

Managing Change Across Complex WebSphere Enterprise Environments

David Sayers
Richard Bettison

Objective

- Environment provision
- Introduce change into environments
- Consistent process for code deployments
- Comparison between environments
- Comparing and environment overtime
- Who, what, when?
- Self contained “super archive” that contains all code, scripts and configuration to provision an entire environment
- Efficiencies through self-service
- Speeding up the software development life-cycle
- Start to view middleware components more as commodities

Scope of seminar

- Version Control System
 - Development area
 - Release area
- Common approach to versioning
- Build Process
- Build Package (input to deployment process)
- Environment Provisioning and Deployment Process

Version Control – Development area

- Development area contains all the source code
- Developers and administration team have access to source
- Application build.xml reside here

Sample Development area

Salesapp_dev\

 \applicationEARs

 \database

 \source

Version Control – Release Area

- Release area contains all the scripts and properties files used for deployment.
- Only administration team have edit access. Developers can view content.
- Shared resources are symlinks (in ClearCase / svn) or IVY Repository if using ANT or in POM.xml if Maven2

Sample Release area

Saleapp_release

\j2ee

\html

\mq

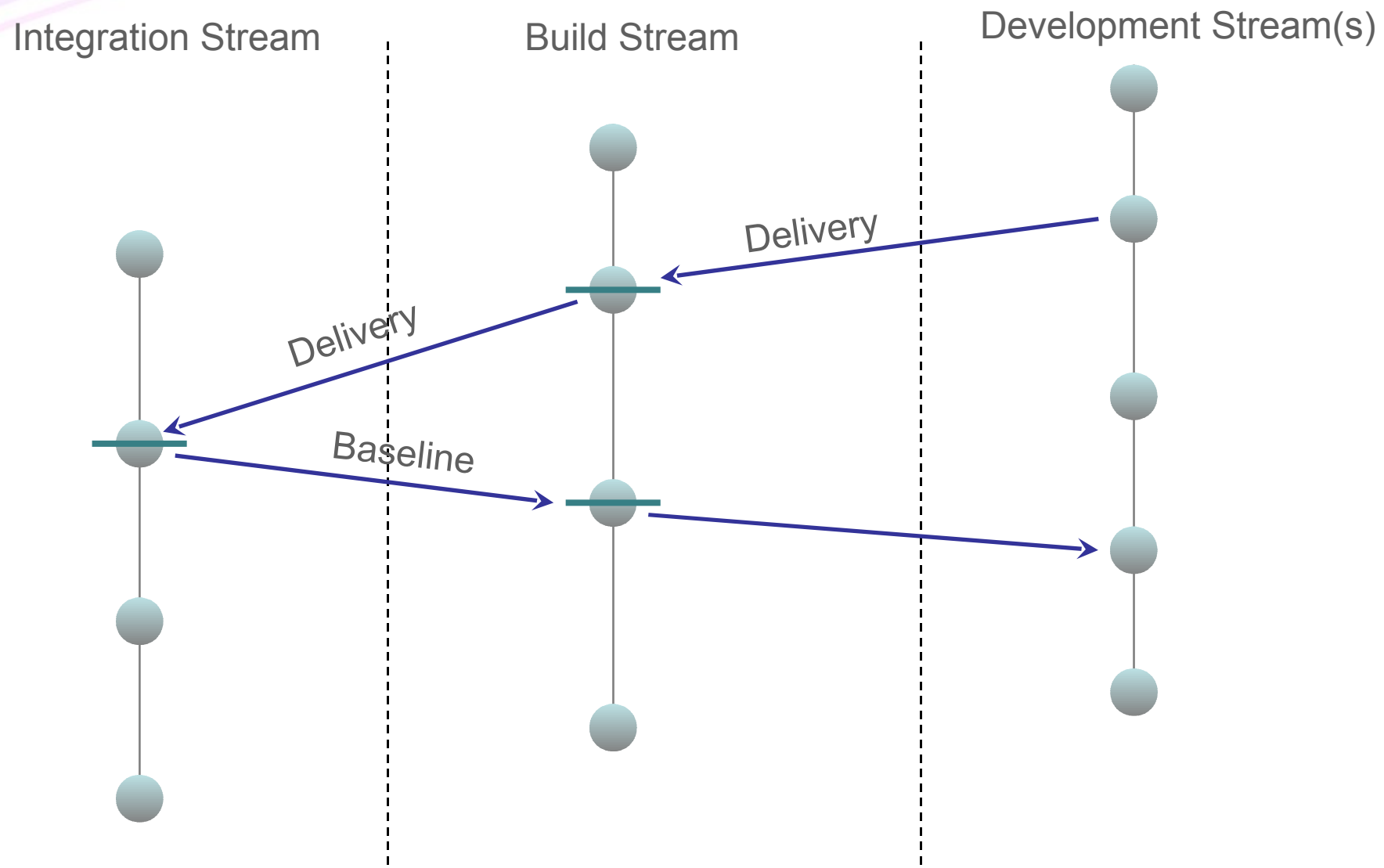
\scripts

\clients

How does the VCS relate to the build process?

- Build process is inextricably linked to standards in place in version control system
- Developer work on the team stream(s)
- Developer delivers to the build stream
- Build initially on build stream. Successful builds are delivered to the integration stream
- Re-built on integration stream
- Ensures only successful builds are done on the integration stream
- Build package is the output from Integration Stream
- Integration stream is baselined
- Baseline is then recommended

Sample Development Area



Build Process #1

Generic build framework of re-usable components to perform all common tasks:

- buildJar
- buildWar
- buildEjb
- buildPortlet
- buildBusinessProcess
- buildEar

Build Process #2

- Application teams create and application specific build.xml file that calls generic build targets
- Example xml
 - buildJar
 - buildJar
 - buildWar
 - buildEar

