

# Provision and Deploy Database Applications in Various Environments

---

This tutorial describes defining a process to deploy applications that use Db2 for z/OS. The deployment process uses the following software products:

- IBM UrbanCode Deploy (UCD)
- IBM Db2 Change Management Solution Pack for z/OS
- z/OS Management Facility for z/OS Version 2.2 (z/OSMF)

The deployment process includes steps for the following activities:

- Provisioning a database schema
- Deploying database schema changes
- Reviewing changes and the deployment process
- Binding or rebinding packages for application changes
- Running Db2 commands and utilities
- Deploying Db2 REST services

## Before You Begin

---

Ensure that the following prerequisites are ready:

- IBM UrbanCode Deploy Version 6.2 server, agent, and toolkit
- IBM z/OS Management Facility Version 2.2 server
- Source-code management client tools (RTC tools or Git tools) installed on the UCD server
- IBM Db2 Change Management Solution Pack for z/OS, specifically, Db2 Administration Tool and Db2 Object Comparison Tool Version 11.2 with APAR [PI67731](#)/PI72396/PI76054
- JDBC driver and RACF PassTicket jar files are available on the Unix System Services on your target subsystem (consult your administrator for more information)
- The following plugins are downloaded and installed from IBM UrbanCode Deploy Plugins:
  - [File Utils plugin](#)
  - [SQL-JDBC plugin](#)
  - [Web Utilities plugin](#)
  - [zOS Utility plugin](#)
  - [z/OSMF plugin](#)
  - [Rational Team Concert - SCM plugin](#) if you use RTC as the source-code management (SCM)
  - [Git plugin](#) if you use GIT for SCM

- Proper access to UCD, z/OSMF, and SCM, and read and execution privileges for the files and folders from APAR [PI67731/PI72396/PI76054](#)
- [Database objects required by Db2 REST services](#)

## Scenarios

---

This tutorial discusses steps for solutions for the following scenarios:

- **Deploying a test environment:** Developers or testers want a service to deploy a test environment with a database schema like another environment. The service can be invoked by a command or by any program or script.
- **Deploying application and database schema changes:** Developers or testers want a simple service that deploys checked-in application and database schema changes to target environments that they are authorized to deploy to. The database schema change deployment may involve complicated database operations including DDL, commands, and utilities. This scenario exploits Db2 Administration Tool for z/OS and Db2 Object Comparison Tool for z/OS to compare the new schema to the target environment to create the process required to convert the target environment to the checked-in schema version.
- **Reviewing changes and deployment process:** Administrators, including DBAs, security administrators, storage administrators, and others might need to review and approve the changes and deployment process in a controlled environment before or during the deployment.
- **Binding or rebinding an application:** Administrators or developers have an application that requires binding a new plan, or rebinding an existing plan to change the package list. The application might require new packages or package versions to be bound in the target environment. The packages might use different bind options. You might already have BIND cards defined for the packages to be bound.
- **Deploying Db2 REST services:** DBAs or developers need to create or delete native Db2 REST services on target environments. The REST services can be deployed using one or many property files and with or without bind options.

## Implement with IBM UrbanCode Deploy

---

The sample template in the tutorial contains:

- Components containing the artifacts.

- Processes that prepare the artifacts and drive the provision and deployment.
- Environments where the deployment runs.
- Application which associates the components and environments.

The tutorial leads you through the following tasks to build the flows:

1. Import the template.
2. Configure the components to associate with SCM systems.
3. Customize the process.
4. Add target environments and corresponding environment property files.
5. Setup the deployment style and notification.
6. Validate the process.

## Create the Solution

---

This tutorial describes how to build deployment processes to address the above scenarios. It also talks about how to integrate the services with different SCM systems and how to automate the entire solution.

The services use existing plugins in IBM UrbanCode Deploy, and the z/OSMF workflows provided by Db2 Administration Tool and Db2 Object Comparison Tool. All of the artifacts used and files referenced in this tutorial are available in the template package that you can download.

All steps described in this section are completed in the IBM UrbanCode Deploy web interface.

## Part I. Provisioning and Deployment Processes

### Task 1. Import the Application

1. Add a role named "DBA". Go to the **Settings** page, click **Role Configuration**, and click **Create Role**. In the **Name** field, type `DBA` and save.

