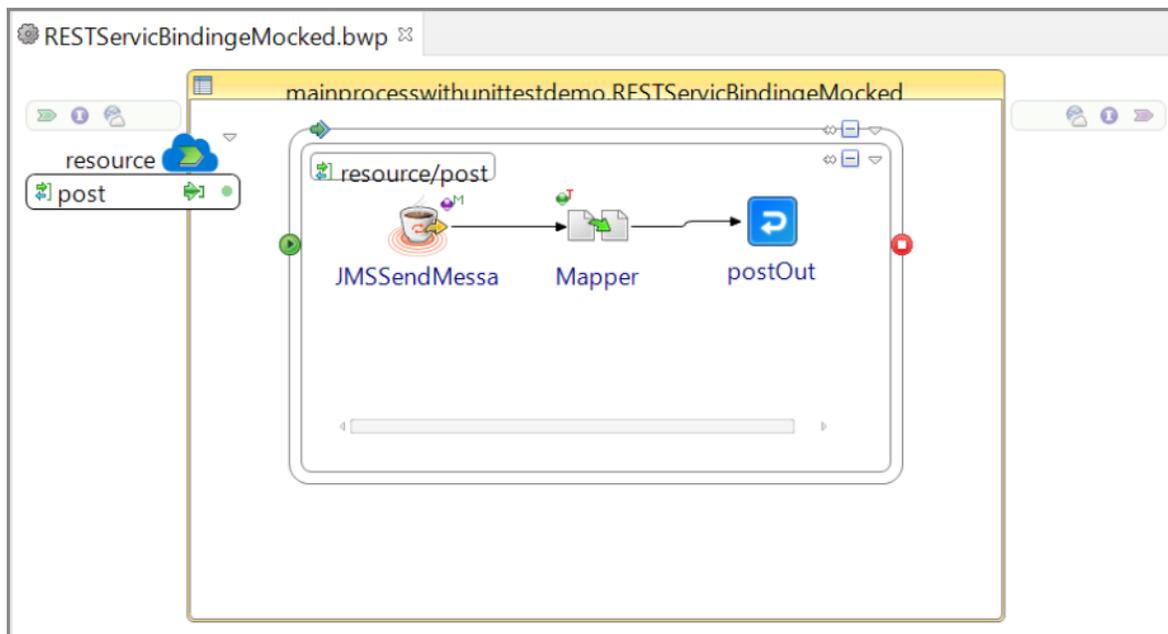
This process has a timer as a starter activity, which is executed only once. Later this process makes a subprocess call to another subprocess, *SubProcessWithPrimitiveAssertion.bwp*.



- **SubProcessWithPrimitiveAssertion.bwp:**

This process does not accept any input. First, it sends a request to a JMS destination, queue in this case, using the JMS Request Reply activity. Later, the JMS Request Reply activity body is logged and the reply message is mapped to the required format using the mapper activity. The output of the mapper is mapped with the end activity.

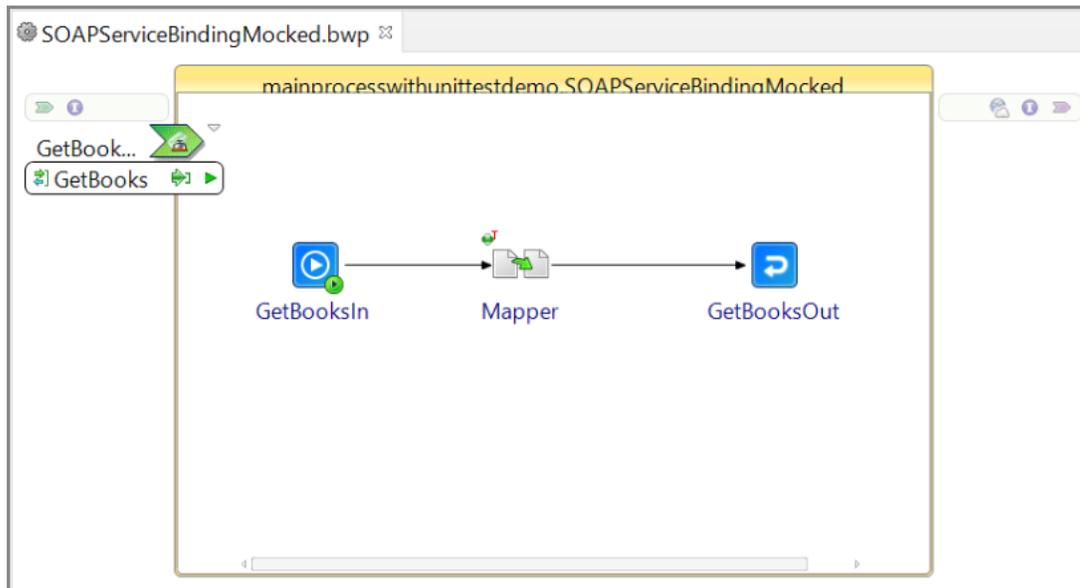
This process shows mocking support and Primitive Assertion in a subprocess.



