

Deployit Cookbook - Create a Deployment Package using the Deployit UI

Version 3.8.0

Table of Contents

Table of Contents	2
Cookbook - Create a Deployment Package using the Deployit UI	3
Creating an application	3
Creating a Deployment Package	3
Adding Deployable content	4
Adding artifacts	4
Specifying property placeholders	5
Export as DAR	5

Cookbook - Create a Deployment Package using the Deployit UI

Deployment Packages are usually created outside of Deployit. For example, packages are built by tools like Maven or Jenkins and then imported using the a Deployit plugin. (See documentation Maven Deployit plugin and Jenkins Deployit Plugins). Or you manually write a Manifest.MF file for the Deployit Archive format (DAR format) and import the package using the Deployit UI.

But, while designing a Deployment Package this may be a cumbersome process. To quickly assemble a package, it is more convenient to edit it in the Deployit UI.

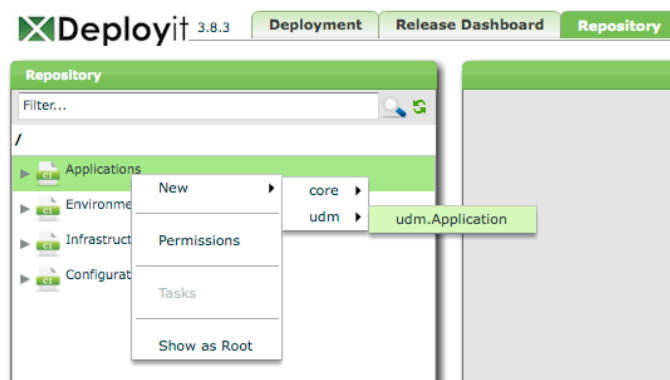
Note: You should be running Deployit 3.8 or higher.

Creating an application

In Deployit, all deployable content is stored in a Deployment Package. The Deployment Package will contain the EAR files, HTML files, SQL scripts, DataSource definitions, etc.

Deployment Packages are versions of an Application. An application will contain one or more Deployment Packages. So before we can create a Deployment Package, we need to create an Application.

To create an application, login to the Deployit UI and go to the **Repository** tab. Right-click on **Applications** and choose **New ► udm ► udm.Application**

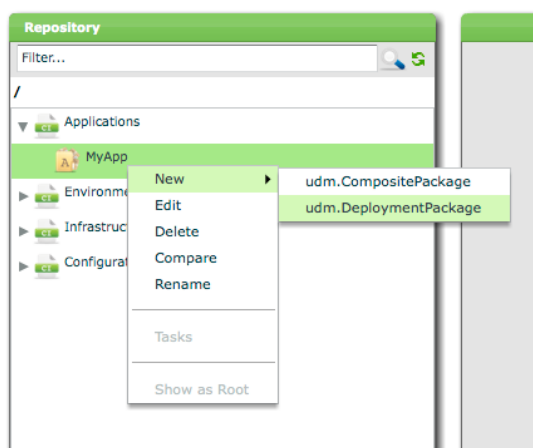


Give it the name 'MyApp' and press save.

Creating a Deployment Package

Now let's create a Deployment Package that has all the content of version 1.0 of MyApp.

Right-click on **MyApp** and choose **New ► udm.DeploymentPackage**



Give it the name '1.0' and press save.

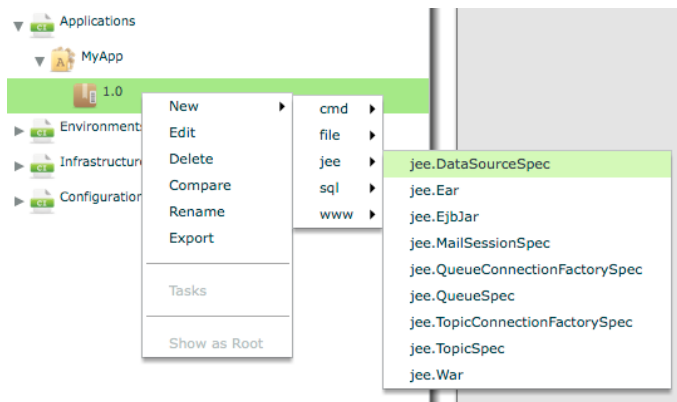
We now have an empty MyApp 1.0 package.

Adding Deployable content

We're now ready to add actual deployable content to the package. Remember that in Deployit, all configuration items (nodes in the repository tree) are *typed*. That is, you have to tell Deployit on beforehand which type a configuration item is, so Deployit will know what to do with it.