Build a JAR file

```
<target name="doCustomerBSJava_Project" depends="init,
generateFoundationJXS" description="Builds all the java code">
  <ant antfile="${GENERICBUILD}" target="buildJar">
    <property name="project" value="${project}"/>
    <property name="classpath" value="${classpath}"/>
    <property name="component" value="CustomerBS"/>
    <property name="sourceDir" value="$
{dir.java.src}/CustomerBS/src"/>
    <property name="workingDir" value="${workingDir}"/>
    <property name="workingDir.lib" value="$
{workingDir}/lib/${project.staging}/lib"/>
  </ant>
</target>
```

Build a WAR file

```
<target name="buildWar_ProjectPortal" depends="init" description="Build
  Project Portal War file">
  <ant antfile="{GENERICBUILD}" target="buildWar">
    <property name="war.name" value="ProjectPortalWeb.war"/>
    <property name="component" value="ProjectPortalWeb"/>
    <property name="sourceDir" value="$
{dir.web.src}/ProjectPortalWeb/src"/>
    <property name="include" value="**/*"/>
    <property name="sourceDir.meta" value="$
{dir.web.src}/ProjectPortalWeb/WebContent"/>
    <property name="project" value="{project}"/>
    <property name="classpath" value="{classpath}:$
{classpathWAR}"/>
    <property name="workingDir" value="{workingDir}"/>
  </ant>
</target>
```

Build an EAR file

```
<ant antfile="${GENERICBUILD}" target="buildEar">
    <property name="sourceDir" value="${dir.ear.src}/$
{application.name}" />
    <property name="sourceDir.jx" value="${dir.ear.src}/$
{application.name}" />
    <property name="include" value="" />
    <property name="excludeJars" value="" />
    <property name="sourceDir.meta" value="${dir.ear.src}/$
{application.name}/META-INF" />
    <property name="workingDir.meta" value="${workingDir.dist}/
META-INF" />
    <property name="project" value="${project}" />
    <property name="component" value="projectapplication" />
    <property name="classpath" value="${classpath}" />
    <property name="workingDir" value="${workingDir}" />
</ant>
```

Build Process – benefits #1

- Can change the implementation of Generic Build and ALL application teams start using this without having to make any changes
- Can add additional components to Generic Build process that are immediately available to all application teams
 - E.g.: Recently added Agitar for code coverage.
- Enforce standard use of libraries: log4j, jms, oracle, etc.

