Deploy database applications against various environments

Introduction

This tutorial describes how to define a deployment process against various environments, including multiple tenant environments and multiple sysplex environments with the delivered plugins in IBM Urbancode Deploy. In this tutorial, the application to be deployed refers to database application running on DB2 for z/OS.

However, the method and technology described in this tutorial can also be used to deploy other types of applications against other database platforms, and the sample introduced in the tutorial can also be easily modified to be shared across different database platforms.

Scenario

The same application can be deployed to various environments for different purposes, like development or unit testing, functional testing, and so on. Some environments might reside on different LPARs and sysplex environments, while others might need to be deployed under different schemas on the same subsystem, like multiple tenant environments.

Other environments might have different requirements. For example, a unit testing environment might already have an old version of certain data objects (stored procedures, tables, etc.), so, the upgrade or alternation is needed. However, a functional regression testing environment might need a clean setup of the application from scratch. Another typical sample is that some functional testing environments might need the packages of the application to be recompiled and to be bound under different corresponding collection IDs.

The template process that is described in this tutorial addresses all of these situations with the same deployment flow.

This tutorial describes how to create a flexible deployment process that completes the following actions:

- Loads the artifacts from source control management solution

- Rolls out an application onto multiple schemas within and across multiple subsystems (possibly multiple sysplex)
- Can check the existence and version or signature of the existing data objects on the target environment
- Can modify or upgrade some of existing data objects that are on an old level to the latest in one target environment and/or create the data objects that don't exist on another environment
- Substitutes the symbols with the environment-specific values from the input of the plain-text property files
- Runs the related DB2 utilities and commands
- Supports RACF PassTicket to avoid storing passwords
- Enables users to drive the deployment process with the batch commands via REST APIs

The process uses existing plugins in IBM Urbancode Deploy. All of the artifacts used or files referenced in this tutorial are available in the template package that you can download.

Implement with IBM Urbancode Deploy

The sample template in the tutorial consists of:

- The components containing the artifacts
- The processes that prepare the artifacts and deploy the application
- The environments where the deployment will be run
- The application which holds the components and environments

The tutorial leads you through the following steps to create the entire deployment flow:

Step 1: create the components and import the artifacts with version control

Step 2: implement the process which prepare the DBRMs to be bound and the related BIND job based on the BIND cards

Step 3: build the main process that deploys the artifacts

Step 4: define the application containing the environments and components

Step 5: add target environments and corresponding environment property files

Step 6: kick off the entire deployment process

You can also download the template and import that into IBM Urbancode Deploy, and go through the template along with the steps in the tutorial.

In order to import the template, on the IBM Urbancode Deploy web UI Applications page, click **Import Applications**, and choose the template package.

All the steps described in the section "Create the solution" are through IBM Urbancode Deploy web UI.

Before you begin

Ensure that the following prerequisites are ready:

- IBM Urbancode Deploy server, agent and toolkit are installed correctly
- JDBC driver and RACF PassTicket jar files are available on the Unix Service System on your target subsystem. Consult your administrator for more information
- The following plugins are downloaded from IBM Urbancode Deploy Plugin website and installed if not yet
    - File Utils plugin
    - SQL-JDBC plugin
    - zOS Utility plugin

Create the solution

Step 1. Create the components and import the artifacts with version control

Step 1.a. On the Components page, click **Create Component**. The Create Component dialog opens.

Figure 1. Create Component "DeployDemoPrep"

Create Component

Name * DeployDemoPrep

Description

Teams

Template None

Component Type Standard

Version Source Configuration

Source Configuration Type

Import Versions Automatically

Copy to CodeStation

Default Version Type * Full

Use the system's default version import agent/tag.

Import new component versions using a single agent.

Import new component versions using any agent with the specified tag.

Cleanup Configuration

Inherit Cleanup Settings

Run Process after a Version is Created

Save Cancel

Type the name and description. Use default values for the other fields. This sample, uses "DeployDemoPrep" as the name, as shown in figure 1. This component is used to prepare the DBRMs and BIND jobs. The DBRMs as artifacts are imported through BUZTOOL supplied by IBM Urbancode Deploy.

Then click **Save**.

The new component "DeployDemoPrep" is added to the list on the Components page. Move mouse over the component "DeployDemoPrep", you will see a little icon right to the component name. Refer to the icon in the red box in figure 2. You can add a

component tag to the component by clicking that little icon, and "deployPrep" is used in the sample in figure 2.

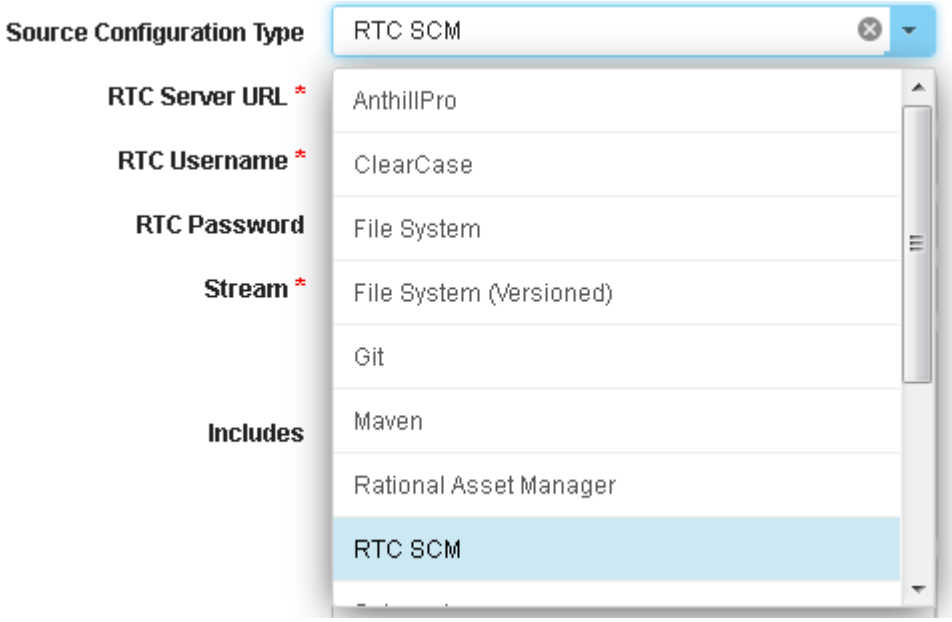
Figure 2. Add a tag

Next, create the other component which is the major one in this template.

Step 1.b. Click **Create Component** again, and configure the component properties.

Figure 3 shows the list of possible source where IBM Urbancode Deploy can use as the input source to import the artifacts.

Figure 3. RTC SCM as a sample

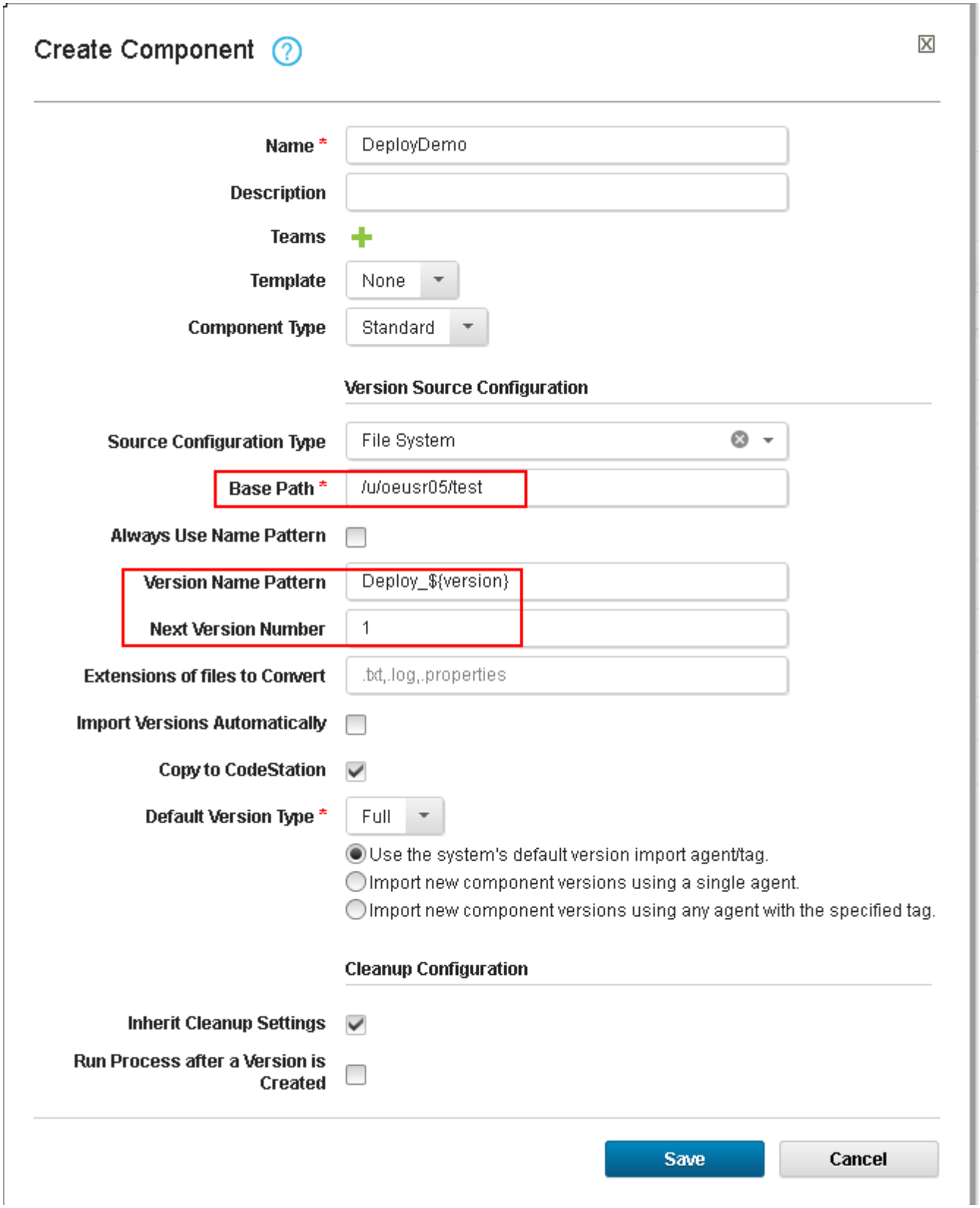


IBM Urbancode Deploy supports a few source control management solutions. You can select the one that is used in your shop if that is listed, and configure the related fields. You might need to consult with your administrator in your shop to get the required information.

The sample template uses "File System" as an example here, because the tutorial explains how to manage the versions for the imported artifacts when the versions are not coming along with the artifacts from the source control management solution.

Configure the component "DeployDemo" as shown in the following figure.

Figure 4. Create Component "DeployDemo" with File System



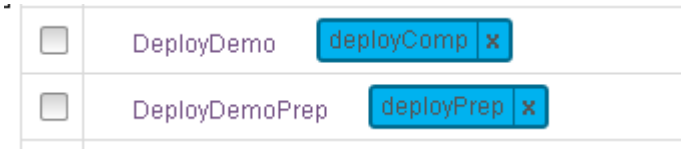
The “Base Path” is the locations where the artifacts are stored and imported. It is a Unix System Services directory. “/usr/oeusr05/test” is used in Figure 4. The “Version Name Pattern” identifies the version that is imported to IBM Urbancode Deploy. The “${version}” in the text field uses the incremental value which starts from the value defined in the text field of “Next Version Number” every time a new version of artifacts is imported into the component. In this sample, you get “Deploy_1” for the artifacts you imported the first time, and “Deploy_2” at the second time, and so on.

“Import Versions Automatically” allows Urbancode Deploy to poll the folder and to import the artifacts automatically when there is any change to the files in the folder. Leave it unchecked to control when to import the artifacts for this tutorial.

Click **Save**, and add a tag “deployComp” to the new component “DeployDemo”.

You now have 2 components created and tagged.

Figure 5. Component list



Step 2. Implement the process which prepare the DBRMs to be bound and the related BIND job based on the BIND cards

Step 2.a. Click the component “DeployDemoPrep”, On the Processes panel, and click **Create Process** to build the process.

