Solving Application Installation Issues During Migration

Introduction

Each new release of IBM WebSphere Application Server provides new features and improves on existing features in the WebSphere runtime, the Java runtime, and in the Java EE technologies used by enterprise applications. The migration tools provided with WebSphere help bring the old configuration settings and the associated set of installed applications forward into the new WebSphere release. The main goal of a WebSphere migration is to have the new environment behave or function as close as possible to the old environment, thus allowing the new features to be leveraged over time.

Various application deployment issues can occur during the migration process in a mixed-cell environment due to “sleeper” constructs contained within the application. Although the application deployed and ran successfully in the old release environment, the unchanged application cannot deploy in the mixed-cell environment because the old node does not support applications configured with Java EE 6 constructs. By using the methods described in this paper, you can identify and handle these possible application installation issues when you migrate to V8.0 or V8.5.5.

Background

In a standard cell migration scenario, the deployment manager (dmgr) must be migrated first because it is required to be at the highest release level of all the nodes that make up the cell. The new dmgr is allowed to manage federated nodes running at the same release level or any of the three prior release levels. In this mixed-cell environment, the dmgr identifies which Java EE levels are supported by each node based on its WebSphere release level. Therefore, the V8.5.5 dmgr does not allow applications running at a Java EE 6 level to be deployed to nodes running a release prior to V8.0. During deployment, an application's Java EE level can be determined in three ways:

- By looking at the application's deployment descriptors
- By scanning the application code for certain annotations associated with a particular Java EE level
- By how the application itself is coded – for example, EJB's “no-interface view”.

Problem

During the migration process, you might encounter a scenario where an application deploys successfully onto a dmgr running V7.0, but this same application will not deploy onto that same dmgr after it is migrated to V8.0 or V8.5.5. This scenario occurs because the application contains at least one Java EE 6 construct that was previously ignored by the V7.0 deployment process. In V7.0 these so-called “sleeper” constructs are unused when the

application is deployed and when the application is running. When a dmgr migrates to V8.0 or V8.5.5, the applications are re-deployed to this new level - which is fully aware of Java EE 6 constructs. If these constructs are present, the application is automatically promoted, regardless of whether the application actually uses the constructs. And because the application is targeted to a V7.0 node, which only supports Java EE 5 constructs, the deployment process fails.

Some possibilities for how these Java EE 6 constructs might have been included in an application targeted for a Java EE 5 environment include:

- Developer tools might automatically add them, or present them to developers who unknowingly accept them.
- An application might package a third-party utility JAR file that includes at least one Java EE 6 construct.

Analyzing Applications For Java EE 6 Constructs

As always, you should thoroughly test the migration process, which includes migrating the applications, before you perform the migration in a production environment. During this testing phase you will need to determine which applications encounter the automatic app-promotion issues described in the previous section. Note that WebSphere's migration process simply re-deploys the applications “as-is” into the new release.

During this testing phase, it is recommended that you use WASPostUpgrade's “-includeApp script” parameter and option. Using this option separates the migration of the server's configuration settings from the re-deployment of the applications into the new environment. The “script” option produces the necessary installation scripts so that the applications can be deployed manually after WASPostUpgrade finishes its other work.

For applications that fail to deploy, search the log and trace files for indications that the application has been automatically promoted and cannot be deployed to a node running a previous release. The following Java EE 6 constructs are known to cause migration application deployment issues:

- No Interface View **(ejb 3.1)**
- Web Fragments **(servlet 3.0)**
- **EJBs in WAR files (Java EE 6)**
- Certain Java EE6 annotation **(@WebServlet (servlet 3.0), @ServletFilter (servlet 3.0), @Singleton (ejb 3.1))**

These constructs will most commonly show up as errors in the log and trace files as something similar to the following example:

```
com.ibm.websphere.management.exception.AdminException: ADMA5055E: Errors in validating application target
association for SampleApp -
ADMA0110E:  This type of J2EE application 50 is not supported on nodes [testNodeAppserver] of version [7]
  at com.ibm.ws.management.application.task.ValidateAppTask.performTask(ValidateAppTask.java:424)
  at com.ibm.ws.management.application.SchedulerImpl.run(SchedulerImpl.java:315)
  at java.lang.Thread.run(Thread.java:773)
```

Other key phrases that indicate an application was automatically promoted include:

- "Upgrade triggered by servlet 30 annotations"
- "Updating versions for Web Descriptor"
- "Upgrade triggered by EJB content"
- "Upgrade triggered by fragments"
- "ADMA5074W" or "has web modules with a version 3.0 deployment descriptor and EJB content"
- "descriptor will be promoted"

To get the above messages, you might need to enable these trace settings:

com.ibm.config.eclipse.wtp=finer:
com.ibm.ws.management.application.*=all:
com.ibm.websphere.management.application.*=all

Disabling Java EE 6 Constructs

A set of properties controls whether certain Java EE 6 constructs are ignored when the application is deployed or are disabled when the application is running. They can be used in solving the application deployment issues discussed previously. Note that deployment and runtime are independently controlled. For deployment these properties can be set at the process, application, or module level. Likewise, for runtime these properties can be set at the server, application, or module level.

- To set these properties at the module or application level, locate the corresponding **META-INF/MANIFEST.MF** file and add one or more of the following properties:

  Ignore-No-Interface-View: true
  Ignore-Web-Fragment: true
  Ignore-JEE6-Annotation: true
  Ignore-Ejb-In-War-Annotation: true

  ** Properties set at the application level apply to all modules in the application.
  ** Both deployment and runtime will use the application and module level settings.

