Part 1 (Basic Concepts): Automated Deployment of Enterprise Application (EAR) Updates

Barry Searle [mailto:searle@ca.ibm.com]
Architect, WebSphere Tools for Automated Build and Deployment
IBM Toronto Lab

Ellen Matheson McKay [mailto:ecmckay@ca.ibm.com]
WebSphere Information Developer
IBM Toronto Lab

© 2004 International Business Machines Corporation. All rights reserved.

Abstract

This two-part article discusses application deployment, particularly automated updates, to IBM® WebSphere® Application Server in a large-scale enterprise environment. It applies to WebSphere Application Server version 5.0, version 5.1, and version 6.0, and also includes an introduction to a few version 6.0 enhancements. This article is not intended to be used as a reference for all the details of WebSphere Application Server administration, but it does describe the key concepts used, and contains a list of references. Although the beginning of the article reviews some fairly basic base server and managed server concepts and operations, much of the remainder of the article will discuss certain complex concepts or operational considerations that will be new even to very experienced enterprise application server administrators.

This first part of the article discusses wsadmin deployment to base and managed servers. It discusses why phased deployments are needed to maintain applications in a WebSphere Application Server Network-Deployment managed cell, and how to maintain high availability in such an environment.

Part 2 of the article discusses pre- and post-deployment validation, and it discusses gradual deployment of incompatible versions of applications. It also discusses the design and implementation of a downloadable **`Automated Deployment`** example program that illustrates how to automate the deployment of randomly built collections of enterprise applications or updates, and how to automatically target those applications or updates to the correct servers, including stage-specific application setup.

The problem

It is quite easy to deploy (install) an application into a WebSphere Application Server setup – typically it just takes a one-line command. For your local running base server, it can be as simple as:

```
wsadmin -c "$AdminApp install X:/MyApp.ear" -lang jacl
```

However, this simplicity is deceiving, and the preceding example is really just the Hello World of deployment – a nice demo, but not typical of the real world.

In a real enterprise environment, there are hundreds of interrelated applications spread over dozens of remote application servers, and regular updates that need to be deployed to the right servers, all the while maintaining application availability to users. Even worse, most large enterprises have different sets of operating environments, or stages, each requiring different setups for the same application. For example, the security role mappings and the database used for a specific application in a Microsoft® Windows® integration stage are likely different from those used for that same application on a Linux® production server.

The result might be that at 3 a.m., when some random group of 20 applications have just been rebuilt because of an automated production build of Library Control System (LCS) code changes, those particular 20 applications

each need to have their updates deployed to their correct individual application servers somewhere in the enterprise. And, although it is 3 a.m. in North America, it is prime time elsewhere in the world, so the application updates need to be done in a way that maintains high application availability. This update build and deployment process is regularly repeated, each time involving a randomly different set of updated applications.

Command line wsadmin and JACL/Jython scripts

WebSphere Application Server has an extensive administration program, the Administration Console. It also has an equivalent command line tool, **wsadmin**, which can be run interactively or which accepts a file of scripted commands. For scripting, **wsadmin** supports the two script languages **JACL** and **jython**. See the reference section for links to related material. In the examples in this article, and in the downloadable example program, **JACL** is used since it runs on WebSphere Application Server version 5.0 and later (**jython** runs on WebSphere Application Server version 5.1 and above).

This article is not intended as a reference for WebSphere Application Server administration or the **wsadmin** tool. For more detailed information, consult the material in the reference section.

Local base server deployment

The simplest deployment scenario is to deploy an application from a build machine to a local running base server. (Network Deployment managed cells are discussed a little later in this article.) The local WebSphere Application Server installation has a **bin** directory containing the **wsadmin** command and other tools.

