IBM WebSphere Exam #252 Study Guide
WebSphere Application Server Network Deployment V6.0, Core Administration

Kevin J. Ackerman

Wednesday February 1, 2007

Table of Contents

Section 1 – Architecture (11%)

Discuss the relationships between IBM WebSphere Application Server, V6.0 and the application components (e.g., browser, HTTP server, plug-in, firewall, database servers, WebSphere MQ, load balancing, ip spraying, etc.)

Edge Components

WebSphere Application Server Network Deployment addresses the needs of highly available, high volume environments with the inclusion of sophisticated load balancing, caching and centralized security capabilities based on Edge Components, known as WebSphere Edge Server in earlier releases.

Features and benefits

- Load Balancing. The load-balancing component in WebSphere Application Server Network Deployment provides a scalable solution for distributing and routing HTTP, servlet, and Enterprise JavaBeanâ„¢ (EJB) requests. As the load on one server or a cluster of servers with similar content increases, the load balancer can redirect this incoming traffic to underused servers to help maintain optimal response times for each site visitor. Incoming user requests are routed to back-end servers depending on their availability, performance and on the relevance of the application or components they host.

- Custom Advisors. Custom advisors can be used to load-balance requests based on unique application and platform criteria. To ensure ever-changing, accurate traffic allocation to back-end servers, an advisor can be deployed to be as high level as periodically determining the overall status of the servers, or as granular as checking specific application response times on the servers. Once the server health is determined, the advisor informs the load balancer "manager" function, which then sets weights for the servers to determine which server should receive new session or application requests. Through advisor code, traffic is appropriately routed to the optimal back-end server.

- Consultants. To extend the load balancing capabilities beyond purely a WebSphere Application Server environment, consultant code can be used to optimize server performance within a Cisco or Nortel infrastructure. Consultants generate server weighting metrics and distributes them to Cisco CSS 11000 switches or Nortel Alteon 180 series of switches for optimal server selection, load balancing and fault tolerance.

- Enhanced Caching. The edge-of-network caching capability in WebSphere Application Server Network Deployment, improves response time by offloading back-end servers and peering links. And, in contrast to other caching proxies that can cache static content, the edge proxy server can also cache - and invalidate â€“ dynamically-generated content from the WebSphere Application Server, such as JSPâ„¢Â and servlet results to create a virtual extension of the application server cache into network-based caches â€“ or to caches in the Akamai network through the implementation of Edge Side Includes (ESI) technology. The caching proxy also provides the following plug-in support:
- A plug-in that allows users to exploit an LDAP-based repository for storing user authentication and authorization information.
- An authentication/authorization plug-in that allows independent software vendors (ISVs) to exploit third party authentication/authorization mechanisms such as RADIUS or SecurID tokens.

- Edge Side Includes (ESI) support. ESI is a simple mark-up language and proposed standard for the dynamic assembly of Web page fragments, such as stock quotes and individual catalog prices. By leveraging ESI technology, dynamic content caching is extended by moving fragments from the Application Server to a proxy server that resides in the network - such as Akamai's. This enables caching to occur at a more granular level, as well as allowing companies to position page composition at the most optimal location, closer to the end user. As a result, companies can improve user experiences through expedited, personalized page composition, and reduce workload on the network servers due to fragment offload to the edge.

- A Tivoli Access Manager plug-in described below.
- Centralized Security. Tighter integration has been developed between the WebSphere Application Server Network Deployment and the Tivoli Access Manager with the Tivoli Access Manager plug-in. This will enable you to build centralized identity management solutions with global sign-on capabilities and enforceable policies to secure cached and non-cached J2EEâ„¢, Portal, Web and legacy resources. In addition, companies who implement this integrated solution will benefit from the ease of working with a single object namespace, representing the full set of security policies for the resources you want to protect.

Data flow from web browser to web container is via HTTPS. From inside the web container, servlet/JSP's access the EJB container is done via RMI/IIOP.
Data flow from Java client to the EJB container is done via RMI/IIOP
JMS Receiver request flow: From message sender to message topic/queue via JMS, MQ. From the topic/queue to the MDB.