This role is assigned to user IDs for database administrators who can review and approve the schema changes to a controlled environment. The role name can be changed later, or if you already have a role defined for the purpose, you can drop it after you import the application. The tutorial describes how to set and change the role name later.

2. Go to the **Applications** page, click **Import Applications** and check **Import with Snapshots**.

3. Click **Browse** and choose the downloaded template package file. Then click **Submit** to import the application. You can now see the new application named **ApplicationDeployment** on the **Applications** page.

## Task 2. Configure the Components

1. Go to the **Components** page and select the option for your SCM system:
  - For **Git**, select the **AppDeploywithAOCGIT** component.
  - For **RTC**, select the **AppDeploywithAOCRTC** component.

If you use other types of SCM systems, see [Creating components from source-code management systems](#).

2. Go to the **Configuration** panel under the component and complete the SCM system information in the **Version Source Configuraiton** section.
  - For **Git**, ensure that Git is installed and is executable by the UCD server, so that the UCD server can communicate with the Git server. The types of files in **Includes** are just samples, and you can fill in the types that your deployment requires.
  - For **RTC**, ensure that RTC SCM tools is installed and is executable by the UCD server, so that the UCD server can communicate with the RTC server. The file types in **Includes** are just samples, and you can fill in the types that your deployment requires.
3. Go to the **Versions** panel under the component and click **Import New Versions** to validate the SCM settings. A new version is shown in the list if the configuration is correct. You might need to click **Refresh** under the list table to refresh the content.

## Task 3. Customize the Processes

The processes in the template package provide the capabilities to accomplish the following tasks:

- Provision a schema like a source environment on a target environment with or without intervention.
- Deploy a schema change to a target environment with or without intervention.

Intervention can be a required at this stage for review and approval by a certain role, such as a DBA or Security Administrator, before the changes are completed in a controlled environment; or the intervention can also be a pause by intention at a certain point to guarantee the changes to fit into a change window; or other proper reasons based on the routine protocol in

your shop. The tutorial discusses customization of the process without intervention and the additional steps of the process with intervention.

## Process 1: Schema Provisioning

**ProvisionLikewithoutApproval** and **ProvisionLikewithApproval** uses the SQL-JDBC plugin to run DDL to provision schemas. **ProvisionLikewithoutApprovalCLP** and **ProvisionLikewithApprovalCLP** are available to use CLP to run DDL instead of the SQL-JDBC plugin. Using CLP is necessary if CALL statements are expected to run from the DDL. See Part III for more details on using CLP to provision a schema like a source environment to a target environment.

All environment-level and resource-level properties used in the processes will be covered later. Environment-level properties are referenced as `${p:environment/propertyName}` and resource-level properties are references as `${p:resource/propertyName}`.

1. Go to the **Components** page and select the component you chose earlier. Go to the **Configuration** panel, click **Component Properties**, and review and customize the properties.
2. Go to the **Processes** panel under the component that you chose earlier. Click on the **ProvisionLikewithoutApproval** process to review and customize.
3. The **downloadPropertyFiles** step downloads the required artifacts to the target environment. Open the step, customize the properties, and save the changes. You can choose to download certain types of files and exclude others in the step.
4. Open the **substituteSymbolics** step, customize the properties, and save the changes.

You can use the **Property List** to replace the tokens in the downloaded artifacts with the key-value pair that is defined at environment level. This is useful when you have any environment-specific values to overwrite the input variables to the underlying tools that the process invokes. Alternatively, you can use **Property File Name** to define a textual property file with key-value pairs to replace the tokens in the download artifacts.

5. Open the **prepareWorkflow** step, customize the properties if needed, and save the changes.

The script defined in the step converts the download artifacts to IBM-1047 encoding to invoke the workflow in the following steps. This step is only needed if you don't have environment-specific values to overwrite the properties in the input variable file. If you do have environment-specific values to overwrite the values in the previous step, then you

don't need this step. The encoding is converted automatically during the token replacement.