The typical sequence for an initial application installation is as follows:

1. If the local server is not already running, then start it:
   ```
   startServer server1
   ```
2. Install the application:
   ```
   wsadmin -c "$AdminApp install C:/MyApp.ear" -lang jacl
   ```
3. Optionally, list all current installed applications (to verify that it really was installed):
   ```
   wsadmin -c "$AdminApp list" -lang jacl
   ```
4. Save this new server configuration:
   ```
   wsadmin -c "$AdminConfig save" -lang jacl
   ```
5. Start the application (this is a one-line command):
   ```
   wsadmin -c "$AdminControl invoke
        [$AdminControl queryNames type=ApplicationManager,*]
        startApplication MyAppName" -lang jacl
   ```

The typical sequence for an uninstallation, using a script file instead of interactive commands, is as follows:

```
wsadmin –f uninstall.jacl –lang jacl
```

where the file **uninstall.jacl** contains the following lines:

```
$AdminApp list
set appMgr [$AdminControl queryNames type=ApplicationManager,*]
$AdminControl invoke $appMgr stopApplication MyAppName
$AdminApp uninstall MyAppName
$AdminApp list
$AdminConfig save
```

Note that the default **wsadmin** server connection type **SOAP** requires a running server. It is possible to connect using connection type **NONE**, but the available operations are restricted. Refer to **wsadmin** documentation for more details.

Remote base server deployment

A slightly more typical environment is one where one or more target base servers are on remote machines. The most obvious approach (and the one used by many customers) is to use File Transfer Protocol (**FTP**) to transfer the application or update from their build machine to the remote target machines, and then to use **telnet**, or some other similar program, to run the **wsadmin** program on those remote machines. This works, but it is quite messy. It requires **FTP** and **telnet** on each remote machine, and quite a few error-prone manual operations. It also requires **FTP** and **telnet** accounts on each remote machine, which introduces administration and security issues many companies would prefer to avoid.

WebSphere Application Server provides a nice solution to this scenario that many users are not even aware exists. If you install WebSphere Application Server on the build machine, then the WebSphere Application Server runtime and its **wsadmin** command tool are available, even if that installation server is never configured or started. You can use the **wsadmin** command to connect to your remote target servers and run your deployment commands against those remote servers. If a local build machine file is being installed on the remote server, WebSphere Application Server will internally do the file transfer for you. All the normal WebSphere Application Server security (if configured) is automatically used according to the **userid** and **password** used with the **wsadmin** tool. Thus, deployment to remote servers is as easy as to a local server; the only difference is that the **wsadmin** invocation command specifies a remote server instead of a local one:

```
Wsadmin -host MyRemoteHost -port MyRemotePort …commands…
```

Note: The above solution for remote base server deployment only works if both the remote and local installation (where **wsadmin** is running) are WebSphere version-6 installations. If either of the installations is a WebSphere version-5.0 or version-5.1 installation then you will get an error "*X:\MyTEMP\appnnnnn.ear does not exist for installation*". Base server (non-managed) **Remote File Transfer** support is a version-6 enhancement, unfortunately the version-5x error is not clear about that.

Thus, a very simple, but representative, base server organization might be similar to this:

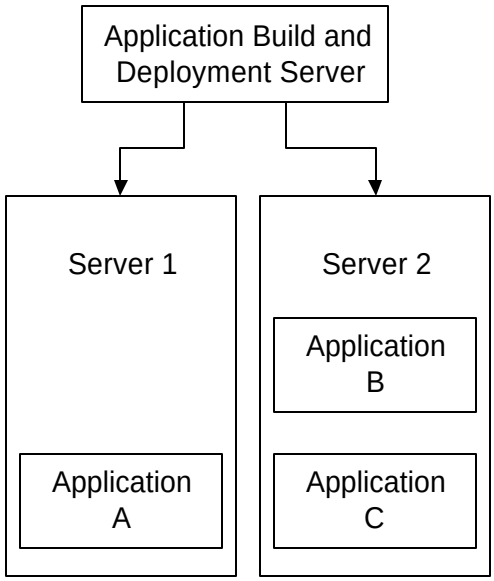


Figure-1: Base Servers

This approach provides a nice solution for a build machine and one or two remote base servers. But as the number of independent base servers increases, it rapidly becomes desirable to have centralized administration to manage the collection of servers as a single administrative cell.

Network deployment managed cells, nodes, and clusters

As the number of applications and the number of target servers grows, it becomes almost essential to provide single administrative control for these large collections.