- **SubProcesswithActivityAssertion.bwp:**

This process accepts authorID as an input of the Start activity. The book details corresponding to the the authorID are displayed in the mapper. The GetBooks SOAP call is invoked by the author as an input. The GetBooks response is logged using the log activity.

This process shows mocking support and Activity Assertion in a subprocess.



- **JMSReceiver.bwp:**

This process has a JMS Message receiver as a starter activity which gets triggered when a message is received on a queue (queue.sample). Later this process replies to the JMS message received using the Reply to JMS Message activity and then log the message.

- **SOAPService.bwp:**

This process implements SOAP Getbooks by author operation, which accepts author as an input and replies the corresponding book details of that author in the response. The response is hard-coded.

## Troubleshooting

If you find the `application = null` error in the console then check the Maven version by navigating to **Window > Preferences > Maven Defaults**. The BW6 Maven Plugin Version should be the same as the plugin version you installed while installing TIBCO Business Studio for BusinessWorks. Similarly, check Maven version in the `pom.xml` file.

Refer [Troubleshooting](#) guide for any issue

# Maven Sample of Shared Module Unit Test Cases

In this sample, a REST endpoint is exposed by using the `HTTPReceiver` activity. This activity is responsible for executing fundamental calculator operations.

Here, the behavior of the `HTTPReceiver` activity is simulated. After the operations are outlined in the mock `HTTPReceiver-mockOutput.xml` file, the corresponding actions are invoked and the obtained results are verified through assertions.

This particular sample also portrays the Test Suite feature that can encompass test cases for an application module.

## Procedure

1. From the samples directory, select **AppSpace > UnitTesting > SharedModuleUnitTestcases** and double-click **tibco.bw.UT.sample.Calculator.OperationsSM.zip**. Wait for the shared module to be imported into the workspace. Then double-click **tibco.bw.UT.sample.Calculator.module.zip**.

For more information, see [Accessing Samples](#).

2. In the Project Explorer, right-click **tibco.bw.UT.sample.Calculator.module.application** and select the **Generate pom.xml** option. The POM Generation window is displayed.

A project **tibco.bw.UT.sample.Calculator.module.application.parent** is generated if it does not exist. If it is already present, it then regenerates with updated JAR values.

3. To run the test cases irrespective of the test suites, right-click **tibco.bw.UT.sample.Calculator.module.application.parent** and select **Run as > Maven test**.
4. To see the result, check the console.