6. Open the **createProvisionWorkflow** step, customize the properties and save the changes.
7. Open the **startProvisionWorkflow** step, customize the properties and save the changes.
8. Open the **getExtractedDDL** step and change the **Password** property to `${p:environment/workflowPwd}` . Customize the other properties and save the changes.

The password is defined at the environment level later. This step gets the extracted data definition statement content from the source environment, and the next step stores the data definition statement content into a file that can be saved in SCM and reused in other rounds of provision, or reviewed by DBAs in the approval process.

9. Open the **genSchemaFile** step, customize the properties if need, and save the changes.
10. Open the **provisionSchemaLike** step, customize the properties if need, and save the changes.

All the steps are now reviewed and customized for the provisioning process without approval.

If an intervention is required, for example, review and approval from DBAs before the process provisions the target environment, a review and approval step can be added. The steps below explain how to setup the process for intervention.

11. Go to the **Processes** panel under the component that you chose earlier. Click on the **ProvisionLikewithApproval** process to review and customize.
12. A **Manual Task** step is added right before the **provisionSchemaLike** step. Open the **Manual Task** step. This is to send the extracted data definition statement content to the DBA role, which was created in Task 1, for review and approval. Customize the properties if you have a different role name.

## Process 2: Deploy Schema Changes

All environment-level and resource-level properties used in the processes will be covered later. Environment-level properties are referenced as `${p:environment/propertyName}` and resource-level properties are references as `${p:resource/propertyName}` .

1. Go to the **Processes** panel under the component that you chose earlier. Click on the

**UpgradeApplicationSchemawithoutApproval** process to review and customize.

2. Open each step and customize the values for the properties based on your environment.

The steps are very similar to the first 5 steps in the **ProvisionLikewithoutApproval** process and the **ProvisionLikewithApproval** process.

In some shops, you might want to pause after the compare is done, but before applying the delta changes to the target environment. You might do that for the following reasons, among others:

- So that Administrators can review and approve the final changes in a controlled environment.
- To separate the compare from the application and the final changes because the compare takes too long in a complicated environment to fit the entire process into the change window of a target environment.

In such cases, the Db2 Object Comparison Tool workflow must be executed step-by-step, so the application deployment process can pause when necessary. The steps below explain how to setup the process for intervention.

3. Go to the **Processes** panel under the component that you chose earlier. Click on the **UpgradeApplicationSchemawithApproval** process to review and customize. Open each step and customize the values for the properties based on your environment.
4. Open the **startCompareWorkflow** step. The difference here is to run the workflow step by step instead of the entire workflow. Look at the **Step Name** and **Perform Subsequent** values. Then do the same for the **startAnalyzeWorkflow** step, which analyzes the comparison result and determines the final changes.
5. Open the **Manual Task** step. This is to send the analyze result to the DBA role, which was created in Task 1, for review and approval. Customize the properties if you have different role name. The default value of the only added property refers to the output property of the previous **startAnalyzeWorkflow** step, for the content of the schema changes from the analysis at the execution time.
6. The **startDeployWorkflow** step rolls out the final changes to a target environment if the DBA approves the changes in the previous step during the execution. The properties of the step are similar to the **startAnalyzeWorkflow** and **startCompareWorkflow** steps. Open the step and review the properties. The final deployment result is shown in the output property of the step at execution time.

Next, define the target environment and setup the properties referenced by the steps in the processes.

## Task 4. Add Target Environments and Properties

1. Go to the **Resources** page, click **Create Top-Level Group**, and enter a name.
2. Open the resource group, click **Create...** and choose **Agent**. Select an agent from the list and save.
3. Click on the agent you just created, then click **Create...** and choose **Component**. Select the component from the list that you chose earlier and save.
4. Go to the **Configuration** panel under the agent, click **Resource Properties**, and add the following properties with customized the values based on your environment:

Name	Value
connectionURL	jdbc:db2//sysmvs1.svl.ibm.com:446/STLEC1
connectionUser	SYSADM
jdbcDriver	com.ibm.db2.jcc.DB2Driver
jdbcJAR	/usr/lpp/db2/devbase_jdbc/classes/db2jcc4.jar
jesHost	sysmvs1.svl.ibm.com
jesLibPath	/usr/lib
jesPort	6715
jesPTID	SYEC1DB2
jesRACF	/usr/include/java_classes/IRRRacf.jar
jesUser	SYSADM

These properties are used to run the SQL statements in the last step of the process by the SQL-JDBC plugin. The plugin uses the RACF PassTicket, so no password is required. For descriptions about each resource property, see the Appendix section.