First we'll add a simple deployable without file content. Let's create a deployable `DataSource` in the package.

Right-click on **MyApp** and choose **New ► jee ► jee.DataSourceSpec**



Give it the name 'MyDataSource' and JNDI-name 'jdbc/my-data-source'. Press Save.

That's it! We've just created a functional Deployment Package that will create a `DataSource` when deployed to a JEE Application Server like JBoss or WebSphere.

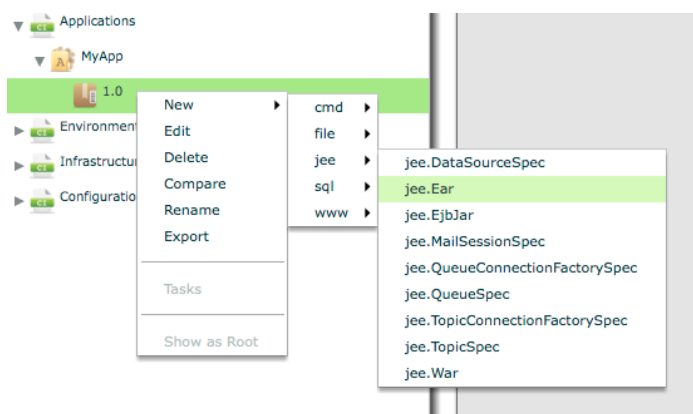
Adding artifacts

Artifacts are configuration items that contain files. Examples are EAR files, WAR files, but also plain files or folders.

Let's add an EAR file to our `MyApp/1.0` deployment package. It will be of type `jee.Ear`.

Note that if you're using specific middleware like WebSphere or WebLogic, you also have the option to add an EAR of type `was.Ear` or `wls.Ear`. Only use this if you really need WebSphere or WebLogic-specific features. The `jee.Ear` type will deploy just fine too.

Right-click on **MyApp** and choose **New ► jee ► jee.Ear**



Give it the name 'PetClinic.ear'. We can now upload the actual EAR file. Hit 'Browse file' and select an EAR file from your local workstation. If you're running the Deployit Server locally, you can find an example EAR file in `deployit-server/importablePackages/PetClinic-ear/1.0/PetClinic-1.0.ear`.

When creating artifacts (configuration items with file content), there are some things to take into account. First, you can only upload files when creating the configuration item. It's not possible to change the content afterwards. The reason for this is that Deployment Packages should effectively be read-only. If you change the contents, you may create inconsistencies between what has deployed onto the middleware and what is in the Deployit repository. This may lead to surprising errors.

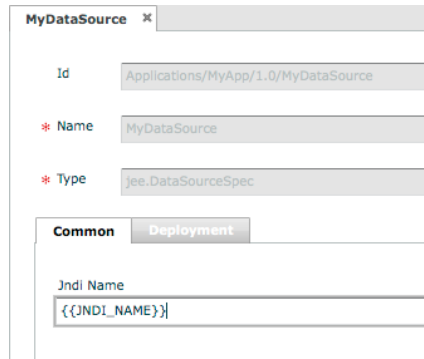
Placeholder scanning of files is only done when they're uploaded. Use the 'Scan Placeholder' checkbox to enable or disable placeholder scanning of files.

When uploading entire directories for the `file.Folder` type, you will need to zip the directory first, since you can only select a single file for browser upload.

Specifying property placeholders

It's easy to specify property placeholders. For any deployable configuration item, you can enter a value surrounded by double curly brackets. For example: `{{PLACEHOLDER}}`. The actual value used in a deployment will be looked up from a dictionary when a deployment mapping is made.

For example, open `MyDataSource` and enter `JNDI_VALUE` as placeholder:



The screenshot shows the configuration form for 'MyDataSource'. The 'Id' field contains 'Applications/MyApp/1.0/MyDataSource'. The 'Name' field contains 'MyDataSource'. The 'Type' field contains 'jee.DataSourceSpec'. Below these fields is a tabbed interface with 'Common' and 'Deployment' tabs. The 'Common' tab is active, showing a 'Jndi Name' field with the value '{{JNDI_NAME}}'.

The value for Jndi Name will be looked up from the dictionary associated with environment you deploy to.

Export as DAR

When you're finished modeling the application, you can export it as a DAR file. Once downloaded, you can unzip it and inspect its contents. For example, the generated manifest file can serve as a basis for automatic generation of the DAR.

To export as DAR, right-click on '1.0' and choose 'Export'