A node is a physical collection of one or many application servers on a particular machine, and there is typically one Node Agent per node controlling those servers on that remote machine. Why should there be more than one server per machine? In addition to the obvious redundancy consideration, there are technical and performance reasons (such as Java threading limitations, Java memory garbage collection considerations, etc.) why splitting one larger server into multiple servers can provide significant performance improvements on that same machine. This is particularly common on very large and highly reliable server platforms. For details, see the WebSphere Application Server performance "*Best Practices*" documents (available online at http://www.ibm.com/developerworks/websphere/zones/bp/). Note that there can be two or more logically separate nodes (each with their own servers) on the same physical machine, but that is very unusual.

Why should there be more than one target machine? In addition to the obvious ability to supply additional resources (additional CPU cycles, physical memory, etc.) and to provide physical redundancy against hardware failure, there may be organizational or geographical requirements. As well, the various Quality Assurance (**QA**) environments, such as development, integration, test, pilot-production and production, are almost always run on different machines for logistic, administrative, and security reasons.

To simplify the setup and ongoing administration of redundant servers, most application server products introduce the concept of clusters. A *c*luster is a logical collection of servers, each of which typically contains the same set of applications. (Any one cluster member could actually contain additional or different programs, but that is highly unusual and confusing.) A vertical cluster contains multiple cluster members on the same node, while a horizontal cluster contains cluster members on different nodes (which is more typical).

A managed cell is an administrative collection of many servers. One of the key features of WebSphere Application Server Network Deployment is that groups of servers can be federated together into a single managed cell. A special server called the Deployment Manager (**DMgr**) manages all the servers and applications in the cell, using one Node Agent (**NA**) per node to control the one or more servers in that node. Node Agents are typically configured as an always-running daemon within their host machine operating system so they are always available. They can start or stop servers on that machine, install or uninstall applications on the servers, and can configure other server settings and control other server functions.

Thus, a very simple, but representative and scalable, cell organization might be similar to this:

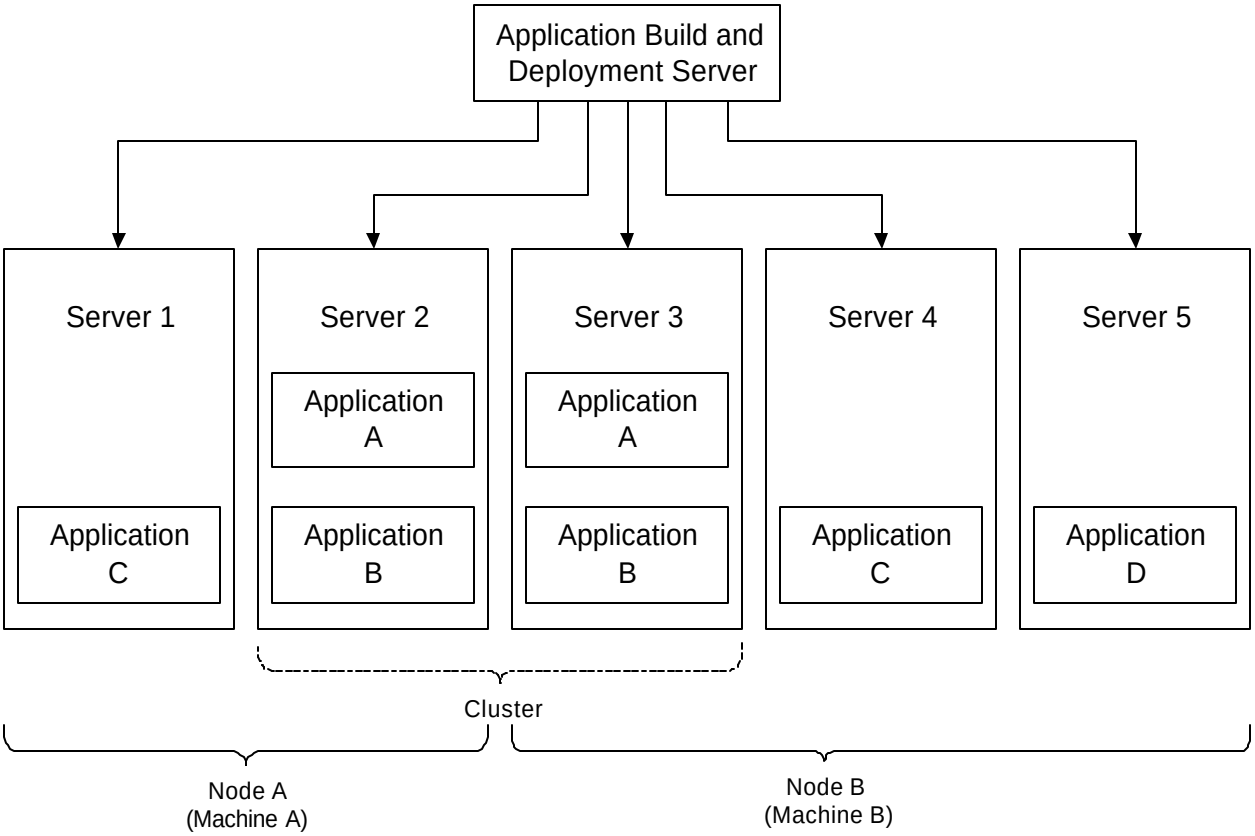


Figure-2: Cell Organization (Servers, Nodes, and Clusters)

Using wsadmin with managed servers (conntype=SOAP)

In the earlier discussion about using **`wsadmin`** to deploy to a local running base server, you probably assumed, correctly, that the **`wsadmin`** program connected to that server to perform its various administrative operations. We also could have started **`wsadmin`**, specifying the default connection parameter type of **`-conntype SOAP`** (or **`RMI`**). If the local server is not running, we can start **`wsadmin`** with **`–conntype NONE`** and then **`wsadmin`** will directly manipulate the local server configuration files and their contents. In this case, there is less functionality available, and some of the commands (like starting an application) are not available. In the case of remote base servers, we supplied a **`host`** and **`port`** parameter to specify which running server to connect to, so clearly it is not possible to use **`–conntype NONE`** for remote servers since the **`wsadmin`** program has no direct access to their (remote) configuration files.

In the case of managed servers (servers federated into a Network Deployment cell), it is not quite so obvious what is happening. If the local server is configured as part of a managed cell, then, by default the **`wsadmin`** program will connect to its (likely remote) Deployment Manager server and will ask that Deployment Manager to perform the specified administrative operations against the specified target servers. Again, the default connection type is **`-conntype SOAP.`** For managed servers, we can override the default destination and use the **`host`** and **`port`** parameters to connect to a Node Agent controlling one or more target servers, or to connect directly to a target server. Connecting to the Deployment Manager (the default option) gives the most functionality, to Node Agents a little less functionality, to servers even less functionality, and **`–conntype NONE`** to a local server gives the least functionality. It is highly recommended that you always connect to the Deployment Manager; otherwise, localized changes might be incompatible with the master cell configuration (such as installing an application with the same name as is already installed elsewhere in the cell) and could cause subsequent synchronization errors. For details, consult the WebSphere Application Server Information Center documentation.

How do you have a build machine deploy to remote production servers? Earlier we said that you could have WebSphere Application Server installed on a build machine, without requiring a server to be actually configured or running. Therefore, you can just start **`wsadmin`** using the **`host`** and **`port`** parameters to specify the remote Deployment Manager for the production cell, and everything works as expected. Note that the Deployment Manager must always be at the same version plus service level as all servers within the cell, or at a later version.

Note that WebSphere Application Server version 6.0 provides support for multiple server profiles (an expanded implementation of version 5.1 instances), where each server profile is essentially a totally independent server (independent configurations sharing a common set of runtime programs). The **`wsadmin`** command will operate against its default server profile (determined by the profile location you are executing from), but you can use the parameter **`–profileName MyProfile`** to override that default and specify the actual server profile (and hence its particular Deployment Manager) to be used, or you can use the **`host`** and **`port`** parameters to directly specify the destination (typically a Deployment Manager).

Web Servers, firewalls, redundancy, and workload management

Note that there are typically one or more Web servers in front of the set of application servers. The Web servers accept incoming user **HTTP** requests and route each request to an appropriate application server to perform the work. Of course, one or more levels of firewall protection are almost always present at different locations within the system.