5. Go to the **Applications** page and click the **ApplicationDeployment** application. Two environments are already defined by default. You can choose to rename the environment

or create your own by clicking **Create Environment**.

6. Click on the environment you've just created. On the **Resources** panel, click **Add Base Resources**, select the root resource group, and click OK.
7. Go to the **Configuration** panel under the environment, click **Environment Properties**, and customize the properties based on your environment. For descriptions about each environment property, see the Appendix section.

## Task 5. Setup Deployment Style and Notification

You can schedule the deployment to run automatically. Choose one of the following methods of automating the deployment:

### Option 1

1. Go to the **Applications** page, click the **ApplicationDeployment** application, and then click **Request Process** beside the created environment.
2. Check **Schedule Deployment** to set the date and time to kick off the deployment at that time. You can also make the deployment happen recursively by checking **Make Recurring** if there is a need to provision or deploy application change on a regular basis.
3. Click Submit to deploy the process.

### Option 2

1. Go to the **Components** page and click the component that you configured in the previous tasks.
2. Go to the **Configuration** panel under the component, and click **Basic Settings**.
3. Check **Import Versions Automatically** to let UCD server import a newly checked-in version (a version after build succeeds) automatically using the SCM information that you set in Task 2. This is useful when developers want to automate the deployment of any application and schema changes right after they check in the code into SCM and the code is built successfully, with the integration of UCD, SCM and build systems.
4. You can also check **Run Process after a Version is Created** to kick off the chosen process immediately after the new version is imported.

Notification is a preferable way to know when a process is kicked off, whether it succeeds or fails, when a process has ended, whether a review or approval is needed or pending,

and so forth.

5. To set the notification, go to the **Settings** page if you are an administrator for the UCD server. Otherwise, you might need the administrator to grant you the related privileges. Click **Notification Schemes**, then either **Create Notification Scheme** or modify an existing one.
6. Click the notification scheme you want to customize, then click **Add Notification Entry** to add an entry.

## Task 6. Validate the Processes

The sample application processes are already created in the template and are ready to validate the entire flow. You can review and customize the processes by going to the **Applications** page, clicking **Application Deployment**, and clicking the **Processes** panel.

You are ready now to validate what you have configured so far.

1. Go to the **Applications** page and click the **ApplicationDeployment** application.
2. Under the **Environments** panel, click the **Request Process** icon next to the target environment.
3. Select and submit the process.

UCD brings you to the page which displays the execution status of the process. Wait for the process complete and check the status.

Everything in green means that you have done a great job!

## Part II. Bind Package Process

The template imported in Part I also contains the application, components, processes and all artifacts to be discussed here. Because the SCM system configuration for the components to be discussed here is the same as that in Part I, we are going to skip the import and SCM setup steps. For instructions for the import and SCM setup steps, see Task 1 and Task 2 in Part I if necessary.

The deployment style and notification setup for the solution in Part II is also similar to the same Part I, so we are also going to skip this part. See the Task 5 in Part I if you want to configure the components of Part II in the same way.

## Task 1. Customize the Prepare Process

Implement the process that prepares DBRMs for bind and related BIND jobs based on the BIND cards.

1. Go to the **Components** page and click the **DeployDemoPrep** component. Go to the **Processes** panel, and click **RDProcess** to review and customize.
2. Open the **FTP Artifacts** step, customize the properties, and save the changes.

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.

3. Open the **Deploy Data Sets** step, customize the properties, and save the changes.

You use an environment property for the PDS Mapping because the target dataset name might be different for each individual environment. You define the environment property later. **Allow Creating Data Set** is set to `TRUE` to enable IBM UrbanCode Deploy to create the target PDS dataset with the default settings, if the dataset is not available. Set the value 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.

4. Open the **GenBndCard** step, customize the properties, and save the changes.

This step iterates all of the PDS dataset members and generates the text, BIND card, in this sample with the **Template** input.