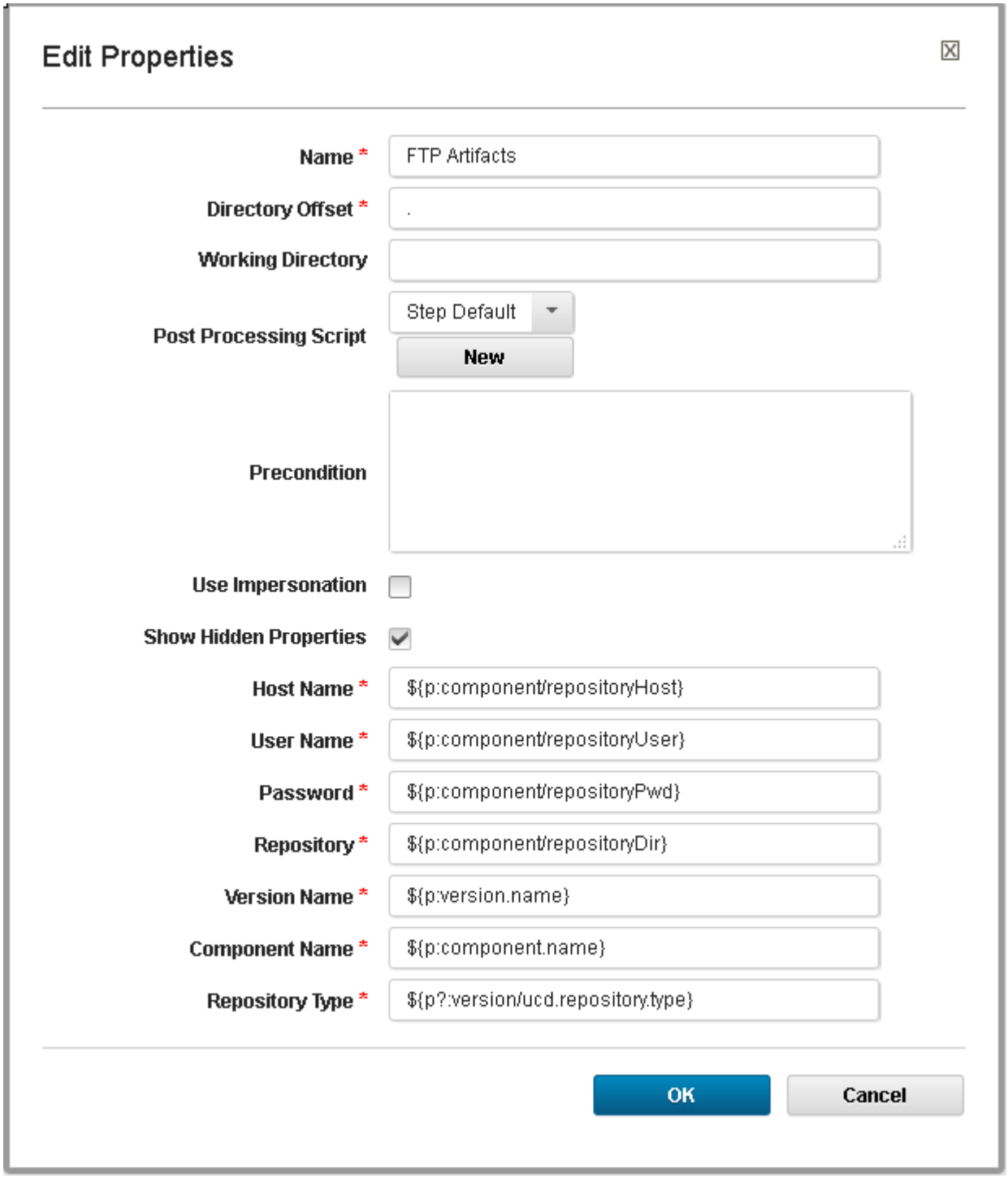
Type a process name in the popup dialog. The sample uses “RDProcess”.

Click the new process “RDProcess” in the list. You are now ready to define the process.

You use the zOS Utility plugin to transfer the DBRM files from the component to the target datasets on the target environment in the sample.

Step 2.b. Drag “FTP Artifacts” from the “Step Palette” in to the working area on the right side, and define the properties of the step as shown in the following figure.

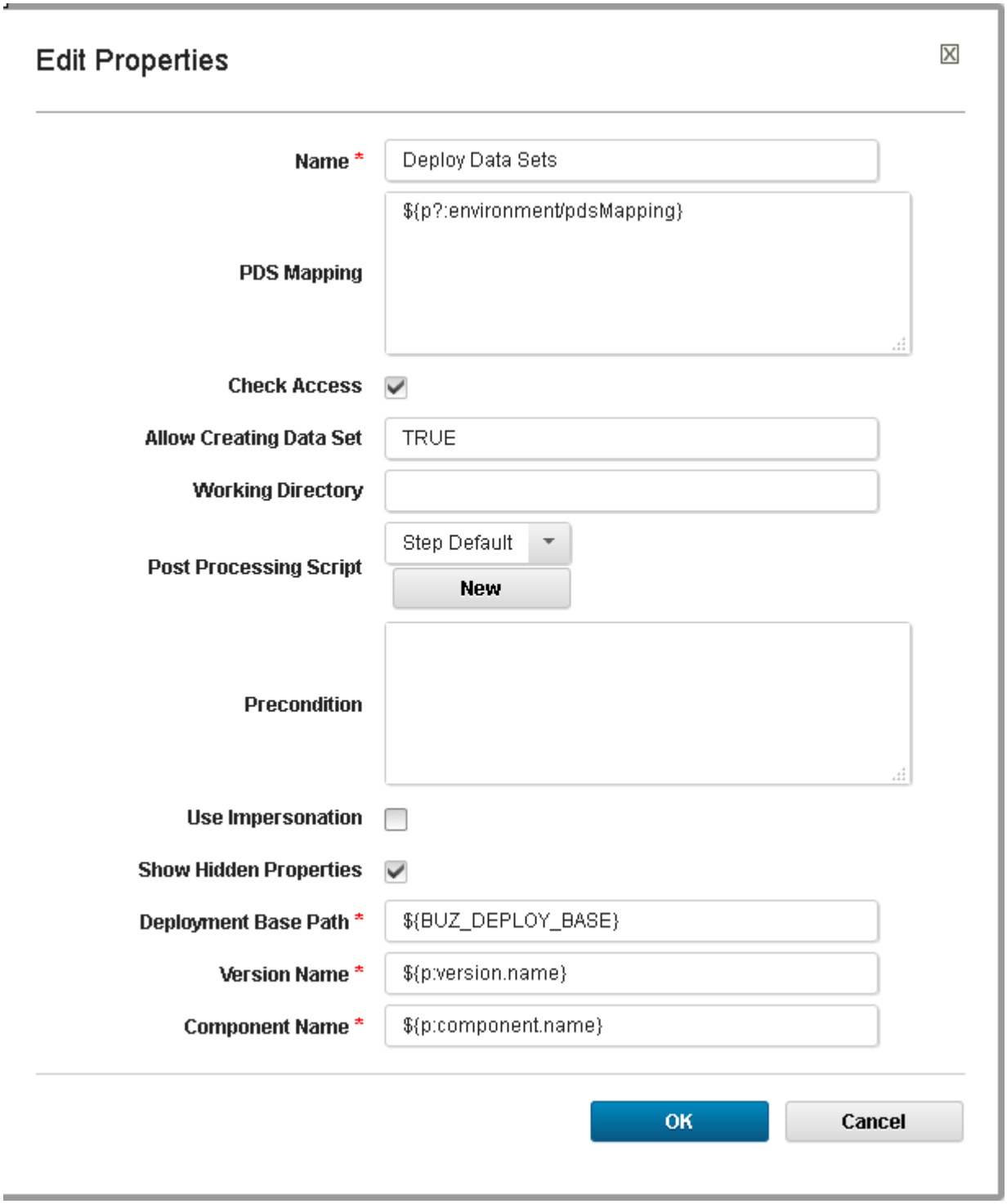
Figure 6. Edit Properties of FTP Artifacts



The sample uses "FTP Artifacts" as the step name. Check "Show Hidden Properties" at the bottom, and you will see additional properties to define the related information of the repository which is used to keep the DBRMs. You can use the names of component properties as the values for the inputs here, and define the real values in the component properties later.

Step 2.c. Add another step to deploy the transferred artifacts to the target datasets. Drag "Deploy Data Sets" to the working area, and define the properties as shown in the following figure.

Figure 7. Edit Properties of Deploy Data Sets



You use an environment property for the value of “PDS Mapping” field, because the target dataset name may be different for each individual environment. You define the environment property later. The field of “Allow Creating Data Set” is set to TRUE, this option enables IBM Urbancode Deploy to create the target PDS dataset with the default settings if the dataset is not available. Set the field to FALSE if your shop doesn’t want this behavior. However, then you need to ensure that the target PDS dataset is created on the target environment before you kick off the deployment process.

Step 2.d. Add another step to generate the BIND card for the DBRMs. Drag “Generate Artifact Information” into the working area, and define the properties as shown in the following figure:

Figure 8. Edit Properties of Generate Artifact Information

Edit Properties

Name * GenBndCard

For Each * Member

Source Data Set Name Filter

Target Data Set Name Filter

Member Name Filter

Deploy Type Filter DBRM

Custom Properties Filter

Template *

```
  BIND PACKAGE(@COLLID@)
MEMBER(${member}) +
      ACTION(@ACTION@)
ISOLATION(@ISOLATION@) +
      RELEASE(COMMIT) ENCODING(EBCDIC)  +
      LIBRARY('${dataset}')
```

Working Directory

Post Processing Script Step Default

New

Precondition

Use Impersonation ☐

Show Hidden Properties ☑

Deployment Base Path ${BUZ_DEPLOY_BASE}

Version Name ${p:version.name}

Component Name ${p:component.name}

OK Cancel

This step iterates all of the PDS dataset members and generates the text, BIND card, in this sample with the input of the template field. In the “Template” field of Figure 8, ${member} is the symbol representing each PDS dataset member; the string quoted with the character ‘@’, like @COLLID@, is the symbol that is substituted with the environment-specific value later during main deployment process. The character ‘@’ is the default quotation mark for symbolic substitution in IBM Urbancode Deploy. You can use other special charaters if ‘@’ is not a good choice for your shop.

In this sample, you put the BIND card template directly in the step. However, you can also use “Read Property File” in the “Step Palette” to get the BIND card from a plain text file if your shop has the BIND card stored in a file. Then you refer to the property used to read in the content of the file in the “Template” field above.

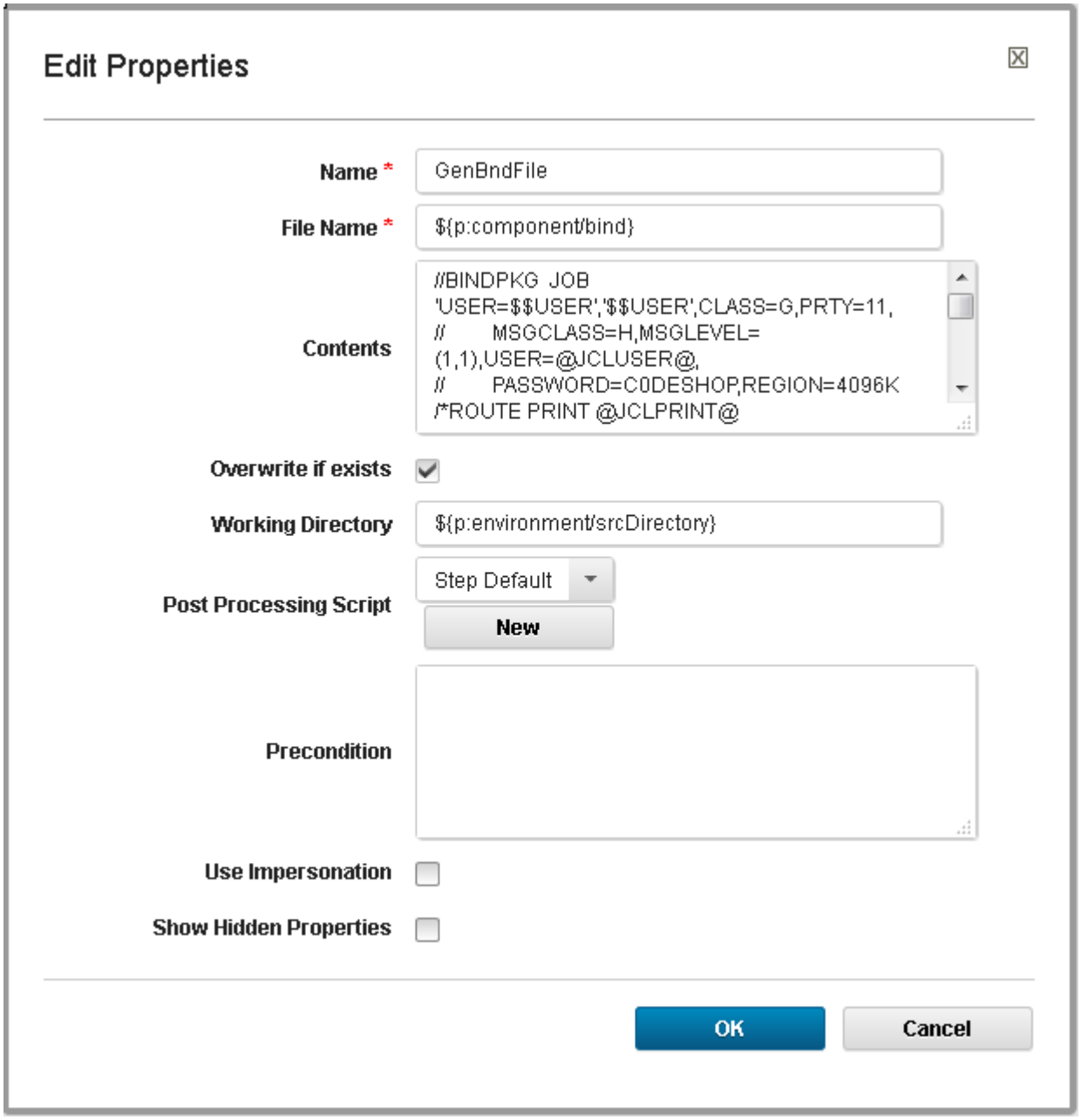
The symbols are very useful for deploying the application to different environments, especially in multiple tenant environments, where you might want to create the data objects under different schemas and to bind the packages with different qualifiers under different collection ids for different end users, so that different end users can share the same subsystem without interference.