Thus, a simple, but representative, **HTTP** organization might be similar to this:

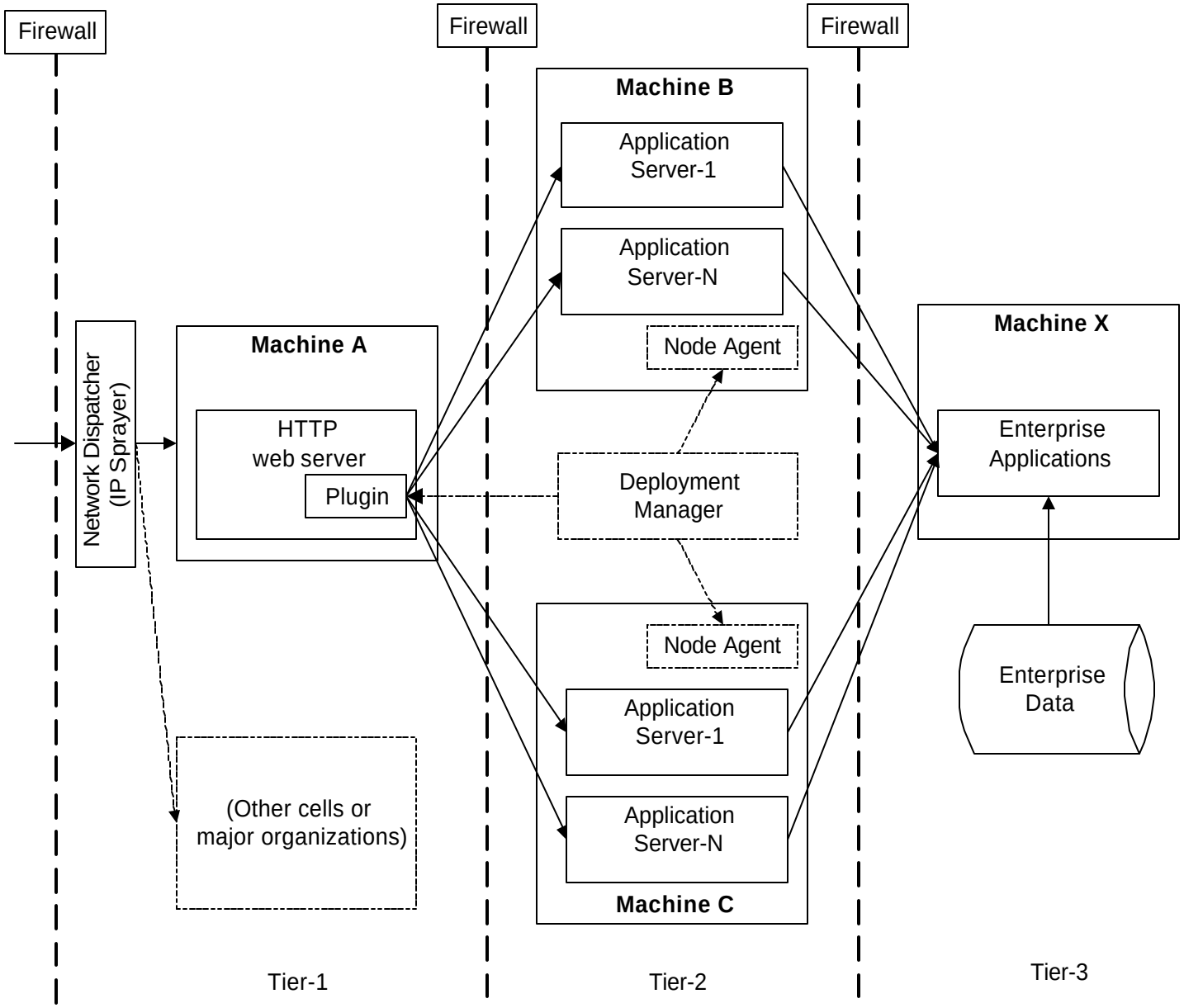


Figure-3: HTTP Organization

Good availability using redundancy and server-failover

Looking after the design, setup, and operation of redundant systems is a significant activity, and there are whole books devoted to this topic. For detailed information, consult the material in the reference section.

If two or more application servers are hosting the same application, then such redundancy can provide workload sharing (better throughput and response time for increasing numbers of requests). This redundancy can also be used to provide reactive server-failover and recovery in case of unexpected hardware or software failure. In case of an application server failure, WebSphere Application Server can be configured such that incoming Web server work requests are routed to a different server running that same application. All new requests are handled appropriately, but any in-progress HTTP session will be disrupted and must be re-initiated by users (it will appear to them that there was a brief service failure).