For the **Template** property, `${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 UCD. You can use other special characters 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 the following **Template** value if your shop has the BIND cards stored in one or multiple files:

```
@${member}@ -  
LIBRARY('${dataset}')
```

Here, `${member}` is a property name to be replaced later with the real value, in the symbolic replacement step. While the real BIND card is kept in one or multiple text property files. The property key-value pairs are defined in the following multi-line format (taking the DSNADMCD DBRM member as a sample here):

```
DSNADMCD = \
BIND PACKAGE(DB20SC) MEMBER(DSNADMCD) - \n\
ACTION(REPLACE) ISOLATION(CS) - \n\
RELEASE(COMMIT) ENCODING(EBCDIC)
```

Thus, you can have different combination of BIND options for different DBRM members.

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.

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.

5. Open the **GenBndFile** step, customize the properties, and save the changes.

Use a component property for the name of the BIND job because this can be same for all environments. You define this property later. For **Contents**, enter 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. **Working Directory** defines a directory to contain the BIND job. This will be the same directory for other artifacts in the main deployment process. Note that you can use the same method explained in the previous step to set a property for job header to generate different headers for different environments, if necessary.

## Task 2: Add Properties and Artifacts for Prepare Process

1. Go to the **Components** page and click the **DeployDemoPrep** component. On the **Configuration** panel, switch to the **Component Properties** panel, and customize the properties based on your environment.
2. (Optional) Go to the **Environment Property Definitions** and customize the following properties based on your environment.

Now you probably notice an interesting thing: As mentioned earlier, these two properties,

`pdsMapping` and `srcDirectory` , 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, those values will 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.

3. Import the DBRMs into the **DeployDemoPrep** component 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 can find more detail of BUZTOOL in [Creating z/OS component versions](#).

After it is done, you will the list of imported artifacts under the **Versions** panel.

The **DeployDemoPrep** component is now ready. 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.

### Task 3. Customize the Bind Process

Build the main process that binds the packages.

1. Go the the **Components** page and click the **AppDeployPackageRTC** or **AppDeployPackageGIT** component.
2. On the **Processes** panel under the component, click the **BindPackages** process to review and customize.
3. Open **Download Artifacts** step, customize the properties, and save the changes.

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.

4. Open the **replaceTokensPropFile** step, customize the properties, and save the changes.

In this step, you substitute the symbols including BIND card or Job header if applicable in the bind job which is generated in the first step. Use **Start Token Delimiter** and **End**

**Token Delimiter** to define a token delimiter for your shop. The default delimiter is the @ character. 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 directory points to the same folder that contains all the artifacts on the target environment.

5. Open the **bindPackage** step, customize the properties, and save the changes.

You can also use the method described here to launch other Db2 commands, or utilities such as RUNSTATS, REORG, CHECK DATA, and others.

RACF PassTicket is supported to run JCL too, so you don't have to store the password in the process. The real values of other fields are defined at the resource level, such as the JDBC properties.

For the details for RACF PassTicket support in the Submit Job of the zOS utility plugin, see [z/OS Utility plug-in](#). You must ensure that the RACF PassTicket is configured on the target subsystem successfully for Job Monitor that is used by the plugin here, before you run your deployment process in the last step.

6. Customize the required component properties. Go to the **Components** page and click the **AppDeployPackageRTC** or **AppDeployPackageGIT** component. On the **Configuration panel**, switch to **Component Properties** then customize and save the properties.

## Task 4. Add Environment and Component Properties for Bind Process

1. Go to the **Resources** page, and open resource that you created earlier. Click on the agent and add the components **DeployDemoPrep** and **AppDeployPackageRTC** or **AppDeployPackageGIT** from the list.
2. Open the environment that you created earlier or create a new environment by following the same steps in Part I. Customize and save the properties.

The `bindsym.prop` text file defines the symbols and the environment-specific values for this environment. The file is in a simple format with a list of key-value pairs, with each key-value pair on one line. You can refer to the file supplied in the template package as a sample to compile your own.

The property file provides the flexibility for you to add any new symbols or values, without changing whatever you defined in 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 UCD every time that you need to make a change to a symbol. 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 advantage 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 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 and utility jobs, 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.