The JCLs and SQL files shipped in the template package show a sample, and the template process is also ready to deploy to multiple tennant environments as a sample.

You can use symbols for anything, like syntax options, parameters and so on, whichever shows up in your artifacts, as long as you have the real value to replace that later.

Step 2.e. Add another step to generate the BIND job for all the DBRMs. Drag “Create File” under “FileUtils” plugin into the working area, and define the properties as shown in the following figure:

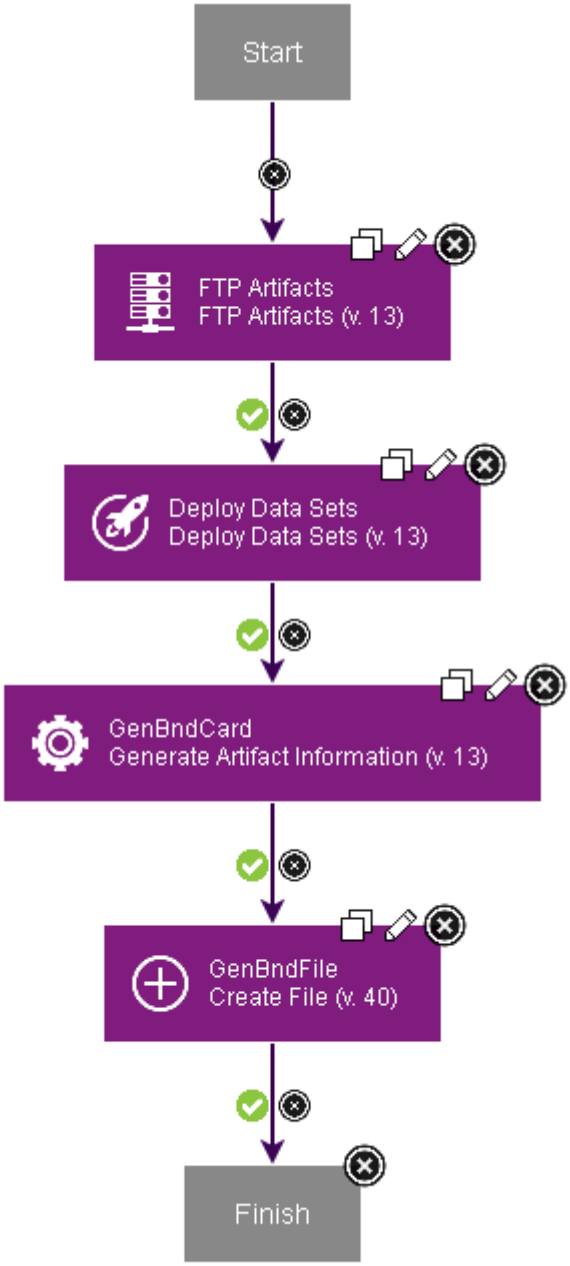
Figure 9. Edit Properties of Create File



Use a component property for the name of the BIND job because this can be same for all environments. You define this property later. In the "Contents" field, put a template JOB header with symbols which will be substituted in the main deployment process, and use the BIND cards, ${GenBndCard/tex}, generated from the previous step. You also defined a directory under which the BIND job will be put in the "Working Directory" field. This will be the same directory for other artifacts in the main deployment process.

You should see the entire process defined as shown in the following field.

Figure 10. The process of DeployDemoPrep

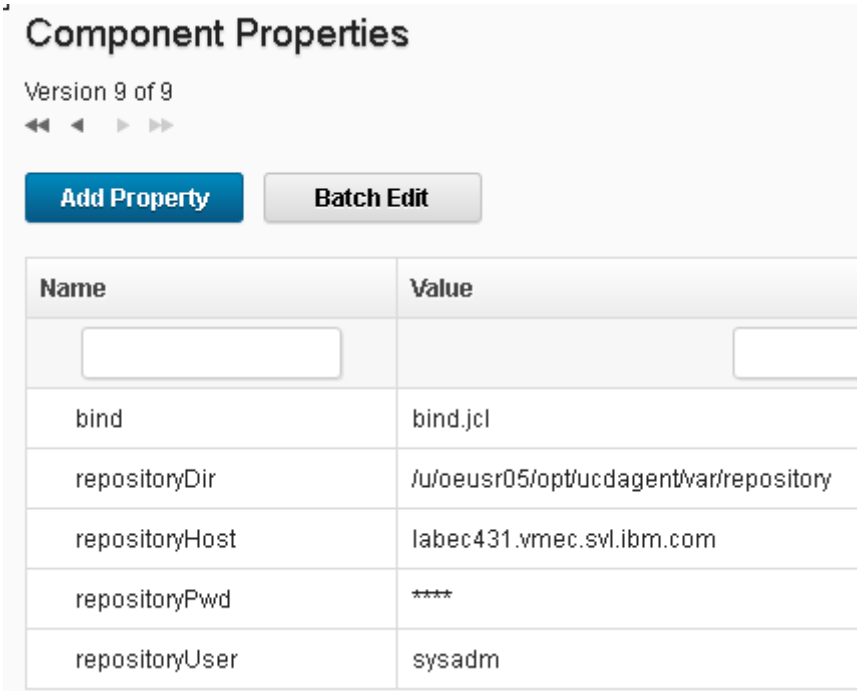


Step 2.f. Define the required properties for the component "DeployDemoPrep".

Click the component "DeployDemoPrep", On the Configuration panel, and switch to "Component Properties" tab.

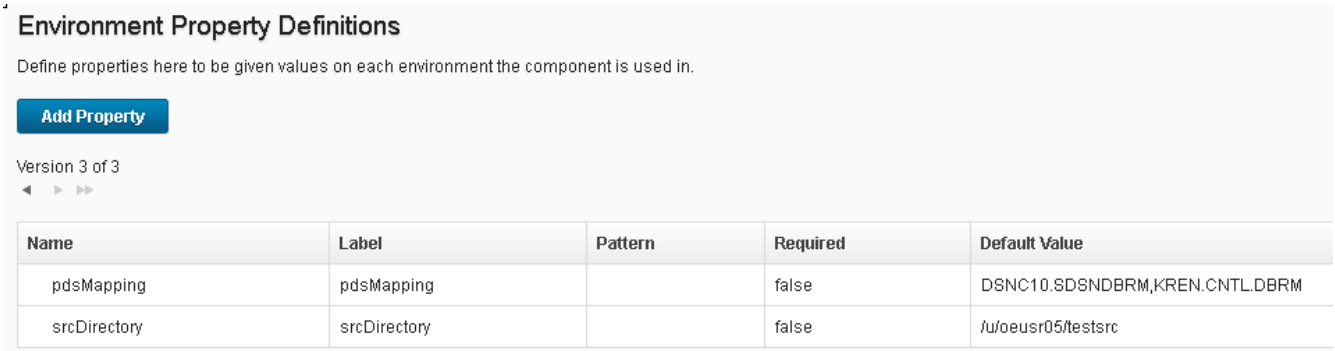
Click **Add Property** to add the following properties as shown in the following figure.

Figure 11. Component properties of "DeployDemoPrep"



Switch to "Environment Property Definitions", and add properties as shown in the following figure.. This is an optional step.

Figure 12. Add environment properties



Now you probably notice an interesting thing.

AS mentioned earlier, these 2 symbols are at the environment level and can have different values for each target environment in the deployment. Why define them in the component? You can trust that this is not a mistake. This step actually defines the default values for these environment properties at the component level. In other words, if you don't provide the real values for a certain target environment, then the deployment process uses the default values that you provide here. If you define the real values for each target environment as in step 5, the values given in the step 5 overwrite the default values defined here during the deployment. This gives you one

way to share some common settings among some environments. Of course, you don't have to do this if this is not the case you want.

Step 2.g. Import the DBRMs into the component "DeployDemoPrep" with BUZTOOL and the sample job, SBUZSAMP(BUZRJCL), supplied by IBM Urbancode Deploy. You can customize and run the sample job to get the DBRMs from the source dataset into the component in IBM Urbancode Deploy. The packageManifest.xml in the template shows how to compile the list of the required DBRMs as the input list to be imported. You could find more detail of BUZTOOL on http://www-01.ibm.com/support/knowledgecenter/api/content/SS4GSP_6.1.3/com.ibm.udeploy.doc/topics/zos_runtools.html?locale=en.

After it is done, you see the imported artifacts listed as shown in the following figure:

Figure 13. Artifact list of "DeployDemoPrep"

The component "DeployDemoPrep" is ready by now. If you need to deal with load modules, the process is similar and simpler, and you can skip the BIND card and the step to generate the BIND job.

Step 3: build the main process that deploys the artifacts

Step 3.a. Click the component "DeployDemo", On the Processes panel, and click **Create Process** to build the process. The sample uses "DeployObjects" as the name.

Step 3.b. Add a step to transfer the artifacts to the target environment. Drag "Download Artifacts" under "Repository->Artifact->IBM Urbancode Deploy" into the working area, and define the properties as shown in the following figure.

Figure 14. Edit properties of "Download Artifacts"

Edit Properties

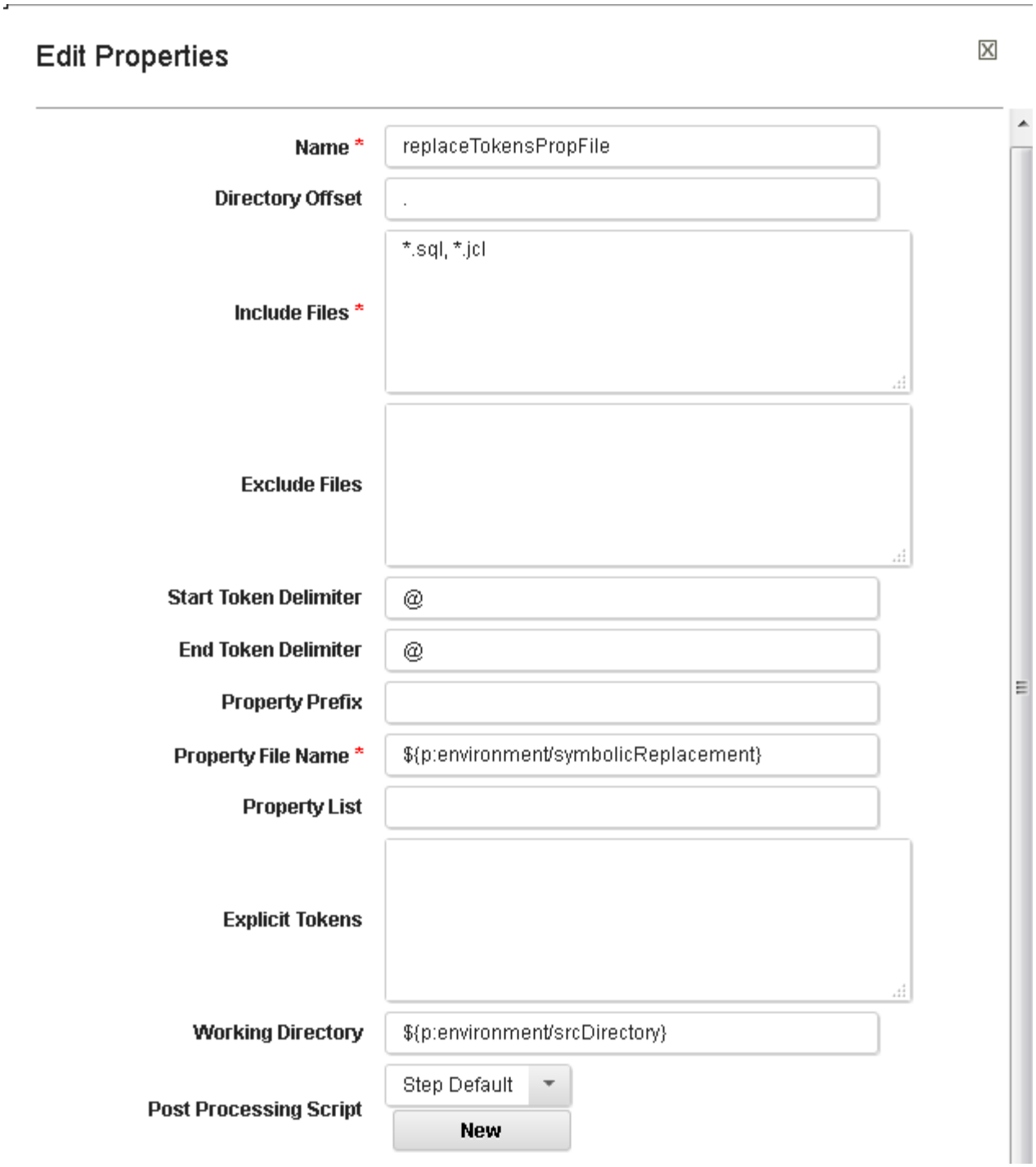
Name *	Download Artifacts
Directory Offset *	.
Artifact Directory Offset	
Includes *	**/*
Excludes	
Sync Mode	none
Full Verification	☑
Set File Execute Bits	☐
Verify File Integrity	☐
Charset	
Working Directory	${p:environment/srcDirectory}
Post Processing Script	Step Default New
Precondition	
Use Impersonation	☐
Show Hidden Properties	☐

OK Cancel

Use the same working directory as the process of component "DeployDemoPrep" to put all the deployable artifacts under the same folder on the target environment.