## Result

The console displays messages similar to the following:

```

[INFO] ## Running Test for Division.bwt ##
[INFO] ## Running Test for cube.bwt ##
[INFO] ## Running Test for multiply.bwt ##
[INFO] ## Running Test for square.bwt ##
[INFO] Starting Tests in Module : tibco.bw.UT.sample.Calculator.OperationsSM
[INFO] Uploading tests for Processes in Module : tibco.bw.UT.sample.Calculator.OperationsSM
tibco.bw.ut.sample.calculator.operationssm.Division
tibco.bw.ut.sample.calculator.operationssm.Cube
tibco.bw.ut.sample.calculator.operationssm.Multiply
tibco.bw.ut.sample.calculator.operationssm.Square
[INFO] <tns:CalculatorOutput xmlns:tns="http://www.example.com/namespaces/tns/1693209694314" xmlns:tns1="http://www.example.com/namespaces/tns/1693209556"
[INFO] <tns:CalculatorOutput xmlns:tns="http://www.example.com/namespaces/tns/1693209694314" xmlns:tns1="http://www.example.com/namespaces/tns/1693209556"
[INFO] <tns:CalculatorOutput xmlns:tns="http://www.example.com/namespaces/tns/1693209694314" xmlns:tns1="http://www.example.com/namespaces/tns/1693209556"
[INFO] <tns:CalculatorOutput xmlns:tns="http://www.example.com/namespaces/tns/1693209694314" xmlns:tns1="http://www.example.com/namespaces/tns/1693209556"
[INFO]
Tests for tibco.bw.ut.sample.calculator.operationssm.Division
Tests run : 1 Success : 1 Failure : 0 Skipped : 0 Errors : 0
Tests for tibco.bw.ut.sample.calculator.operationssm.Cube
Tests run : 1 Success : 1 Failure : 0 Skipped : 0 Errors : 0
Tests for tibco.bw.ut.sample.calculator.operationssm.Multiply
Tests run : 1 Success : 1 Failure : 0 Skipped : 0 Errors : 0
Tests for tibco.bw.ut.sample.calculator.operationssm.Square
Tests run : 1 Success : 1 Failure : 0 Skipped : 0 Errors : 0
Results
Success : 4 Failure : 0 Skipped : 0 Errors : 0

```

```

[INFO] ## Running Test for calculator.bwt ##
[INFO] ## Running Test for DivideByZeroFault.bwt ##
[INFO] Starting Tests in Module : tibco.bw.UT.sample.Calculator.module
[INFO] Uploading tests for Processes in Module : tibco.bw.UT.sample.Calculator.module
tibco.bw.UT.sample.Calculator.module.Calculator
[INFO] 2024-11-12T16:03:34,289 INFO [bwEngThread:In-Memory Process Worker-5] com.tibco.bw.palette.generalactivities.Log.tibco.bw.UT.sample.Calculator.n
[INFO] <tns:CalculatorOutput xmlns:tns="http://www.example.com/namespaces/tns/1693209694314" xmlns:tns1="http://www.example.com/namespaces/tns/1693209556"
[INFO] 2024-11-12T16:03:34,313 INFO [bwEngThread:In-Memory Process Worker-5] com.tibco.bw.palette.generalactivities.Log.tibco.bw.UT.sample.Calculator.n
[INFO] 2024-11-12T16:03:34,335 INFO [bwEngThread:In-Memory Process Worker-7] com.tibco.bw.palette.generalactivities.Log.tibco.bw.UT.sample.Calculator.n
[INFO] 2024-11-12T16:03:34,338 INFO [bwEngThread:In-Memory Process Worker-7] com.tibco.bw.palette.generalactivities.Log.tibco.bw.UT.sample.Calculator.n
[INFO]
Tests for tibco.bw.UT.sample.Calculator.module.Calculator
Tests run : 2 Success : 1 Failure : 0 Skipped : 0 Errors : 0
Results
Success : 1 Failure : 0 Skipped : 0 Errors : 0

```

Properties Problems BW Help JUnit Console

tibco.bw.ut.sample.calculator.operationssm.Division

Runs: 5/5 Errors: 0 Failures: 0

- tibco.bw.ut.sample.calculator.operationssm.Division
  - End
- tibco.bw.ut.sample.calculator.operationssm.Cube
  - End
- tibco.bw.ut.sample.calculator.operationssm.Multiply
  - End
- tibco.bw.ut.sample.calculator.operationssm.Square
  - End
- tibco.bw.UT.sample.Calculator.module.Calculator
  - cube

You can also configure the Maven goal to run specific test suites. For example, in the **Run As -> Maven build..** field for "Goals", you can enter the following command:

```

test -
DtestSuiteName=TestSuites\CommutativeOperations.bwts;TestSuite\Calculati
onSuite.bwts