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

## Task 5: Validate the Entire Bind Package Process

The sample application processes are already created in the template and are ready to validate the entire flow. You can review and customize the processes by going to the **Applications** page, clicking **Application Deployment**, clicking the **Processes** panel.

You are ready now to validate what you have configured so far.

1. Go to the **Applications** page and click **ApplicationDeployment**.
2. Under the **Environments** panel, click the **Request Process** icon next to the target environment.
3. Select and submit the process.

UCD brings you to the page which displays the execution status of the process. Wait for the process complete and check the status.

Now it is time to grab a cup of coffee, or another! You can see the process run successfully after you return. Congratulations!

## Part III. Provisioning with Db2 command line processor (CLP)

You can provision schemas by using the Db2 command line processor (CLP) on z/OS Unix System Services (USS). Consider using CLP instead of the SQL-JDBC plugin when CALL statements are needed in the DDL statements. The current UCD SQL-JDBC plugin does not support CALL statements. The CLP processes are part of the **AppDeploywithAOCGIT** component and are currently available only with Git.

You must set up RACF on the target Db2 system to use the CLP process. Consult your security administrator to setup RACF and for valid RACF userids.

The template imported in Part I also contains the application, components, processes, and all artifacts to be discussed here. Because the SCM system configuration for the **AppDeploywithAOCGIT** component is the same as that in Part I, we are going to skip the import and SCM setup steps. See Task 1 and Task 2 in Part I if necessary.

## Task 1: Customize the Processes

1. Go to the **Components** page and click the **AppDeploywithAOCGIT** component. On the **Processes** panel, click the **ProvisionLikewithoutApprovalCLP** process to review and customize.
2. Because this process is very similar to the **ProvisionLikewithoutApproval** process, see Task 3 in Part I to modify the properties in each step. Again, the environment-level, component-level, and resource-level properties are covered later.
3. For the **getExtractedDDL** step, change the Password property to `${p:environment/workflowPwd}` and click **OK**.
4. For the **provisionSchema** step, a shell script is used to set and export environment variables in the USS environment, run the groovy script to generate a RACF PassTicket, and use the Db2 CLP to read input from the generated SQL file. Customize the **Shell Script** property and other properties if needed.

Ensure that `genticket.groovy` is in the working directory of the CLP process. Other work files can be used, but `genticket.groovy` is required to generate the RACF PassTicket.

5. After customizing the **ProvisionLikewithoutApprovalCLP** process, click **Save** to remember the changes.

All the steps are now reviewed and customized for the provision process with CLP without approval.

If an intervention is required, for example, review and approval from DBAs before the

process provisions the target environment, a review and approval step can be added. The steps below explain how to setup the process for intervention.

6. Go to the **Components** page and click the **AppDeploywithAOCGIT** component. On the **Processes** panel, click the **ProvisionLikewithApprovalCLP** process to review and customize.
7. A Manual Task step is added right before the **provisionSchema** step. Open the **Manual Task** step. This is to send the extracted data definition statement content to the DBA role, which was created in Task 1, for review and approval. Customize the properties if you have a different role name.

## Task 2: Modify Agent and Environment Properties

1. Go to the **Resources** page. On the Agents panel, select the agent that you created in Part I.
2. On the **Configuration** panel, click **Agent Properties**. Add or modify the following properties with customized values based on your environment:
  - **CLASSPATH** includes the full path to the CLP main class jar file and RACF PassTicket jar files.
  - **CLPHOME** is the full path to the base directory of the Db2 CLP installation.
  - **DB2HOME** is the full path to the Db2 CLP and JDBC driver USS installation directory.
  - **GROOVY\_HOME** is the full path to the Groovy installation directory which comes along with UCD agent installation.
  - **JAVA\_HOME** is the full path to the JDK installation directory.
3. Go to the **Applications** page and click the **ApplicationDeployment** application. Select the environment you created in Part I.
4. On the **Configuration** panel, click **Environment Properties**. In the **Filter By Component**, select **AppDeploywithAOCGIT**, and review the environment properties. For descriptions about each environment property, see the Appendix section.

The agent and environment properties have now been customized to run the CLP process on your environment.

To setup deployment style, notifications, and to validate the processes, see to Task 5 and Task 6 in Part I.