Step 3.c. Substitute the symbols with the environment specific values. Drag "Replace Tokens" under "FileUtils" into the working area, and define the properties as shown in the following figure.

Figure 15. Edit properties of "Replace Tokens"



Substitute all the symbols in both SQL and JCL files. Use "*.sql, *.jcl" in the "Include Files" field. "Start Token Delimiter" and "End Token Delimiter" are the fields to define your own token delimiter which works with your shop, the character '@' is the default. In this sample, use the text property files to feed the real environment values into the symbols in all the files which contain the symbols. The working diretory is pointing to the same folder where that contains all the artifacts on the target environment.

Step 3.d. Add a step to check the existence and signature of the data objects, like database, tables, etc. Drag "Execute SQL Scripts" under "SQL-JDBC" plugin under "Database" into the working area. Define the properties as shown in the following figure:

Figure 16. Edit properties of "checkDBObjects"

Edit Properties

Name *	checkDBObjects
Database JDBC Driver Name *	${p:resource/jdbcDriver}
Driver Jar *	${p:resource/jdbcJAR}
Connection String *	${p:resource/connectionURL}
User *	${p:resource/connectionUser}
Password	••••
Password Script	
Files	${p:component/checkObj}
Include Files	**/*.sql
Exclude Files	
SQL Statement Delimiter *	${p:component/delimiter}
Autocommit	☐
Print Result Sets	☑
Error Handling *	Abort
Working Directory	${p:environment/srcDirectory}
Post Processing Script	checkSQLCode New Edit

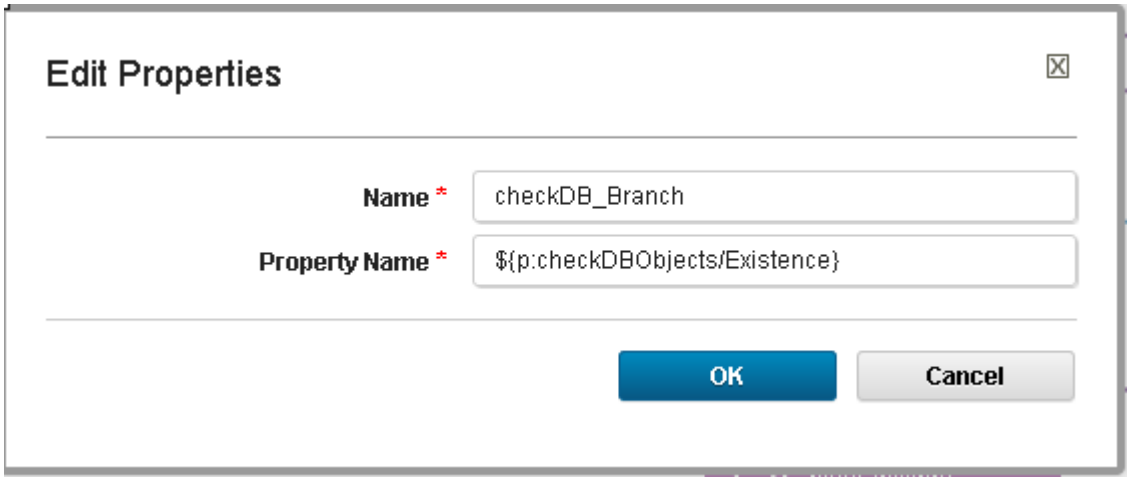
Use the resource properties for the required fields of JDBC driver which the plugin is using, because this information is at resource level. The same resource can be used in different environments, like multiple tennant environments, so you don't have to specify the same information on each environment that shares the same resource. Note, the password for JDBC connection can also be from a property. SQL-JDBC plugin doesn't support RACF PassTicket, so you must provide the password for the JDBC connection in this step.

Run a SQL script to inquiry the catalog to do the check, the SQL file is defined in one component property. You can also define the SQL statement delimiter with any special character that you prefer for your shop, and another component property holds the value. Again, set the working directory to the same folder.

A new post processing script "checkSQLCode" is created in this step. Refer to the source code in the template package. The post processing script generates a property, "Existence", with different values which are returned from the SQL checking the catalog. The values of the property "Existence" are used to decide which logic path to go in the following deployment process.

Step 3.e. Add the switch logic. Drag "Switch" under "Utility Steps" into the working area, and define the properties as shown in the following figure.

Figure 17. Edit properties of "checkDB_Branch"

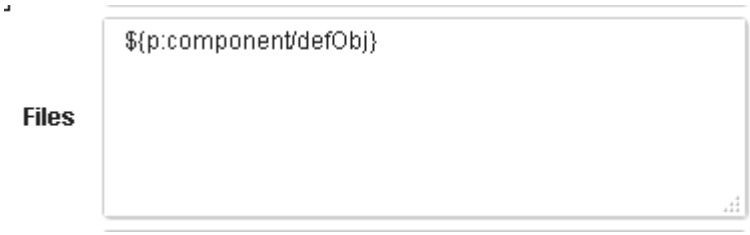


This step checks the value of the property generated in the post processing script of the previous step and goes to the different branches correspondingly. In this sample, the deployment process will skip the creation of the data objects (database, tables, indexes, views, udf and triggers) as well as the load of the data (note, INSERT statements are used to load the data in this sample. If the amount of the data is huge, you can use LOAD utility, refer to Step 3.g on how to run the utility) if the

target environment already had the required version of the objects defined; otherwise, the process will run the DDL to create the objects in the following step 3.f.

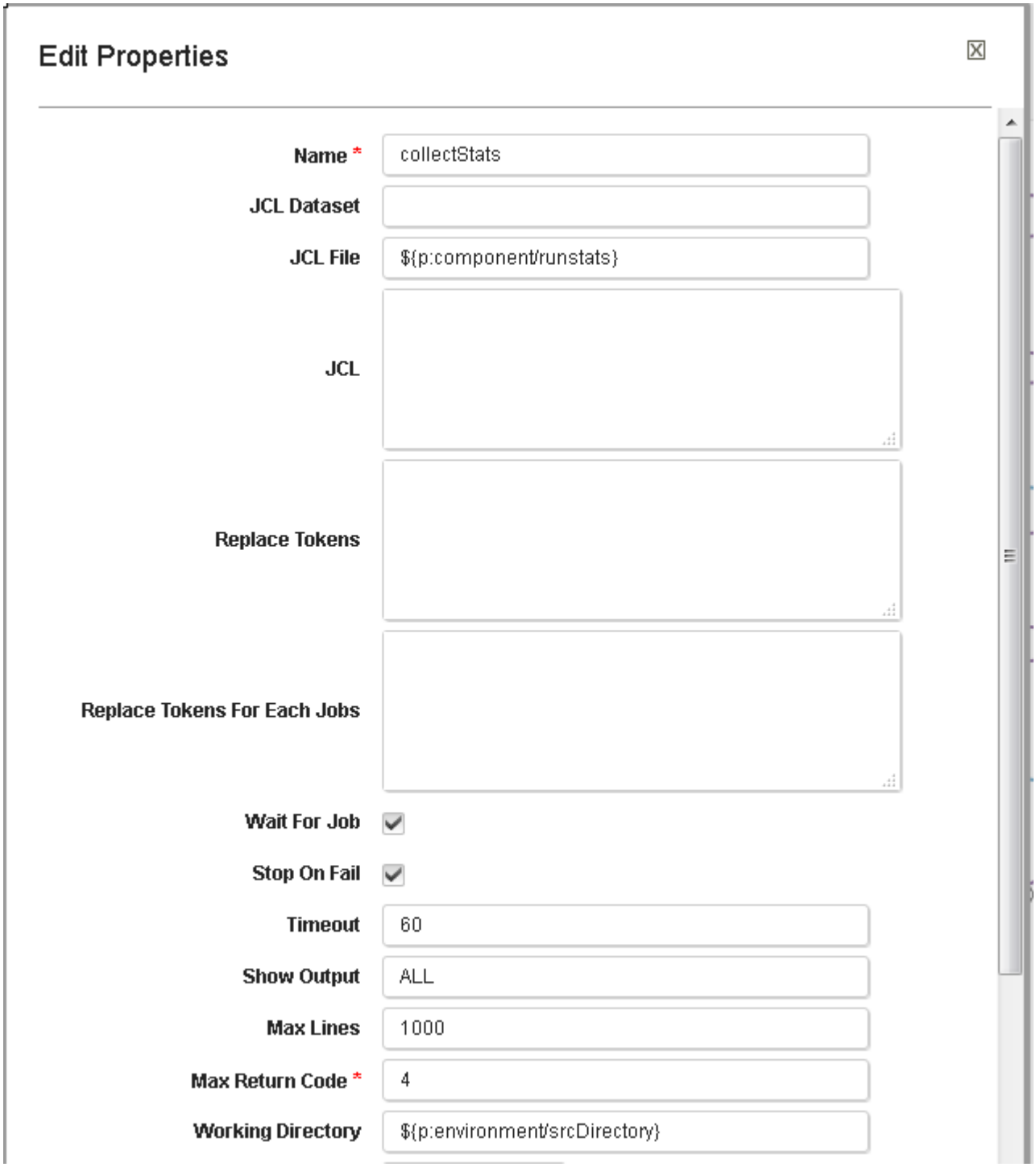
Step 3.f. Create the data objects if the data objects don't exist. Drag "Execute SQL Scripts" into the working area and define the SQL file as shown in the following figure.. Other properties are same as Step 3.d.

Figure 18. Edit properties of "defObj"



Step 3.g. Collect the statistics with RUNSTATS utility after step 3.e or step 3.f. to ensure that the statistics are up to date before you bind the packages later. Drag "Submit Job" under "zOS Utility" into the working area and define the properties as shown in the following figure.

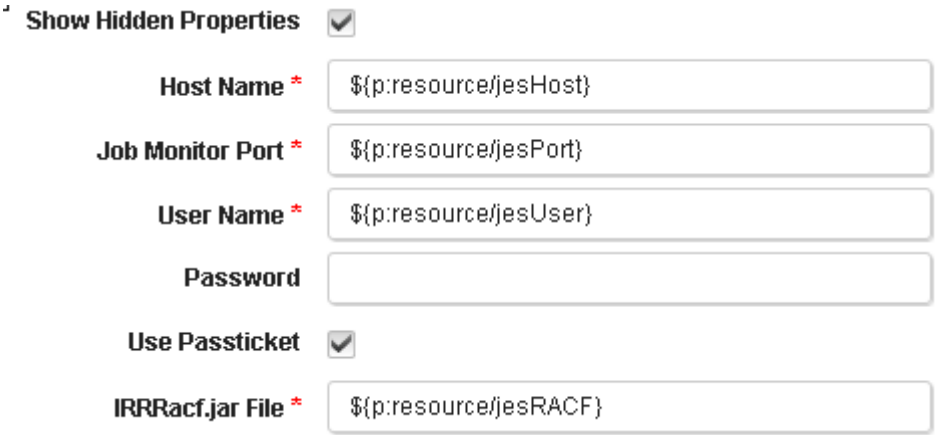
Figure 19. Edit properties of "collectStats"



This step uses JCL to run RUNSTATS utility. The JCL file is one of artifacts defined in one component property, while the actual JCL file contains RUNSTATS statements with symbols that are substituted with the environment specific values in the step 3.c during the deployment, so don't define anything in "Replace Tokens" field here. The step uses the default value 4 for the "Max Return Code" to decide whether the job fails or not, you can change the value based on your needs. Set the same working directory where you put the artifacts.