Build Process – benefits #2

- Application teams can still implement custom build components if required
- Output of the build process is a the deployment “Build Package”
- Optionally developers can own the build process and abide by a contract to provide Build Package in a specific format

Versioning

- Baseline / label in version control system should be the only version required
- Standard approach
 - MAJOR.MINOR.BUILD
 - e.g. 03.02.001
- Common approach to versioning allows you think your version control system , deployment and runtime infrastructure

Versioning

Version used as an identifier in:

- Version Control System
- MANIFEST.MF
- Build package (super archive containing code and deployment scripts / properties)
- Deployment tool
- Use of J2EE standard tags in MANIFEST (Application-Version) the version is displayed in SystemOut

Audit

Typically audit is carried out backwards – starting at runtime environment

- Check version of deployed code in SystemOut
- Check deployment audit logs to see who / when deployment happened
- May also want to code has been promoted through the environments correctly
- Finally, may also want to perform some analysis on version control system i.e. what is the difference between this version of code and a previous one

Example Audit

MidVision DeployUI - Main x

http://localhost:8080/rapiddeploy1/main.zul

Customize Links Work Other bookmarks

MidVision RapidDeployUI - Main Richard Bettison Logged On

Deploy ClearCase Security Administration User Reports Help

Current Deployment Queue **Previous Deployments**

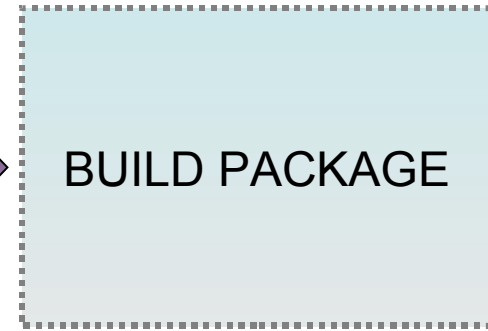
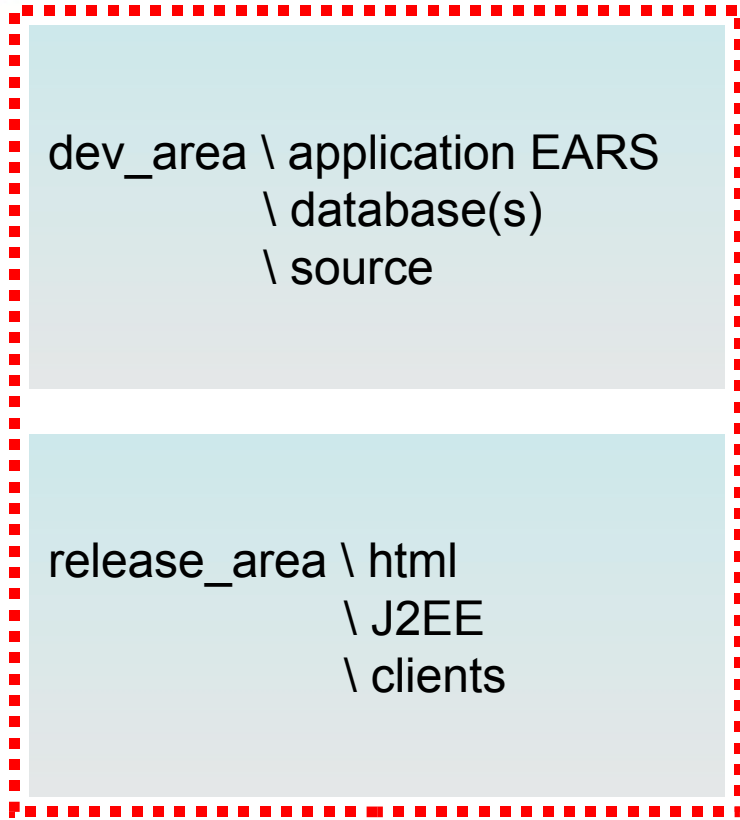
Search: ct000235