## Part IV: Deploy Db2 REST Services

The template imported in Part I also contains the application, components, processes and all artifacts to be discussed here. Because the SCM system configuration for **DeployRESTService** is the same as that in Part I, we are going to skip the import and SCM setup steps. See Task 1 and Task 2 in Part I if necessary.

## Task 1. Customize the Processes

The **DeployRESTService** component has three processes:

- **DeployRestServicewithBindCombine**: Create a REST service using one file
- **DeployRestServicewithBindSeparate**: Create a REST service using separate files.
- **DropRestService**: Drop an existing REST service.

All sample processes use HTTP protocol to make HTTP requests calls. HTTPS protocol can be used instead if environment is configured to support SSL.

**DeployRestServicewithBindSeparate** process is the preferred method to create REST services. Thus, this tutorial will go over the separate process. All environment-level and resource-level properties used in the processes will be covered later.

1. Go the **Components** page and click the **DeployRESTService** component. Go to the **Configuration** panel, click **Component Properties**, and review and customize the properties.
2. Go to the **Processes** panel, and click the **DeployRestServicewithBindSeparate** process to review and customize.
3. The **downloadPropertyFiles** step downloads the required artifacts to the target environment. You can choose to download certain types of files and exclude others in the step.
4. The **getServiceInfo** step reads and retrieve property values from an external file to be used in the process.
5. The **prepareServiceInfo** step uses a script to converts the files to IBM-1047 encoding to be used in subsequent steps.
6. The **substituteEnvSymbolics** step replaces tokens in the downloaded artifacts with environment-level key-pair values.
7. The **getSQLStatement** step retrieves and formats the SQL statement to be used in the REST service.

8. The **prepareSQL** step sets the SQL statement as a variable to be used in the process.
9. The **checkBindOption** step checks for existing files that include bind options to append to the body of the HTTP request. If there are no bind options, the body of the HTTP request does not include any bind options.

The **generateBindOption** step uses the output from the **checkBindOption** step to determine how to create the body of the HTTP request. The next steps continues on one of the following paths:

- If there are bind options, the **generateServiceDefinitionFilewithBindOptions** step creates a JSON file with the required parameters. Then, the **appendBindOptions** step adds the additional bind options.
  - If there are no bind options, the **generateServiceDefinitionFile** step creates a JSON file with the required parameters.
10. The **createRestService** step makes a REST call to the Db2 REST service manager API to create a Db2 REST service. The created JSON file is used as the data file for the REST call.
  11. The **grantExecutionPrivilege** step grants execution privileges on the REST service package.
  12. The **validateService** step invokes the created Db2 REST service to confirm the validity of the REST service and SQL statement.

One file can be used to define a REST service instead of having separate files for the service properties, SQL statement, and bind options. The **DeployRestServicewithBindCombine** process uses one service information file to create a REST service. If you prefer to use the **DeployRestServicewithBindCombine** process, use the preceding steps to customize if instead.

## Task 2. Add or Modify Properties

For instructions for customizing resource-level properties, see Task 4 in Part I.

Follow the steps below to customize environment-level properties:

1. Go to the **Applications** page and click the **ApplicationDeployment** application. Select the environment you created in Part I.
2. On the **Configuration** panel, click **Environment Properties**. In the **Filter By Component**,

select **DeployDemoPrep**, and customize the properties based on your environment. For descriptions about each environment property, see the Appendix section.

To setup deployment style, notifications, and to validate the processes, see to Task 5 and Task 6 in Part I.

## Deploy the Application with Batch Commands

---

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 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 other provisioning solution in a cloud computing environment. For example, a developer in your shop might request a Db2 for z/OS subsystem 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-provisioned after the developer's work is done. Furthermore, the entire solution can be made as a self-service.

The following example shows a sample command to launch an application process with a REST call.

```
curl -k -u <ucd-userid>:<ucd-userpwd> https://sysmvs1.svl.ibm.com:8443/cli/applicat
```

## Summary

---

In this tutorial, you learned how to build a solution to provision schema on an environment like the one with golden copy of the schema, to build a solution to deploy the schema changes to environments with previous versions, to build a solution to deploy packages (bind or rebind) against environments 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.