Check “Show Hidden Properties”, and set the fields as the following.

Figure 20. Hidden properties of “collectStats”



The “Submit Job” step supports RACF PassTicket, so you don’t have to store the password in the process. The real values of other fields are defined at the resource level, same as JDBC properties.

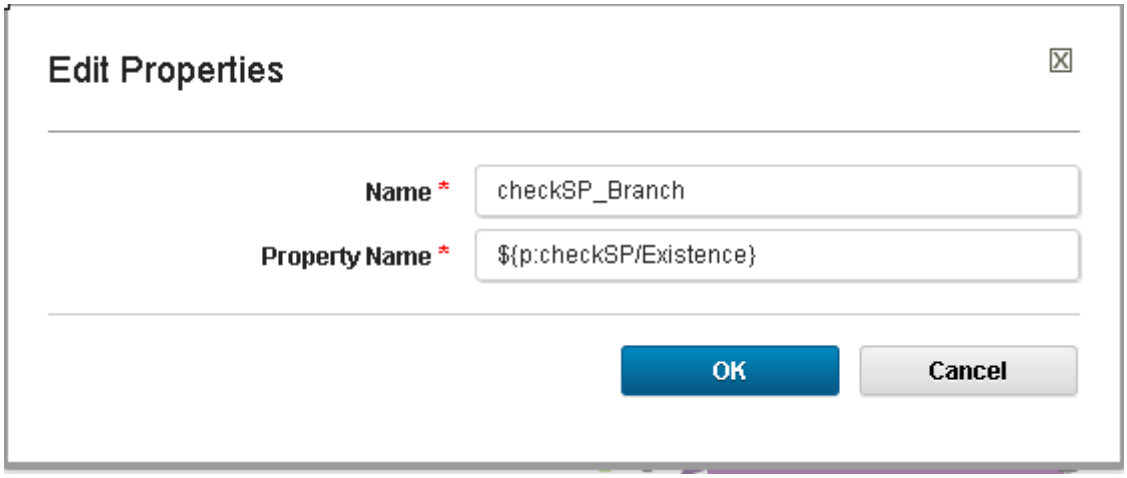
Refer to https://developer.ibm.com/urbancode/plugindoc/ibmucd/zos-utility-plug/1-2/ for the detail of RACF PassTicket support in the Submit Job of zOS utility plugin. You need to ensure that the RACF PassTicket is configured on the target subsystem successfully for Job Monitor which is used by the plugin here, before you run your deployment process at the last step.

Step 3.h. Add the check for routines that must be handled differently on different target environments. Drag “Execute SQL Scripts” into the working area, similarly as Step3.e. Define the SQL file to do the check against the catalog. Reuse the same post processing script as well.

Figure 21. Edit properties of “checkSP”

Step 3.i. Add the switch step like step 3.f, but set the property to point to the step defined in the step 3.h.

Figure 22. Edit properties of “checkSP_Branch”

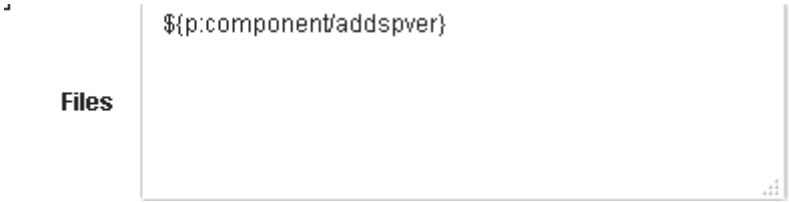


Step 3.j. Add the step to creat the routine if it doesn't exist on the target environment. Drag the "Execute SQL Scripts" into the working area, and define the SQL file as shown in the following figure..

Figure 23. Edit properties of "create stored procedure"

Step 3.k. Add the step to alter/replace the routine if the existing one is on the old version on the target environment. Drag "Execute SQL Scripts" into the working area, and define the SQL file as shown in the following figure..

Figure 24. Edit properties of "alter stored procedure"



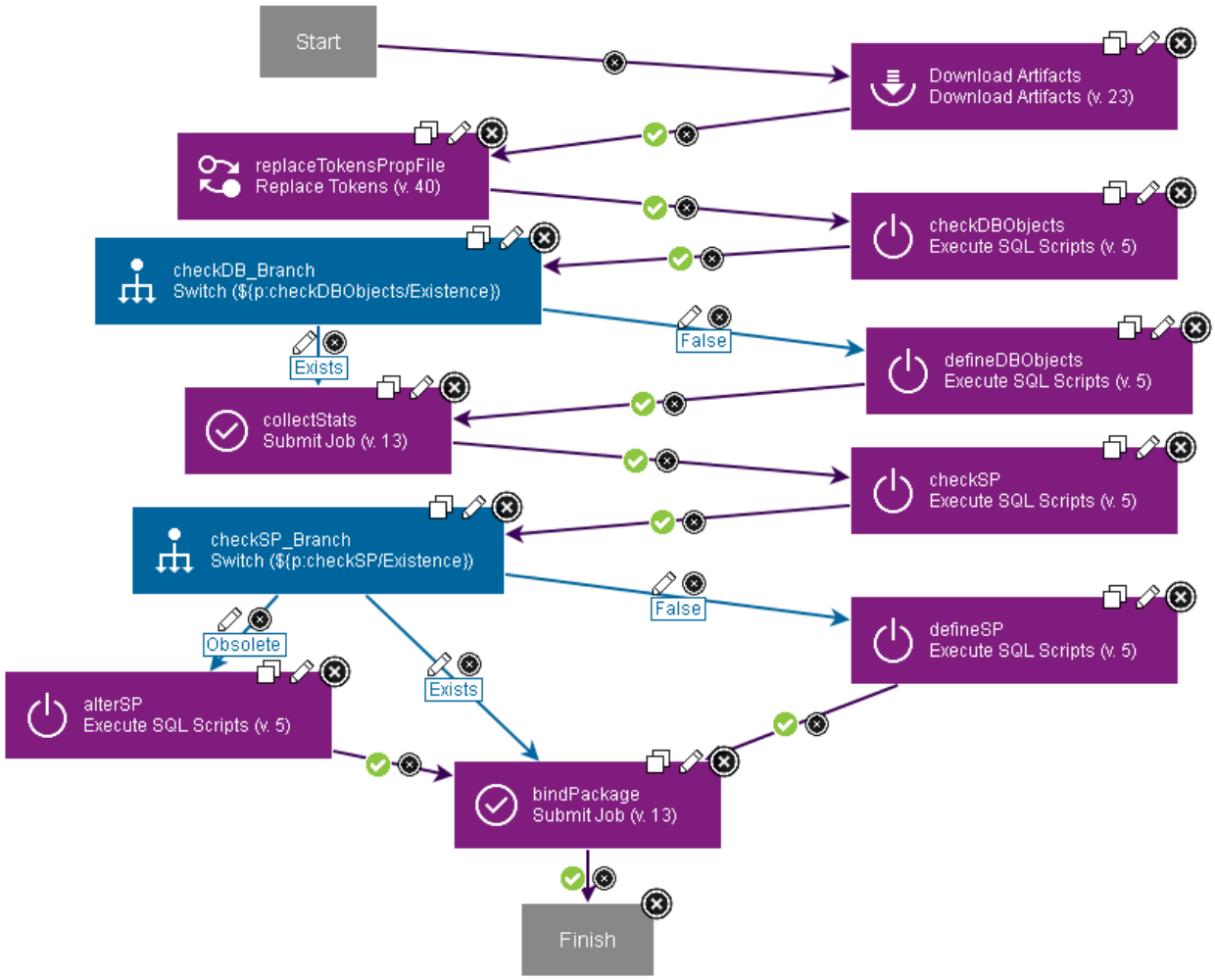
Step 3.l. Add the BIND step to bind the required packages using the BIND job which is generated in the "DeployDemoPrep" component. Drag "Submit Job" into the working area, and define the properties as shown in the following figure..

Figure 25. Edit properties of "bindPackage"

This step is similar as the step 3.g. Set the bind job in one component property and use that in "JCL File" field. All other fields are set same as the Step 3.g. Don't forget the hidden properties.

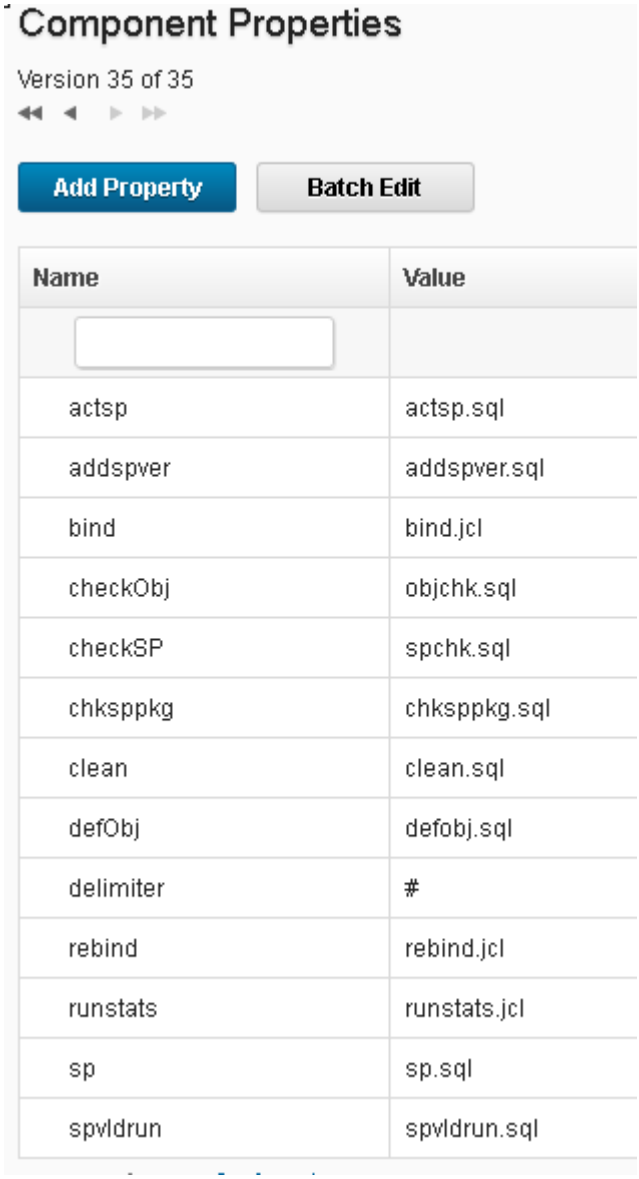
Step 3.m. Verify the final process diagram. You should get the process like the following one by now.

Figure 26. The process of "DeployDemo"



Step 3.n. Add the required component properties. Click "DeployDemo" on the component list, and go to "Configuration" panel. Switch to "Component Properties", and add the properties as shown in the following figure.

Figure 27. Add property of "DeployDemo"



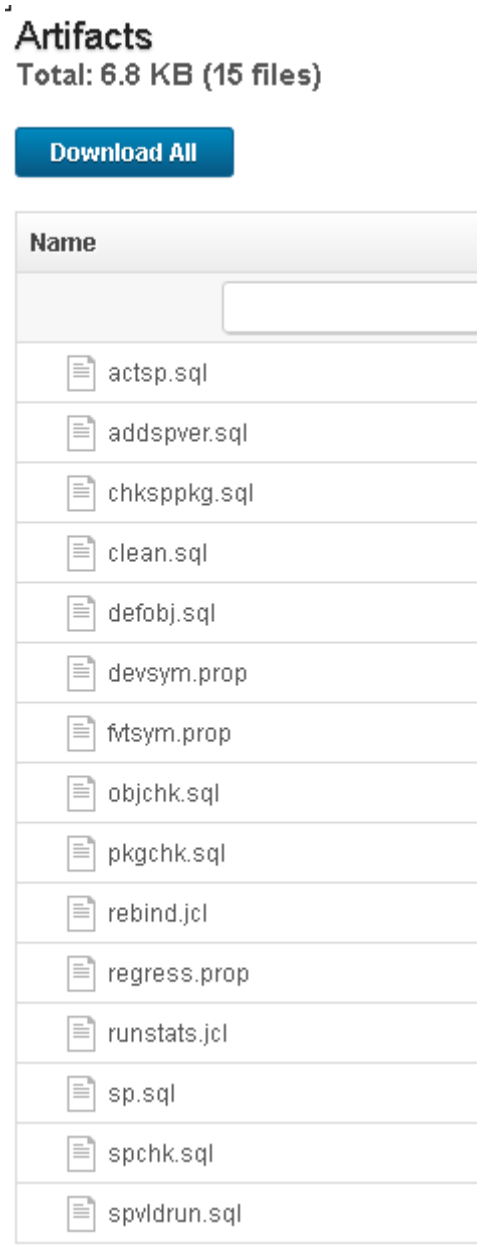
Step 3.o. Import the artifacts into the component. First ensure all the artifacts are already stored under the directory from Step 1.b Figure 4.