```

## Result

When running test suites with the Maven goal, the console displays messages similar to the following:

```
[INFO] ## Running Test Suite TestSuites\CommutativeOperations.bwts ##
[INFO]   Running Test for square.bwt
[INFO]   Running Test for Division.bwt
[INFO] Starting Tests in Module : tibco.bw.UT.sample.Calculator.OperationsSM
[INFO] Uploading tests for Processes in Module : tibco.bw.UT.sample.Calculator.OperationsSM
tibco.bw.ut.sample.calculator.operationssm.Square
tibco.bw.ut.sample.calculator.operationssm.Division
[INFO] <tns:CalculatorOutput xmlns:tns="http://www.example.com/namespaces/tns/1693209694314" xmlns:tns1="http://www.example.com/namespaces/tns/16932095564"
[INFO] <tns:CalculatorOutput xmlns:tns="http://www.example.com/namespaces/tns/1693209694314" xmlns:tns1="http://www.example.com/namespaces/tns/16932095564"
[INFO]
Tests for TestSuite TestSuites\CommutativeOperations.bwts
Tests run : 2   Success : 2   Failure : 0   Errors : 0

Results
Success : 2   Failure : 0   Errors : 0
[INFO]
[INFO] ## Running Test Suite TestSuite\CalculationSuite.bwts ##
[INFO]   Running Test for calculator.bwt
[INFO] Starting Tests in Module : tibco.bw.UT.sample.Calculator.module
[INFO] Uploading tests for Processes in Module : tibco.bw.UT.sample.Calculator.module
tibco.bw.UT.sample.Calculator.module.Calculator
[INFO] 2024-11-26T17:26:54,893 INFO [bwEngThread:In-Memory Process Worker-3] com.tibco.bw.palette.generalactivities.Log.tibco.bw.UT.sample.Calculator.m
[INFO] <tns:CalculatorOutput xmlns:tns="http://www.example.com/namespaces/tns/1693209694314" xmlns:tns1="http://www.example.com/namespaces/tns/16932095564"
[INFO] 2024-11-26T17:26:55,148 INFO [bwEngThread:In-Memory Process Worker-3] com.tibco.bw.palette.generalactivities.Log.tibco.bw.UT.sample.Calculator.m
[INFO]
Tests for TestSuite TestSuite\CalculationSuite.bwts
Tests run : 1   Success : 1   Failure : 0   Errors : 0

Results
Success : 1   Failure : 0   Errors : 0
```



# TIBCO Documentation and Support Services

---

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

## How to Access TIBCO Documentation

Documentation for TIBCO products is available on the [Product Documentation website](#), mainly in HTML and PDF formats.

The [Product Documentation website](#) is updated frequently and is more current than any other documentation included with the product.

## Product-Specific Documentation

The following documentation for this product is available on the [TIBCO BusinessWorks™ Container Edition](#) page:

- *TIBCO BusinessWorks™ Container Edition Release Notes*
- *TIBCO BusinessWorks™ Container Edition Installation*
- *TIBCO BusinessWorks™ Container Edition Application Development*
- *TIBCO BusinessWorks™ Container Edition Application Monitoring and Troubleshooting*
- *TIBCO BusinessWorks™ Container Edition Bindings and Palettes Reference*
- *TIBCO BusinessWorks™ Container Edition Concepts*
- *TIBCO BusinessWorks™ Container Edition Error Codes*
- *TIBCO BusinessWorks™ Container Edition Getting Started*
- *TIBCO BusinessWorks™ Container Edition Maven Plug-in*
- *TIBCO BusinessWorks™ Container Edition Migration*
- *TIBCO BusinessWorks™ Container Edition Performance Benchmarking and Tuning*
- *TIBCO BusinessWorks™ Container Edition REST Implementation*
- *TIBCO BusinessWorks™ Container Edition Refactoring Best Practices*

- *TIBCO BusinessWorks™ Container Edition Samples*

## How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our [product Support website](#).
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the [product Support website](#). If you do not have a username, you can request one by clicking **Register** on the website.

## How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the [TIBCO Ideas Portal](#). For a free registration, go to [TIBCO Community](#).

# Legal and Third-Party Notices

---

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of Cloud Software Group, Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, ActiveMatrix BusinessWorks, ActiveSpaces, Business Studio, TIBCO Business Studio, TIBCO Designer, TIBCO Enterprise Administrator, Enterprise Message Service, Rendezvous, and TIBCO Runtime Agent are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (<https://www.cloud.com/legal>) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: <https://scripts.sil.org/OFL>

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file for the availability of a specific version of Cloud SG software on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at <https://www.cloud.com/legal>.

Copyright © 2015-2024. Cloud Software Group, Inc. All Rights Reserved.