High availability using server-failover and session-recovery

WebSphere Application Server can reroute new work requests, but it can also be configured to provide reactive session-recovery to allow most in-progress requests to continue and complete on the redundant server without any visible interruption to end users. Thus, high application availability can be achieved in spite of unexpected system failures. For most users, even in-progress HTTP sessions appear to continue without any visible interruption.

Continuous availability using preemptive work rerouting (quiesce)

Even though many HTTP sessions can be recovered, some applications with very large session data or other session data that cannot be persisted, can cause an in-progress HTTP session to fail. Thus, if an application or system has a planned outage (due to maintenance, or other operator-initiated situations), then preemptive work rerouting can be used to provide nearly continuous availability.

The affected servers are first quiesced, which means that all new incoming work requests are routed instead to other redundant servers, and all in-progress HTTP sessions on the quiesced servers are temporarily allowed to run until completed. After a reasonable time, the quiesced servers are then stopped, and the planned maintenance or other activity is performed. After the planned activity is completed, the servers are re-activated. New incoming requests can again be routed to them (in addition to the other, still running, redundant servers).

Quiescing entire cells (as discussed a little later in the “gradual rollout” section) is typically done up front at the Network Dispatcher (IP Sprayer) level using products such as WebSphere Load Balancer. The Network Dispatcher must be configured with session affinity to ensure that all user requests from the same user HTTP session get routed to the same processing organization. Quiescing individual application servers is typically done at the HTTP Web server level, by manipulating the routing table in the **`plugin-cfg.xml`** file. This file is checked for changes by the HTTP server once per minute or at another specified frequency. The HTTP server must similarly be configured with session affinity. Note that WebSphere Application Server version 5.x could generate the **`plugin-cfg.xml`** files, but they needed to be manually transferred to the HTTP servers. WebSphere Application Server version 6 has a new feature to allow its HTTP servers to be federated into a cell, and to transfer the **`plugin-cfg.xml`** files.

AutoSync application updates throughout an enterprise cell

WebSphere Application Server has the ability to provide automated distribution of application updates to servers throughout a managed cell. The application update is first installed into the Deployment Manager application repository. If a Node Agent has its **`AutoSync`** feature enabled (which is the default setting), then that Node Agent will periodically (by default, every 60 seconds) perform a **`NodeSync`**, which asks the cell Deployment Manager for any application updates. Any such updates are then transferred to the Node Agent, and that Node Agent then updates all its affected servers (since it might have multiple servers that are configured to run that same application). Thus, if each Node Agent has the default **`AutoSync`** enabled, any application updated or installed into the Deployment Manager will be automatically distributed to every affected node, and to every affected server on those nodes, over a relatively short period of time.

Application updates can cause application availability failures

What some administrators fail to understand is that default application updates can cause application availability failures, even if their entire system is configured for **`server-failover`** and **`session-recovery`**. This is because while an application is being updated it is not available to process work requests, and any work routed to it from a Web server will not get a response. Even worse, any other interdependent application will have a service failure if it tries to use the unavailable application instance. This is because the Web servers and the workload management programs don't know that the application is temporarily unavailable. The solution is to first stop the affected application server before doing the application update (not just the application, the complete application server). The stopped application server will be detected, and **`server-failover`** and **`session-recovery`** will then take place as expected.

Note that WebSphere Application Server version 6.0 has the improved ability to perform incremental in-place updates of application components, which will help improve application availability. However, certain types of application component updates will still require stopping and restarting the application, and hence can still result in application availability failures unless the servers are first stopped and **`server-failover`** and **`session-recovery`** is active, or unless preemptive work using quiesce and rerouting is performed.

Remember the earlier discussion of the **`AutoSync`** feature to automatically distribute application updates to all affected nodes in a cell? There is an availability issue that needs to be understood and handled. Each of the affected applications will be unavailable on their individual servers during the update. Even worse, if several Node Agents happen to request their **`NodeSync`** at the same time or at overlapping times, then the affected application on each of those affected nodes will be transferred and updated at essentially the same time, resulting in application unavailability on multiple nodes simultaneously! Where application availability is a serious concern, the solution is to specifically control the phased distribution of updates throughout the enterprise cell.