Node	Cell	Cluster	Application	Current Status	User	Deployment Date	Version
p09427gens27	wasgp05	GALAXYServerIntTst01	galaxyapplication	COMPLETED	ct000235	05-Jan-2009 07:29:48	GALAXY_DEV_J2EE_BUILD_1_0_24
p09427gens27	wasgp05	GALAXYServerIntTst01	galaxyapplication	COMPLETED	ct000235	06-Jan-2009 10:46:13	GALAXY_DEV_J2EE_BUILD_1_0_24
p09427gens27	wasgp05_111	GALAXYServerIntTst02	galaxyapplication	COMPLETED	ct000235	06-Jan-2009 16:10:41	GALAXY_DEV_J2EE_BUILD_1_0_24
p09427gens27	wasgp05	GALAXYServerIntTst01	galaxyapplication	COMPLETED	ct000235	07-Jan-2009 20:09:02	GALAXY_DEV_J2EE_BUILD_1_0_24
p09427gens27	wasgp05	GALAXYServerIntTst01	galaxyapplication	COMPLETED	ct000235	08-Jan-2009 08:52:32	GALAXY_DEV_J2EE_BUILD_1_0_24
p09427gens27	wasgp05	GALAXYServerIntTst01	galaxyapplication	COMPLETED	ct000235	08-Jan-2009 11:14:06	GALAXY_DEV_J2EE_BUILD_1_0_24
p09427gens27	wasgp05_111	GALAXYServerIntTst02	galaxyapplication	COMPLETED	ct000235	08-Jan-2009 11:19:07	GALAXY_DEV_J2EE_BUILD_1_0_24
p09427gens27	wasgp05	GALAXYServerIntTst01	galaxyapplication	COMPLETED	ct000235	08-Jan-2009 13:37:03	GALAXY_DEV_J2EE_BUILD_1_0_24
p17025dtg017	wasgp61_08	GALAXYServerPerfTst01	galaxyapplication	COMPLETED	ct000235	08-Jan-2009 13:37:03	GALAXY_DEV_J2EE_BUILD_1_0_24
p09427gens27	wasgp05	GALAXYServerIntTst02	galaxyapplication	COMPLETED	ct000235	12-Jan-2009 08:33:21	GALAXY_DEV_J2EE_BUILD_1_0_24
p09427gens27	wasgp05	GALAXYServerIntTst01	galaxyapplication	COMPLETED	ct000235	15-Jan-2009 15:27:03	GALAXY_DEV_J2EE_BUILD_1_0_24
p09427gens27	wasgp05_111	GALAXYServerIntTst02	galaxyapplication	COMPLETED	ct000235	18-Jan-2009 20:02:37	GALAXY_DEV_J2EE_BUILD_1_0_24

Build Package

- So far we have focused on standard java / j2ee modules
- Build Package is a superset of multiple j2ee modules along with ALL the scripts and properties that allow you to provision, configure and deploy to specific target environments
- Build Package is the output from an application build.xml plus the contents of the release area
- The combined package is base-lined, zipped or tarred (depending on target) to provide a self-contained archive that will provision an entire runtime environment (clusters, web server, virtualhosts, datasource, QCF, Queue destination, QMGR's, Queues, Topics, etc.) and installs code