Swtich to "Versions" panel, and click **Import New Versions**. You will see all the artifacts imported into the component as the generated version which you will use to run the deployment.

All the artifacts used in this tutorial are included in the template package, and all are already imported into the component when you import the template.

You will see the imported artifacts as shown in the following figure. after it is done.

Figure 28. Artifact list of "DeployDemo"



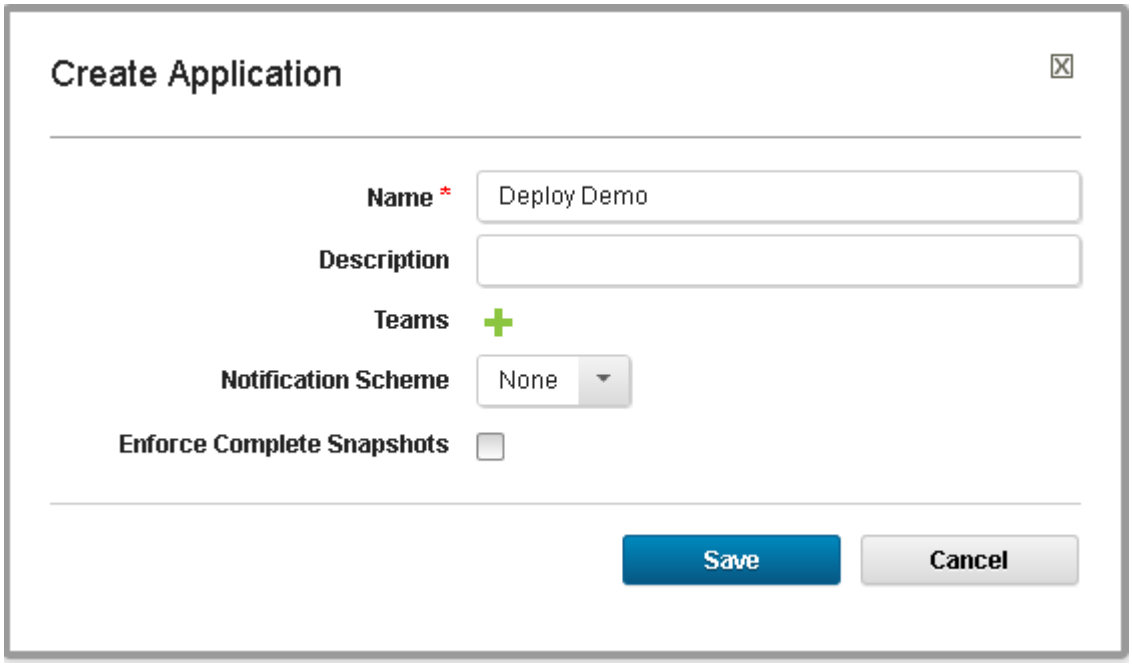
The "DeployDemo" component is now ready.

Step 4: define the application containing the environments and components

An application must be created to hold the components and corresponding target environments in IBM Urbancode Deploy.

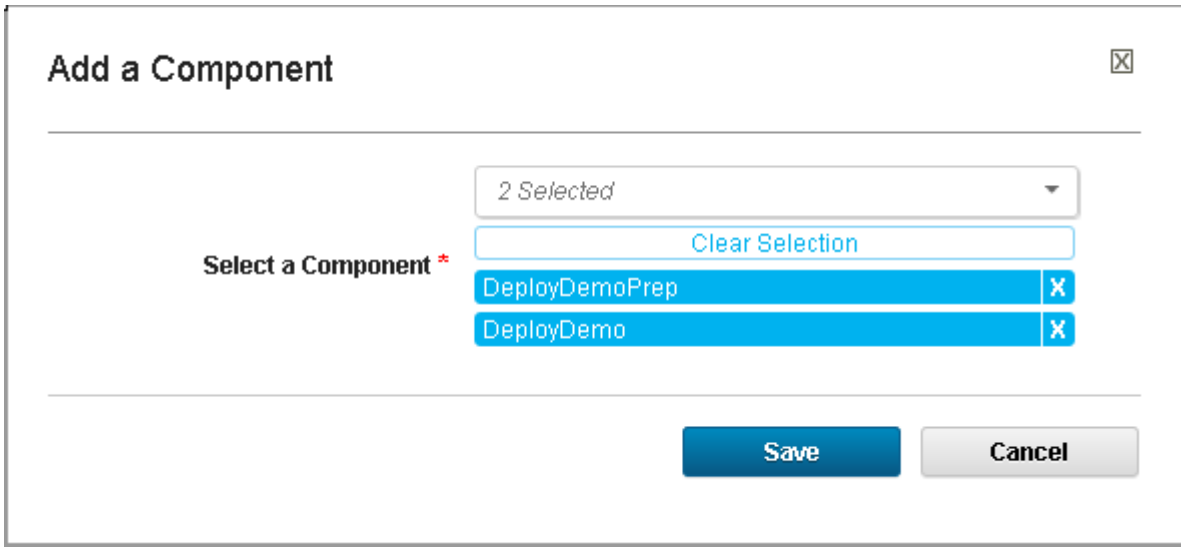
Step 4.a. Create an application. On the Applications page, click **Create Application**. The sample uses "Deploy Demo".

Figure 29. Create application



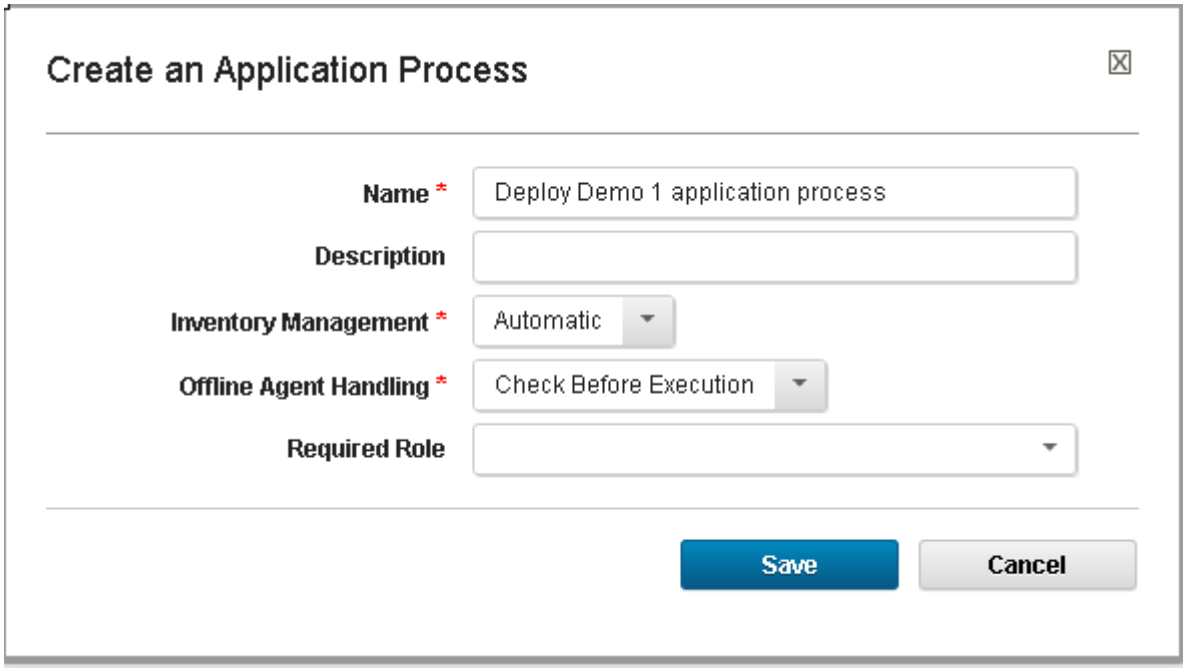
Step 4.b. Add components into the application. On the Components panel, click **Add Component**, and choose "DeployDemoPrep" and "DeployDemo" from the dropdown list.

Figure 30. Add components



Step 4.c. Create an installation process for the application. On the Processes panel, and click **Create Process**. The sample uses "Deploy Demo 1 application process".

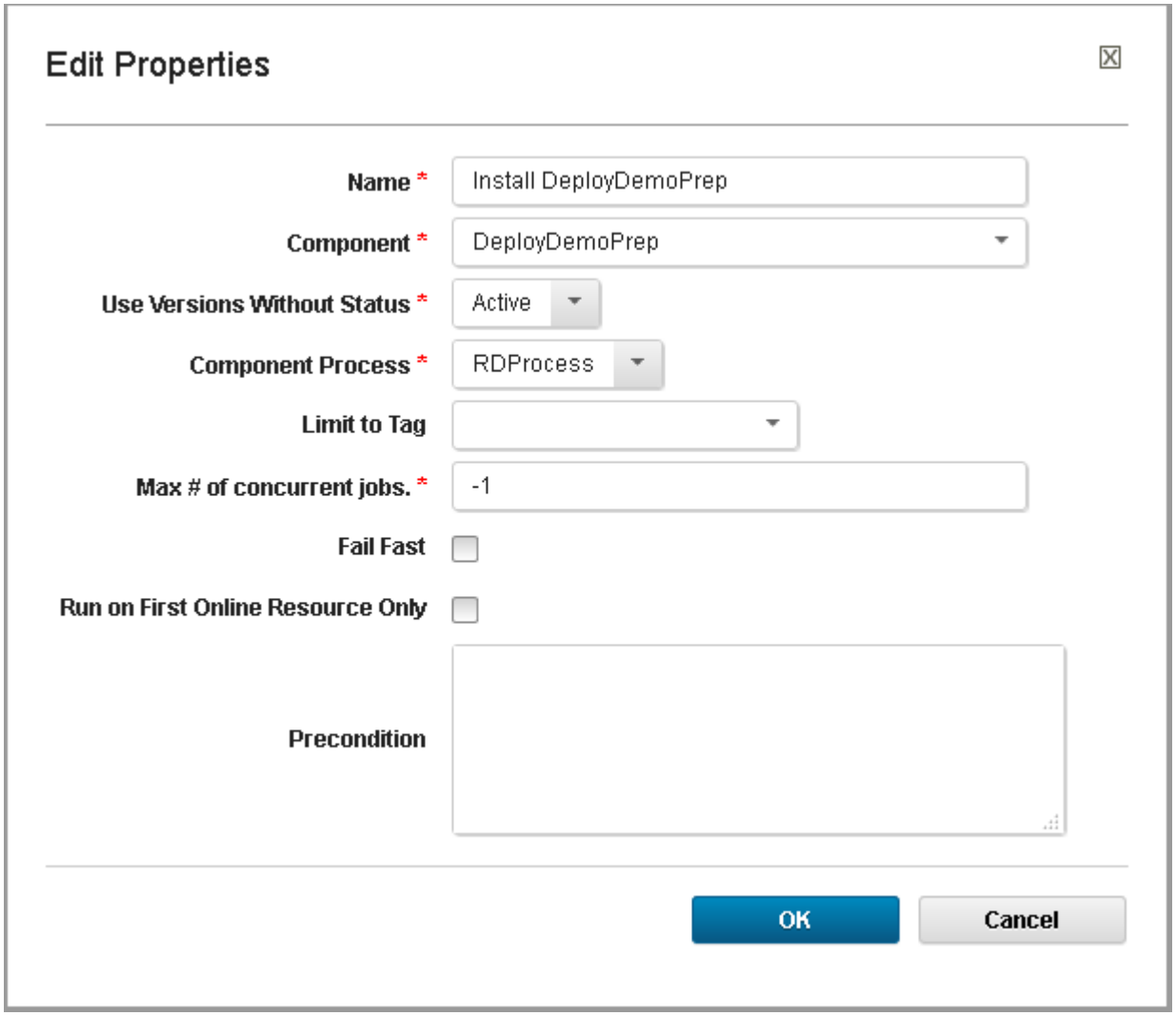
Figure 31. Create an application process



You will be led to the design view of the process.

Drag "Install Component…" into the working area, and define the properties as shown in the following figure.