Better: Phased distribution of a single (compatible) application update

As previously mentioned, a specifically controlled phased distribution of an application update (compatible with previous versions) throughout a cell will minimize application availability issues. The required sequence is as follows:

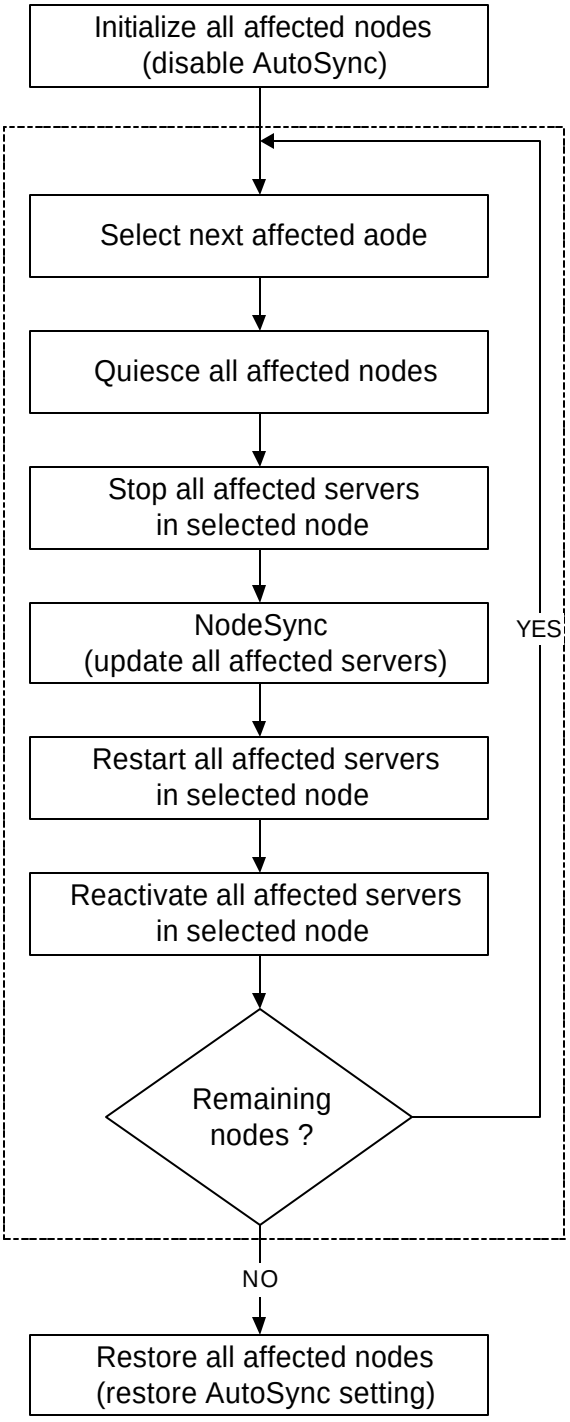


Figure-4: Phased Deployment

1. Save the current setting and then disable **`AutoSync`** on all affected nodes:
    a. Can also optionally save and disable **`SyncOnStartup`**.
2. Sequentially, for each affected node, phase-deploy the update:
    a. Select the next affected node to be updated.
    b. Optionally quiesce all its affected servers (reroute new work requests).
    c. Stop all its affected servers.
    d. **`NodeSync`** that node to retrieve the update and to install it in each affected server.
        - *Note:* Wait to ensure the EAR expansion is complete.
    e. Restart the affected servers.
        - *Note:* Test and wait to ensure the server is running.
    f. Optionally reactivate the affected servers (to process new work requests).
    g. Optionally validate installed application operation.
    h. Optionally request manual confirmation to accept (or reject and restore) this update.
    i. Repeat for the next affected node.
2. Restore the previous **`AutoSync`** settings for all affected nodes:
    a. Including **`SyncOnStartup`** if it was optionally disabled.

The above process involves a lot of manual operations and is very error-prone. A script can be created to perform the specific steps, but a different script is required for each different application and each application's

different set of associated target servers. The scripts can be quite complex and difficult to create, and require ongoing maintenance as the environment changes.