Creating the Build Package

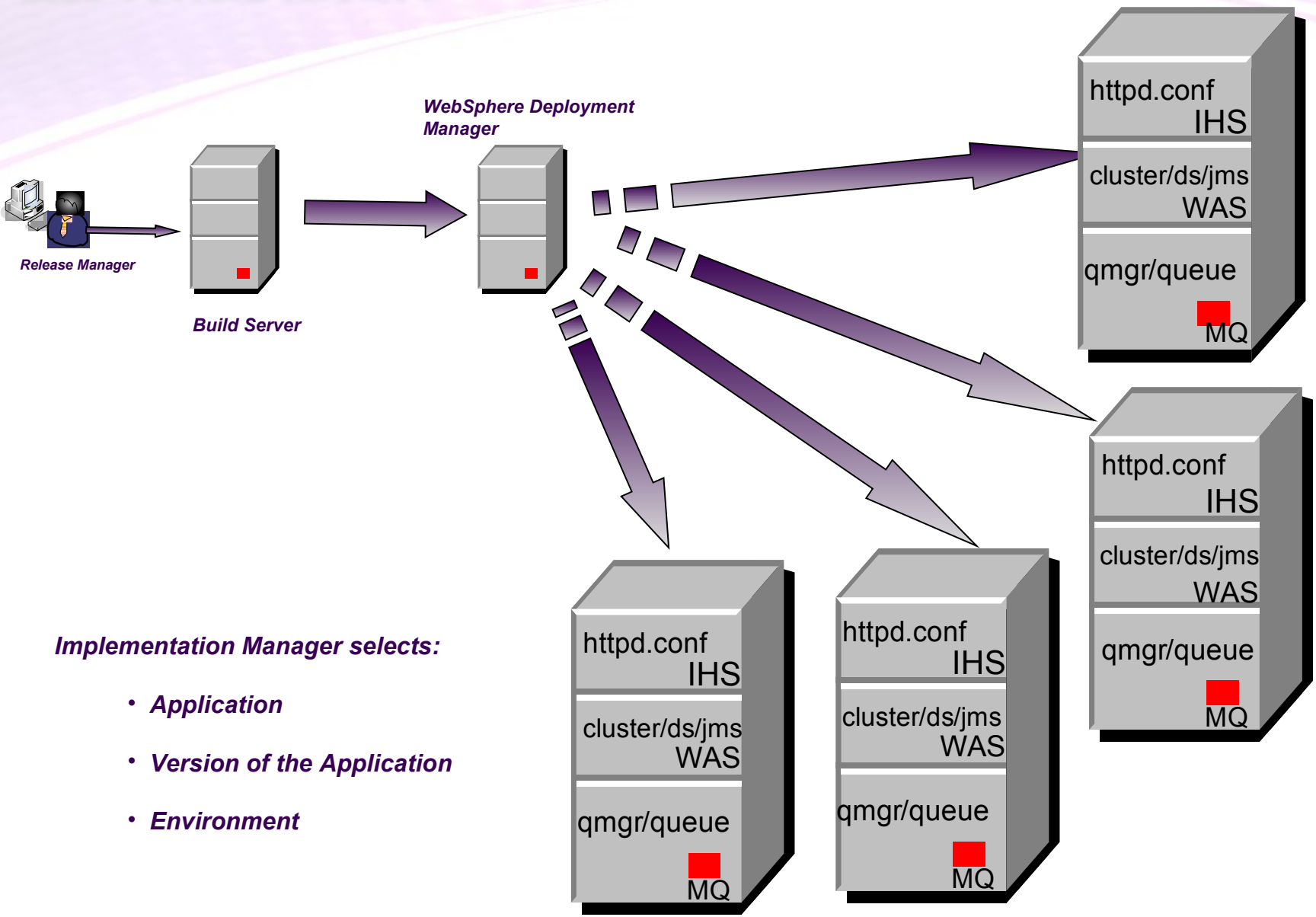


SALESAPP_BUILD_01_02_003.tar

Baseline File: SALESAPP_BUILD_01_02_003

Deployment Process

- The input to the deployment process is the build package
- Self-contained archive that can be deployed to any target environment. Only pre-requisite is a binary install of target runtime (DM, nodeagents, MQ, IHS, etc.)
- Build Package is pushed to local DM, unpacked and installed using the scripts, utilities and properties contained in the Build Package (no scripts are required locally)



Implementation Manager selects:

- **Application**
- **Version of the Application**
- **Environment**

Example deployment tool

The screenshot shows a web browser window with the URL `http://localhost:8080/rapiddeploy1/deployment_selection.zul`. The page title is "MidVision RapidDeployUI...". The browser's address bar shows the URL and navigation icons. Below the browser window, the application interface is visible. It has a header "Deployment Selection" with the user "Richard Bettison Logged On". A navigation menu includes "Deploy", "ClearCase", "Security", "Administration", "User", "Reports", and "Help". A form field "Select project for this deployment:" has "Galaxy_Dev_RMB" selected. Below this are tabs for "Selection", "Confirmation" (which is active), and "Status". The "Confirmation" tab displays a table with the following data:

Application	Version	Node	Cell	Cluster
galaxyapplication	GALAXY_DEV_J2EE_BUILD_1_0_27	midvision	wasgp61_07	GALAXYServerPerfTst01
galaxyapplication	GALAXY_DEV_J2EE_BUILD_1_0_29	midvision	wasgp61_08	GALAXYServerPerfTst02
galaxyapplication	GALAXY_DEV_J2EE_BUILD_1_0_23	p17025dtg017	wasgp61_07	GALAXYServerPerfTst01

At the bottom of the page, there is a "Begin Deployment" button.