- To set these properties for the deployment process, go to the dmgr profile's properties directory and locate the wsadmin.properties file and add one or more of the following properties:

  org.eclipse.jst.j2ee.commonarchivecore.ignore.no.interface.view=true
  org.eclipse.jst.j2ee.commonarchivecore.ignore.web.fragment=true
  org.eclipse.jst.j2ee.commonarchivecore.ignore.jee6.annotation=true
  org.eclipse.jst.j2ee.commonarchivecore.ignore.ejb.in.war=true

  ** Properties set at the process level apply to all applications being deployed.

- Use the administrative console to set one or more of the previously mentioned **org.eclipse.*** properties in the server's Java runtime process settings.
  ** Properties set at the process level apply to all applications running on the server.
  ** Deploying an application through the administrative console requires that one or more of the **org.eclipse.*** properties be set in the dmgr's Java runtime process settings.

Solutions:

Fixing the Application in the Current Release

The best solution to avoid application deployment issues is to ensure that all applications are properly assembled for the current targeted WebSphere release before you migrate. For example, deployment descriptors should indicate the correct version, code should not contain

unsupported annotations, and so on. This solution, however, requires coordinating with the development teams to re-test and re-deploy applications that are currently working. This solution is ideal because the constructs that cause deployment and runtime behavior changes are eliminated. Newer releases of WebSphere always support older levels of the Java EE technologies, so the application migrates forward “as-is” without the need for any special handling.

“All-at-once” Migration

A solution that avoids the single deployment issue of the application Java EE level not being supported by the WebSphere release of the target node is to migrate the entire cell all at once and to delay deploying any applications until after all nodes have been migrated to the new release. This strategy might require extensive downtime, especially if there are a lot of nodes to be migrated and apps to be installed. It also requires using WASPostUpgrade's “-includeApp script” option to separate the application install process from the configuration portion of the dmgr's migration. Be aware that this solution does not address possible Java EE 6 behavior changes during deployment or when the application is running on a server at the new release level. This is inconsistent with WebSphere migration's goal of “no behavior changes”. To address these concerns, see the discussion above on “Disabling Java EE 6 Constructs” for both deployment and runtime.

“Mixed-cell” Migration

The solution that supports a mixed-cell migration is to have the deployment process ignore these Java EE 6 constructs during the dmgr's migration. See the discussion above on “Disabling Java EE 6 Constructs”. This solution also requires using WASPostUpgrade's “-includeApp script” option to separate the application install process from the configuration portion of the dmgr's migration. Over time the other nodes will be migrated to the new release. As these nodes are migrated, applications containing Java EE 6 constructs might change behavior running at a newer level of WebSphere. See the discussion “Disabling Java EE 6 Constructs” above on how to handle these runtime behavior changes.

When you migrate profiles, keep the following in mind:

- WASPostUpgrade's “-installApp script” option generates an install script for each application migrated. These scripts are located in the migration backup directory. Each application is staged for deployment in the dmgr profile's installableApps directory. Instructions are provided in the install_all_apps.jy script.
- Applications containing deployment descriptors with versions belonging to Java EE 6 that were previously deployed to a federated node running a WebSphere release prior to V8.0 will not deploy when migrated to V8.0 or V8.5.5. WebSphere does not provide a special property to ignore deployment descriptor version settings. The application staged in the new dmgr's installableApps directory will have to be manually changed.
- If an application is not deployed on the dmgr after it is migrated and the old node is synchronized, whether automatically or manually, the application will be stopped and removed from the old node. Be sure that all nodeagents are stopped before you start the new dmgr.

Additional Help with Applications and Annotation Scanning:

How an application is coded, packaged, and assembled can make a difference in its deployment and runtime behavior. The following suggestions and articles can help you to work with applications.

Deployment Descriptors:

Some deployment descriptors support a “metadata-complete” flag. If set to true, that portion of the application will not be scanned for annotations. Consider using this flag if you know your application does not use annotations.

Shared Library Support:

Consider using shared libraries for utility type JAR files, especially if they are from a third party. Using shared libraries not only reduces the size of your application and the amount time spent scanning it, but it can also avoid the automatic “app-promotion” problems discussed earlier. Shared libraries are simply made available to applications as part of the class path. You can configure these libraries for visibility and scope. The drawback to this approach is that the applications are now dependent on the environment. The shared libraries need to be configured on each system the same way, which places more of a burden on the system administrator. For more information about using shared libraries, see *Best Practices for Integrating Open Source Software Frameworks* (https://www.ibm.com/support/pages/node/494997).

Use Annotation Filtering:

WebSphere provides files that list a set of archives that, if found, should not be scanned for annotations. These lists are found at the install, profile, application, and module level and are enabled by setting a property to point to its locations. For more information about managing application scanning from a runtime point of view, see *How to speed up annotation processing in WAS V7/V8* (http://wasdynacache.blogspot.com/2012/05/how-to-speed-up-annotation-processing.html).

WebSphere Application Migration Toolkit:

The WebSphere Application Migration Toolkit can also help to identify code issues for target WebSphere releases. For more information, see the documentation (https://www.ibm.com/support/pages/websphere-migration-knowledge-collection-downloads).