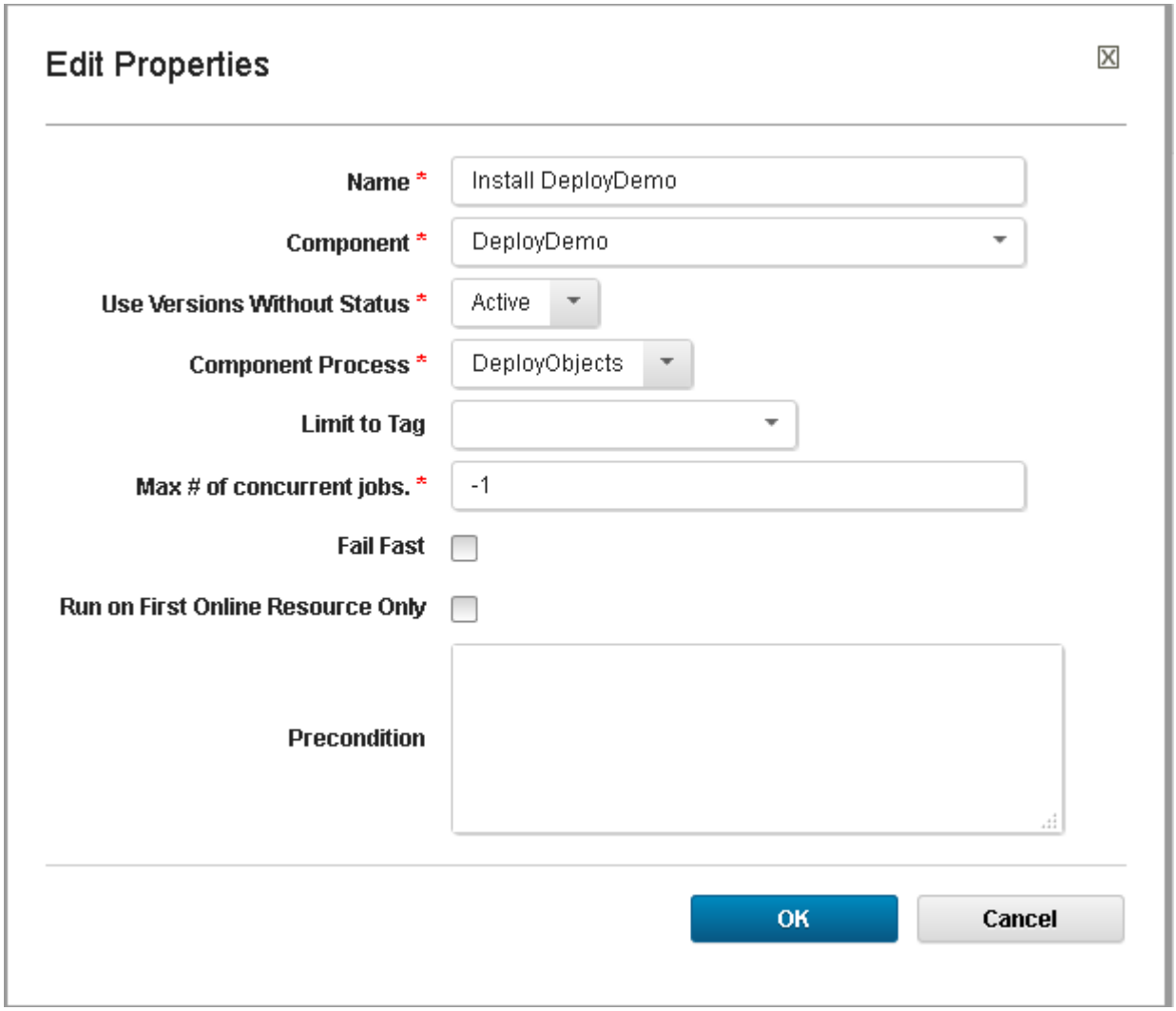
Figure 32. Edit properties of "Install DeployDemoPrep"



You must install the component "DeployDemoPrep" first to get the DBRMs and BIND job ready.

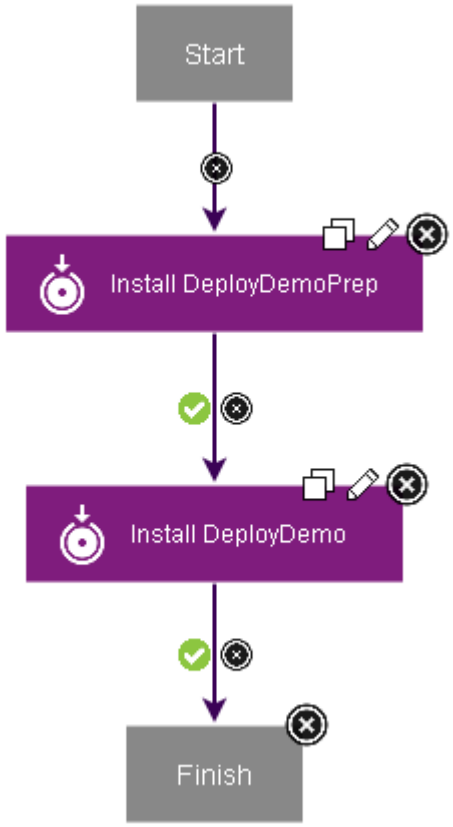
Drag "Install Component…" into the working area again to add the other component, and define the properties as shown in the following figure.

Figure 33. Edit properties of "Install DeployDemo"



You should have a process like below by now.

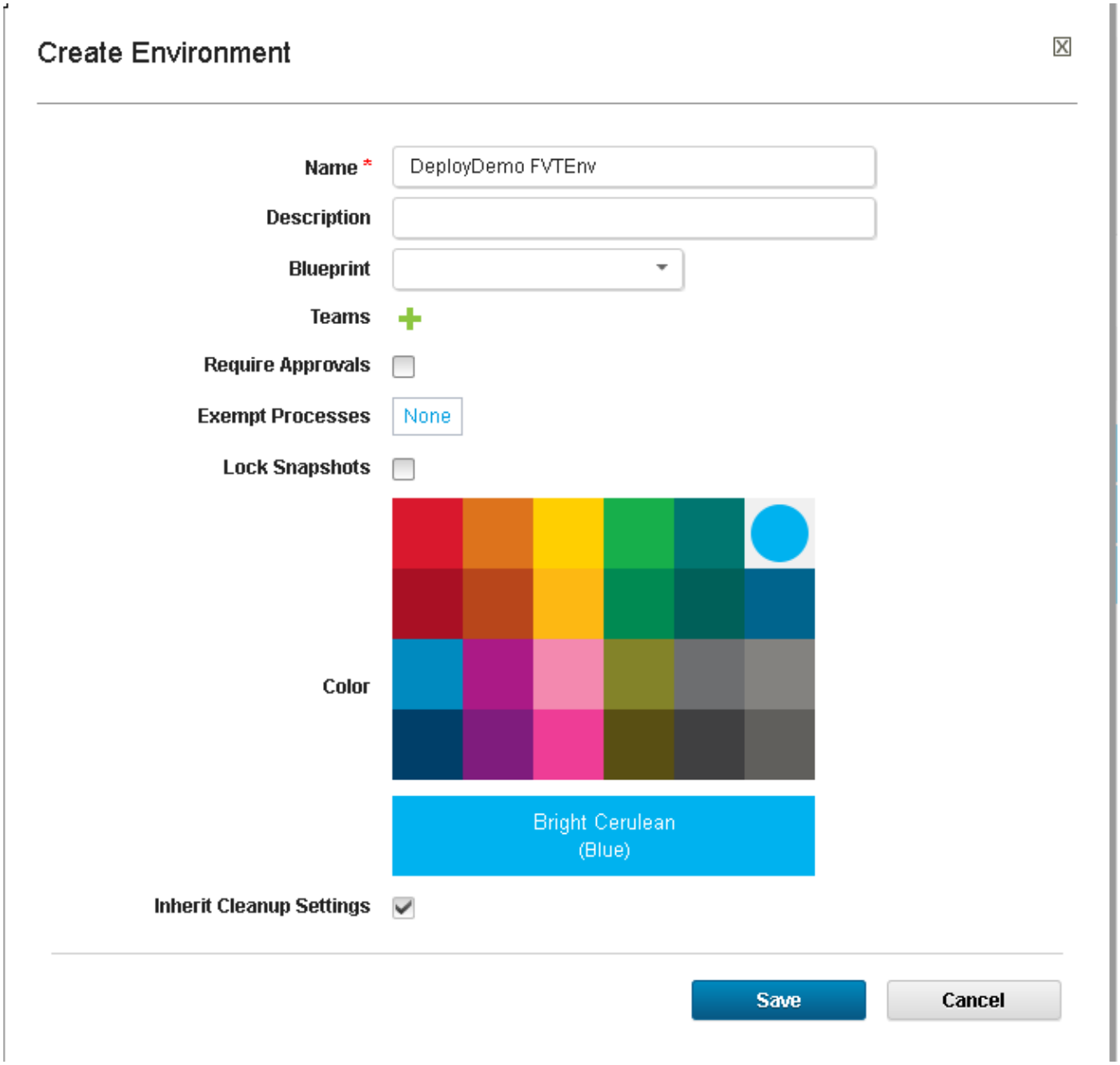
Figure 34. The process of the application



Step 5: add target environments and corresponding environment property files

Step 5.a. Add target environments into the application. In the Environments panel, click **Create Environment**, and define your environment the following figure:

Figure 35. Create environment



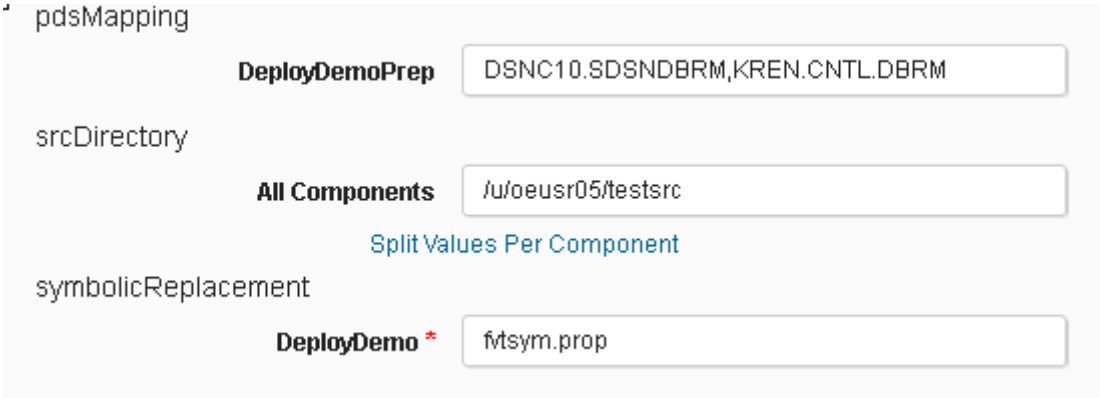
THe environment page opnes. Click **Add Base Resources**, and choose the target resource which has the IBM Urbancode Deploy agent installed and started. Then, click **Actions** to the right of the base resource, and choose "Add Component" to add both components.

You can add all of the environments by repeating the same process for each.

Step 5.b. Add the required properties to the target environments and underlying resources from the earlier steps.

Click the new added target environment, go to the Configuration panel, and switch to "Environment Properties". Click **Add Property**, and add the properties, as shown in the following figure:

Figure 36. Add property of the target environment



Add the same set of properties and values to each target environment that you added in the Step 4.d.

The plain textual file "fvtsym.prop" is the file to define the symbos and the environment specific values for this environment. The file is in a simple format with a list of key-value pairs, and each key-value pair takes one line. You can refer to the file supplied in the template package as a sample to compile your own.

The plain-textproperty file provides the flexibility for you to add any new symbols or values, without changing whatever you defined in the IBM Urbancode Deploy. The property file can be upgraded and managed along with all other artifacts in the source control management solution. The alternative is to define the symbols as the properties on each target environment Then you must modify the configuration to add, update, or remove a property in IBM Urbancode Deploy every time that you need to make a change to a symbol.