## Appendix

---

### Resource Properties (Agent)

Go to the **Resources** page and click on agent resource. On the **Configuration** panel, click **Resource Properties**.

- **connectionURL** is the JDBC connection URL for the target environment.
- **connectionUser** is the user name for the JDBC connection, and the name must be granted by RACF PassTicket for JDBC ahead.
- **jdbcDriver** is the driver name for Db2 for z/OS.
- **jdbcJAR** is the full path name pointing to the db2 jcc jar file on the USS of the target environment.
- **jesHost**, **jesLibPath**, **jesPort**, **jesPTID**, **jesRACF** and **jesUser** are RACF PassTicket values for JES. However, they are also shared by the RACF PassTicket for JDBC. The RACF PassTicket for JES allows UCD to run JCL statements without requiring a password stored outside.

### Component Environment Properties

Go to the **Applications** page and click the **ApplicationDeployment** application. On the **Environments** panel, select an environment. On the **Configurations** panel, click **Environment Properties**.

- **aocCMPProfDataset** is a dataset keeping the profile to the Db2 Object Comparison Tool workflow, which will be used in application deployment process. Add this property if you may also want to run that process against the environment, otherwise, skip this one.
- **aocCMPProfFile** is the name of the property file to keep your own values of the profile to the Db2 Object Comparison Tool workflow. Add this property if you may also want to run that process against the environment, otherwise, skip this one.
- **aocCMVars** is the name of the property file to keep the values of the parameters to the Db2 Object Comparison Tool workflow. Add this property if you may also want to run that process against the environment, otherwise, skip this one.
- **aocGenVars** is the name of the property file to keep the values of the parameters to the Db2 Administration Tool workflow.
- **aocMaskDataset** is a dataset keeping the mask definition to the b2 Object Comparison Tool workflow. Add this property if you may also want to run that process against the environment, otherwise, skip this one.

- **aocMaskDefFile** is the name of the property file to keep the values of the mask parameters to the Db2 Object Comparison Tool workflow. Add this property if you may also want to run that process against the environment, otherwise, skip this one.
- **aocWorkflowDirectory** is the full path of the folder where your Db2 Administration Tool and Db2 Object Comparison Tool workflows locate after APAR PI67731.
- **applid** is the application name defined in RACF profile on the target Db2 system to generate RACF passticket for Db2 command line processor.
- **collid** is the collection identifier of the package that is associated with the new REST service.
- **dbSchemaFile** is a file name for the extracted or downloaded DDL under USS.
- **ddlOutputDataset** is the dataset keeping the extracted data definition statement and content by the Db2 Administration Tool workflow. In this tutorial, **sourceDDLDataset** is the same dataset without the high qualifier, but they could be different. Also, the high-level qualifier uses **workflowUser** as the default. Customize the value correspondingly based on your environment.
- **defaultBindopt** is a file containing the options for binding the package that is associated with the new REST service.
- **extSrcEnv** is the ID of source Db2 subsystem from which the process extracts the source schema DDLs.
- **grantee** is the user who will be granted execution privileges on the new REST service package.
- **qualifier** is the implicit qualifier of unqualified names of tables, views, indexes, and aliases contained in the new REST service package.
- **serviceDefinition** is the JSON file containing the body of the HTTP request for the REST service.
- **serviceInfo** is the file containing properties for the REST service.
- **sourceDDLDataset** is the dataset keeping the source data definition statement and content used by the Db2 Administration Tool workflow. In this tutorial, **ddlOutputDataset** is the same with the high qualifier, but they could be different.
- **srcDirectory** is the path to the working directory.
- **systemID** is the ID of the target environment.
- **workflowPwd** is the password for the owner of the workflow to be created on z/OSMF.
- **workflowServer** is the hostname of the target environment where z/OSMF is running.
- **workflowServerLoc** is the unique location name of the target Db2 system defined during installation.
- **workflowServerPort** is the DRDA port number to connect to the target Db2 system.
- **workflowServerUser** is the user name which will be the owner of the workflow to be created on z/OSMF.

## Resources

---

- [UrbanCode Deploy Official Documentation](#)
- [UrbanCode Deploy Plugins](#)