Deploy Process is Application Centric

- Application central point for all configuration (clusters, datasource, qcf's, QMGR's, Queues, Web Servers, etc.)
- Each application has a single or group of properties file for each technology it installs
- Binary runtimes need to be installed. ALL other configuration is encapsulated in Build Package
- Need to cater for share components, such as MQ, cell scope resources i.e.URL Providers, etc.

Deploy Process

Install is split into three parts:

➤ Pre-install

- Set flag file on web server to bring service offline
- After application has quiesced stop middleware components

➤ Install

- Load reference data
- Configure WebSphere Application Server and install EAR
- Deploy HTTP configuration and static content
- Deployment MQ config. – QMGR, Queues, Topics, etc.

➤ Post install

- Start middleware components
- Running post-install unit tests
- Remove LB flags on web server

Deploy Process - Properties files

deplomentNodeHost.CellName.ClusterName.Application.**py**

deplomentNodeHost.CellName.ClusterName.Application.**mq**

deplomentNodeHost.CellName.ClusterName.Application.**props**

deplomentNodeHost.CellName.ClusterName.Application.**ibmihs**

deplomentNodeHost.CellName.ClusterName.Application.**WebServer1.webconf**

deplomentNodeHost.CellName.ClusterName.Application.**WebServer2.webconf**

deplomentNodeHost.CellName.ClusterName.Application.**clients** (J2EE Client)

deplomentNodeHost.CellName.ClusterName.Application.**dtd**

deplomentNodeHost.CellName.ClusterName.Application.**cdm**

Deploy Process – properties sample #1

```

-----
# General Variables
#-----
appName                = "SalesApplication"
appVersion              = "1"
envIdentifier           = SysInt01
portIdentifier          = 190

#-----
# Application Server Related Variables
#-----
serverName              = "SalesCluster" + envIdentifier
serverNode              = "PrimaryAppServerNode"
cookieName              = "JSESSIONID"
threadPoolMinSize      = 10
threadPoolMaxSize      = 50
minJVMHeapSize         = 512
maxJVMHeapSize         = 1024

# Note: This need to match the node name given during installation of Deployment Manager.
sslTransportSettingNodeName = nodeName
# Can be removed at a later point of time. Is not used at this moment #

# The state of the cluster or appserver after a restart of WebSphere
nodeRestartState       = "running"
sleepForInstallDuration = 600
restartServerAfterInstall = "true"
forceSleepForInstall    = "false"

# Logging related settings
maxLogFileCount        = 5
logRolloverSize        = 2
    
```

Deploy Process – properties sample #2

Server security settings

```
asSecurityEnabled = "false"
asSecurityAppEnabled = "false"
```

Transaction settings

```
asTransactionLifetimeTimeout = 5
asTransactionClientInactivityTimeout = 5
```

```
WC_defaulthost_port = 10190
WC_defaulthost_secure_port = 11190
BOOTSTRAP_ADDRESS_port = 12190
SOAP_CONNECTOR_ADDRESS_port = 13190
SIB_ENDPOINT_ADDRESS_port = 14190
SIB_ENDPOINT_SECURE_ADDRESS_port = 15190
SIB_MQ_ENDPOINT_ADDRESS_port = 16190
SIB_MQ_ENDPOINT_SECURE_ADDRESS_port = 17190
SIP_DEFAULTHOST_port = 18190
SIP_DEFAULTHOST_SECURE_port = 19190
```

#-----

```
httpServerNosecureTransportPortNo = 80
httpsCSSSprayerSecureTransportPort = 443
```

Cascading properties

```
wsadmin.sh -f genericWASFunctions.py -profile  
profileDefaults.py -profile project.py -profile  
deplomentNodeHost.CellName.ClusterName.Application.py
```

Wsadmin Taskinfo function

- Wsadmin function to describe resource mappings in ear file:
- Example: `AdminApp.taskInfo(earLocation, "MapResRefToEJB")`
- Abstracted to script to display all mappings
- Jacl format: `displayMappings.sh -displayJaclMappings <ear file>`
- Jython format: `displayMappings.sh – displayJythonMappings <ear file>`