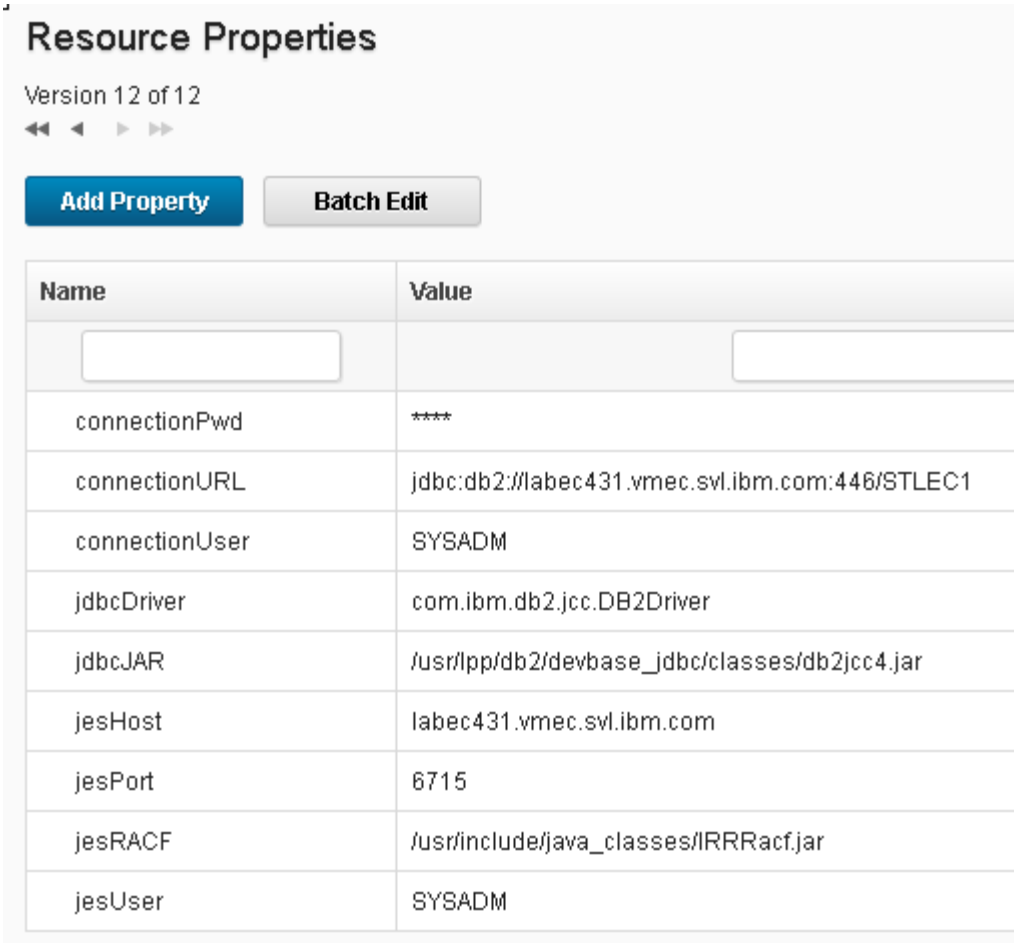
Another advantadge of using the property file is that you can easily share the same set of real values among various target environments. However, if you define the symbols as the properties directly in the IBM Urbancode Deploy, you might have to duplicate the definition in multiple places.

As you already learned from the preceding steps, you use symbols in the BIND card, utility jobs, and SQL statements, which enable you to easily add new syntax from future DB2 releases. You just check in the new artifacts and import them as a new version into the component, and run the deployment process.

Now add the required properties, such as the JDBC driver and JES information, to the underlying resource.

On the Resources page, click the resource that you added to your target environment, and switch to the Configuration panel. Add the properties on "Resource Properties" tab, as shown in the following figure:

Figure 37. Add resource property



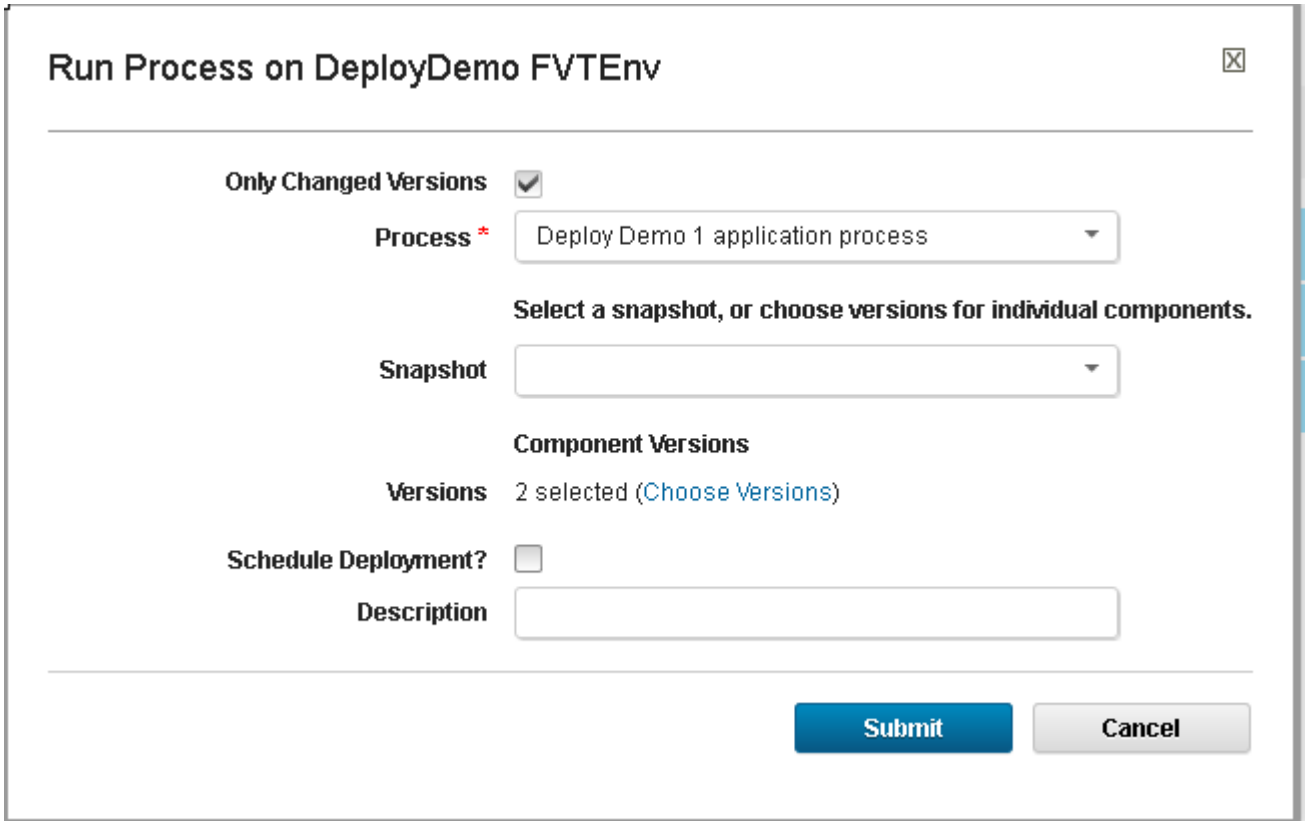
You need to add these properties to all the resources that you will use.

Everything is defined well now, and you are ready to deploy.

Step 6: kick off the entire deployment process

On the Applications page, click **Request Process**, which is a small one beside your target environment, against the target environment. Choose the corresponding process and proper version for each component like below, then submit the request.

Figure 38. Run deployment process



A page opens where you can monitor the results of the deployment. Now it is time to grab a (or another) cup of coffee!

You should see the process run successfully after you return. Congratulations!

Deploy the application with batch command

IBM Urbancode Deploy supports REST APIs and CLI commands, which enable you to create all the required elements of the deployment process and to run the deployment process.

You can build a batch script, or a job, or even a program to implement the entire or part of work that is described above above with REST APIs or CLI commands based on your needs, for example, you can use a batch script to add a target environment and to deploy the application to that instead of going through the GUI interface described above. The batch script or job can be integrated in or launched by the specific procedure in your shop.

This can also be integrated with a database provisioning solution in a cloud computing environment. For example, a developer in your shop might request a database environment for unit testing of certain applications on demand, the integrated solution

can build an ad hoc subsystem with the applications deployed per developer's specific request. The environment can be de-provisoned after the developer's work is done. Furthermore, the entire solution can be made as a self-service.

Below is a sample for adding a new target environment and to run the deployment process aginst the new added environment with REST APIs.

It uses the same set of steps from above. curl is an open source command line tool for sending data with URL syntax. It is used to send requests with REST APIs to IBM Urbancode Deploy server in the sample. Each command must be put in one line. Change the server and parameters based on your environment correspondingly. All the referenced JSON files can be found in the template package.

1. create a new target environment:

```
curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd>
"https://plxeditor.usca.ibm.com:8443/cli/environment/createEnvironment?application=
Deploy%20Demo&name=Deploy%20ProvEnv" -X PUT
```

2. create a resource to map the component:

```
curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd>
https://plxeditor.usca.ibm.com:8443/cli/resource/create -X PUT -d @UTBaseRes.json

curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd>
https://plxeditor.usca.ibm.com:8443/cli/resource/create -X PUT -d @UTAgent.json

curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd>
https://plxeditor.usca.ibm.com:8443/cli/resource/create -X PUT -d @UTResPrep.json

curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd>
https://plxeditor.usca.ibm.com:8443/cli/resource/create -X PUT -d @UTResComp.json
```

3. add base resource to the environment:

```
curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd>
"https://plxeditor.usca.ibm.com:8443/cli/environment/addBaseResource?environment
=Deploy%20ProvEnv&application=Deploy%20Demo&resource=%2FDeployNewResou
rce" -X PUT
```

4. add environment properties:

curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd> "https://plxeditor.usca.ibm.com:8443/cli/environment/propValue?environment=Deploy%20ProvEnv&application=Deploy%20Demo&name=srcDirectory&value=/u/oeusr05/testsrc" -X PUT

curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd> "https://plxeditor.usca.ibm.com:8443/cli/environment/propValue?environment=Deploy%20ProvEnv&application=Deploy%20Demo&name=symbolicReplacement&value=fvtsym.prop" -X PUT

5. add resource properties:

curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd> "https://plxeditor.usca.ibm.com:8443/cli/resource/setProperty?resource=%2FDeployNewResource%2FLABEC431.vmec.svl.ibm.com%2FDeployDemo&name=connectionURL&value=jdbc:db2://labec431.vmec.svl.ibm.com:446/STLEC1" -X PUT

curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd> "https://plxeditor.usca.ibm.com:8443/cli/resource/setProperty?resource=%2FDeployNewResource%2FLABEC431.vmec.svl.ibm.com%2FDeployDemo&name=connectionPwd&value=c0deshop&isSecure=true" -X PUT

curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd> "https://plxeditor.usca.ibm.com:8443/cli/resource/setProperty?resource=%2FDeployNewResource%2FLABEC431.vmec.svl.ibm.com%2FDeployDemo&name=connectionUser&value=SYSADM" -X PUT

curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd> "https://plxeditor.usca.ibm.com:8443/cli/resource/setProperty?resource=%2FDeployNewResource%2FLABEC431.vmec.svl.ibm.com%2FDeployDemo&name=jdbcDriver&value=com.ibm.db2.jcc.DB2Driver" -X PUT

curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd> "https://plxeditor.usca.ibm.com:8443/cli/resource/setProperty?resource=%2FDeployNewResource%2FLABEC431.vmec.svl.ibm.com%2FDeployDemo&name=jdbcJAR&value=/usr/lpp/db2/devbase_jdbc/classes/db2jcc4.jar" -X PUT

curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd> "https://plxeditor.usca.ibm.com:8443/cli/resource/setProperty?resource=%2FDeployN

ewResource%2FLABEC431.vmec.svl.ibm.com%2FDeployDemo&name=jesHost&value=labec431.vmec.svl.ibm.com" -X PUT

curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd> "https://plxeditor.usca.ibm.com:8443/cli/resource/setProperty?resource=%2FDeployNewResource%2FLABEC431.vmec.svl.ibm.com%2FDeployDemo&name=jesPort&value=6715" -X PUT

curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd> "https://plxeditor.usca.ibm.com:8443/cli/resource/setProperty?resource=%2FDeployNewResource%2FLABEC431.vmec.svl.ibm.com%2FDeployDemo&name=jesRACF&value=/usr/include/java_classes/IRRRacf.jar" -X PUT

curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd> "https://plxeditor.usca.ibm.com:8443/cli/resource/setProperty?resource=%2FDeployNewResource%2FLABEC431.vmec.svl.ibm.com%2FDeployDemo&name=jesUser&value=SYSADM" -X PUT

6. launch process:

curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd> https://plxeditor.usca.ibm.com:8443/cli/applicationProcessRequest/request -X PUT -d @deployDemoProvEnv.json

If you only want to run the deployment process against an existing target environment, then the last command is all you need. You can see how easy it is to run the deployment across dozens of environments in one batch.

Summary

In this tutorial, you learned how to build the deployment process for a DB2 application to roll out an application in various environment including multiple tenant environments with symbolic substitution, how to resolve the situation of creation vs. alternation of the data objects, and how to launch the DB2 utilities and commands with RACF PassTicket support in the sample.

The tutorial also elaborated the usage with REST APIs and CLI commands, and how to deploy the application on an ad hoc environment with REST APIs.

Resources

- Find more useful resources on Urbancode Deploy.
- Find more plugins on Urbancode Deploy Plugins.