There are two special notes in the above steps. First, after performing the **`NodeSync`**, the application update (EAR) has been distributed down to the node, but the EAR file must still be expanded into the server **`installed application directory`**. Until this EAR expansion is complete, attempting to start the server will produce indeterminate results. An IBM Problem Report has been created about this, and there may, in the future, be a downloadable WebSphere Application Server Interim Fix to allow scripts to test for the completion of the EAR expansion. Second, after returning from the **`wsadmin startServer`** command, the command has been processed by the Node Agent but the actual server startup may not yet be complete. Scripts need to test that the server has completed startup and is running.

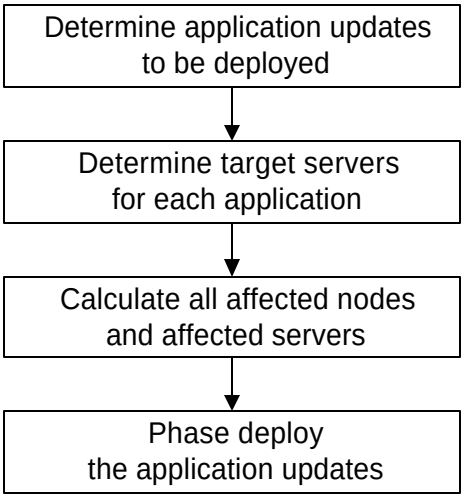
Note that WebSphere Application Server version 6.0 has a new cluster-update feature to distribute (in phases) an update to each of a cluster's members one node at a time, including the stopping and restarting of affected servers being updated. Where **`server-failure`** and **`session-recovery`** have been configured and provide sufficient functionality, this will provide very good high availability during cluster updates. This feature currently cannot be used to phase distribute an update to unclustered servers, and does not do preemptive quiescing or reactivation of work rerouting. If multiple cluster members are on the same node (vertical clustering for scalability), then they will be simultaneously updated (and simultaneously unavailable) as if the Node Agent had performed a regular **`NodeSync`**.

Also WebSphere Application Server version 6.0 cluster update can only be done for one application at a time. If multiple applications need to be updated within a cluster, then multiple **`cluster-update`** operations must be individually performed, and each cluster member node and its affected servers will be cycled again during each individual **`cluster-update`**.

In summary, the WebSphere Application Server version 6.0 new **`cluster-update`** feature is a convenient way to distribute a single update throughout a horizontal cluster with cluster members across multiple nodes, and does so while maintaining high-availability.

Best: Phased distribution of multiple (compatible) application updates

As just mentioned, a specifically controlled phased distribution of an individual application update helps maximize availability. However, if multiple application updates are required, with each application update targeted to a potentially different, but overlapping, set of enterprise servers, then the process is slightly more complex, even though all the same basic principles apply. The sequence required is as follows:



1. Analyze the set of all current updates to be deployed.
2. For each application, determine (read) its specific set of target nodes and servers.
3. From the total set of affected nodes and servers, calculate the subset of unique affected nodes and unique affected servers.
4. Perform the previously described **`phased distribution`** for each affected node.

Figure-5: Deploying Multiple Updates

As for the phased deployment of single applications, the above process involves a lot of manual operations and is very error-prone. Additional scripts can be created to assist in the above steps, but with ever-increasing numbers of different applications, target servers, and stage environment and applications settings, the number and complexity of specialized scripts becomes an enormous challenge to create and then to maintain.

Part-2: Deployment validation, gradual deployment, automated deployment, and references

This part 1 has covered the basics of deployment of enterprise application (EAR) updates throughout an enterprise cell. Part 2 covers pre- and post-deployment validation and gradual deployment (of incompatible versions of applications), describes the Automated Deployment example program in detail, and includes an extensive set of references.

About the authors

Barry Searle is the Architect for WebSphere Tools for Automated Build and Deployment. He is a Professional Engineer who has worked at the IBM Canada Lab for over fifteen years on various application development tools. Prior to that he has many years of industry experience developing command and control systems and leading complex communications development projects. You can reach Barry at searle@ca.ibm.com

Ellen Matheson McKay is an Information Developer for IBM Canada Ltd. She writes online help and publications for Rational Application Developer. You can reach Ellen at ecmckay@ca.ibm.com

Trademarks

IBM and WebSphere are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.