Resource reference mapping 1 – Map EJB references to their resources

Extract from Python properties file showing mapping EJB references to resources via JNDI. [this is also output format from wasadmin.sh –displayJythonMappngs

```
#####  
# Resource 1: Map Resource References to EJB Resources  
#####  
#  
# Fields:- Module:EJB:URI:Resource Reference:Resource type:Target Resource JNDI  
# Name>Login configuration name:Properties:  
#  
res1_1 = ["ProjectPortalWeb", "", "ProjectPortalWeb.war,WEB-INF/web.xml",  
"dmap/UIConfigCache", "com.ibm.websphere.cache.DistributedMap",  
"cache/ui_config_IBANK_IntTst02", "", ""]  
res1_2 = ["ProjectPortalWeb", "", "ProjectPortalWeb.war,WEB-INF/web.xml",  
"FinancialTransactionProcessingWS", "java.net.URL",  
"url/FinancialTransactionProcessingWS_IBANK_IntTst02", "", ""]  
res1_3 = ["ProjectPortalWeb", "", "ProjectPortalWeb.war,WEB-INF/web.xml",  
"AccountWS", "java.net.URL", "jdbc/Account_IBANK_IntTst02", "", ""]  
...
```

Resource reference mapping 2 – Define resources

Python properties file extract showing two URL provider definitions.

```
#-----  
# URL Provider 1 and URL Resource Related Variables  
#-----  
urlpName_1 = "urlProvider_IBANK_IntTst02"  
urlpStreamHandlerClassName_1 = "unused"  
urlpProtocol_1 = "unused"  
urlpScope_1 = "cell"  
urlName_1 = "FinancialTransactionProcessingWS_IBANK_IntTst02"  
urlJNDIName_1 = "url/"+urlName_1  
urlDestination_1 = "http://10.200.142.55:97/PaymentProcessingServiceSO"  
  
#-----  
# URL Provider 2 and URL Resource Related Variables  
#-----  
urlpName_2 = "urlProvider_IBANK_IntTst02"  
urlpStreamHandlerClassName_2 = "unused"  
urlpProtocol_2 = "unused"  
urlpScope_2 = "cell"  
urlName_2 = "Account_IBANK_IntTst02"  
urlJNDIName_2 = "url/"+urlName_2  
urlDestination_2 = "http://10.160.74.63:96/Account"
```

Environment Comparison / Cloning

- Comparison between environments
- Comparison of the same environment over time
- Provision environments from templates
- Environment cloning:
 - Deployment Manager host and CellName
 - Environment identifier – SysTst01, IntTst01, etc.
 - Ports, ClusterName
 - Backend resources – JDBC, JMS, URL Providers
 - Memory, Pool sizes, Number of AppServers, etc.

Challenges

- Tools need to be comprehensive i.e. WebShere deployment process needs to cater for ALL your environments requirements
- Up front investment required
- Maintenance can be expensive
- New versions and products need to be incorporated quickly
- Can become reliant on a small number of individuals

Methodology #1

- Encapsulation of code, scripts / utilities and properties into a single zip or tar
- Can be applied across a broad range of technologies
 - WebSphere using jython
 - MQ using mqsc
 - IBM IHS using shell
 - Portal using jython and XMLAccess
 - WebSphere Datapower using xmi

Methodology #2

Can be applied to many third party applications

- Chordiant
- BusinessObjects
- Group1 Doc1
- PegaRules

Bringing it all together

- Weekly rebuilds of test environments
- Code and configuration baselined together
- rollback will revert Code and configuration
- Disaster recovery
- Re-building / migrations
- Environment comparison i.e. dev1 and dev2
- Comparing environments over time

Bringing it all together

- Code and configuration baselined together
- Environments provisioned if they don't exist
- Configuration introduced into environments
- Rollback will revert Code and configuration
- Disaster recovery
- Re-building / migrations / cloning
- Environment comparison i.e. dev1 and dev2
- Comparing environments over time
- Weekly rebuilds of test environments
- Full audit of code and configuration changes