JMS Sender Request flow: From the topic/queue to the message receiver via JMS, MQ.
Web Service Provider SOAP/HTTP(s) flow: From Web Service Client to Web Container done via SOAP/HTTP(s) to the web services engine to the EJB session bean.
Web Service Provider SOAP/JMS flow: From the Web service client to the message topic/queue via SOAP/JMS, then to the MDB and onto the EJB session bean or the web services engine.
Web Service Client SOAP/HTTP(s) request flow: Application artifacts (servlets, EJB's nad JavaBeans) within the application server act as Web Service clients. Flow goes from the artifacts to the web services engine and outbound to the Web Service Provider or Gateway via SOAP/HTTP(s)
Web Service Client SOAP/JMS request flow: Flow goes from the artifacts on the application server to the web services engine on to the message topic/queue and onto the web service provider via SOAP/JMS.

The web container executes servlets and JSP's, both of which are java classes that generate markup to be viewed by a web browser. Traffic into and out of the web container travels through the embedded HTTP server. While servlets and JSP's can act independently, they most commonly make calls to EJB's to execute business logic or access data. EJB's, which run in the EJB container, are easily reusable Java classes. They most commonly communicate with a relational database or other external source of application data, either returning that data to the web container or making changes to the data on behalf of the servlets/JSP.
In order to access a database from an application server, a JDBC provider is necessary. The JDBC provider allows access to the database.
There are three JDBC driver implementations for database access:

- Type 2 JDBC Drivers (Thick) – Require the database client software on the client node to connect to the database server.
- Type 3 JDBC Drivers (Net Protocol) – Require server side code to map net protocol to native database.
- Type 4 JDBC Drivers (Native Protocol) – Connect directly to the database using its native protocol.

Rather than having the JDBC Drivers directly communicate with the database, the communication is abstracted into a data source. Data sources can improve performance and portability for database access. Can use standard and XA data source.
Connection pooling is provided by two parts, a J2C connection manager or a Relational Resource adapter.

There are two main tools used to administer WAS, the administrative console and the wsadmin command line tool.

Evaluate the design considerations of IBM WebSphere Application Server, V6.0 packaging and installation in an enterprise environment (e.g., LDAP, database servers, Service Integration Bus Technology (SIB), etc.)

Service Integration Bus
A bus is a shared communication channel, and contains the bus-based messaging infrastructure in WAS.
A bus is totally contained within a cell, and you can have multiple buses in a cell.
Links can be created to other buses or to an MQ Queue Manager.
Adding a bus member associates a cluster or application server with an SIB. When a new Bus Member is defined, one messaging engine is automatically created on the corresponding application server or cluster.
An ME runs inside an application server and manages messaging resources. Messaging Engines are transparent to application code. Each server or cluster has at least one ME for each Bus with which it is associated. The ME has an associated data store for message persistence and overflow. A Messaging Engine is a light-weight runtime object.
Destinations are created on the bus and hosted on a Messaging Engine. Each Messagine Engine has its own data store for storing messages, transactions states and delivery records. This needs to be a persistent data store and WAS uses JDBC to access it. Messaging Engines can share a database, but each ME has its own schema within the database, which results in different tables.
Scalability is achieved by deploying multiple ME's per Bus to a cluster. (Bus(Cluster(Server1-ME)(Server2-ME))). Failover is handled by the HAManager.

Service Integration Technologies Architecture

- Service Integration Bus
- Bus Member
- Messaging Engine
- Destinations
- Message Store
- Mediation

WAS topology can have multiple buses
Each Bus can have servers or clusters or both as bus members.
When a server or cluster is made a bus member, a Messaging Engine is created.

Mediation
Mediation is the manipulation of in-flight messages between the production of a message by one application, and the consumption of a message by another application.
Mediation provides the ability to manipulate a message at a destination. The message can be transformed, or rerouted to a different destination. Mediation is a java program that encapsulates primitive mediation operations.

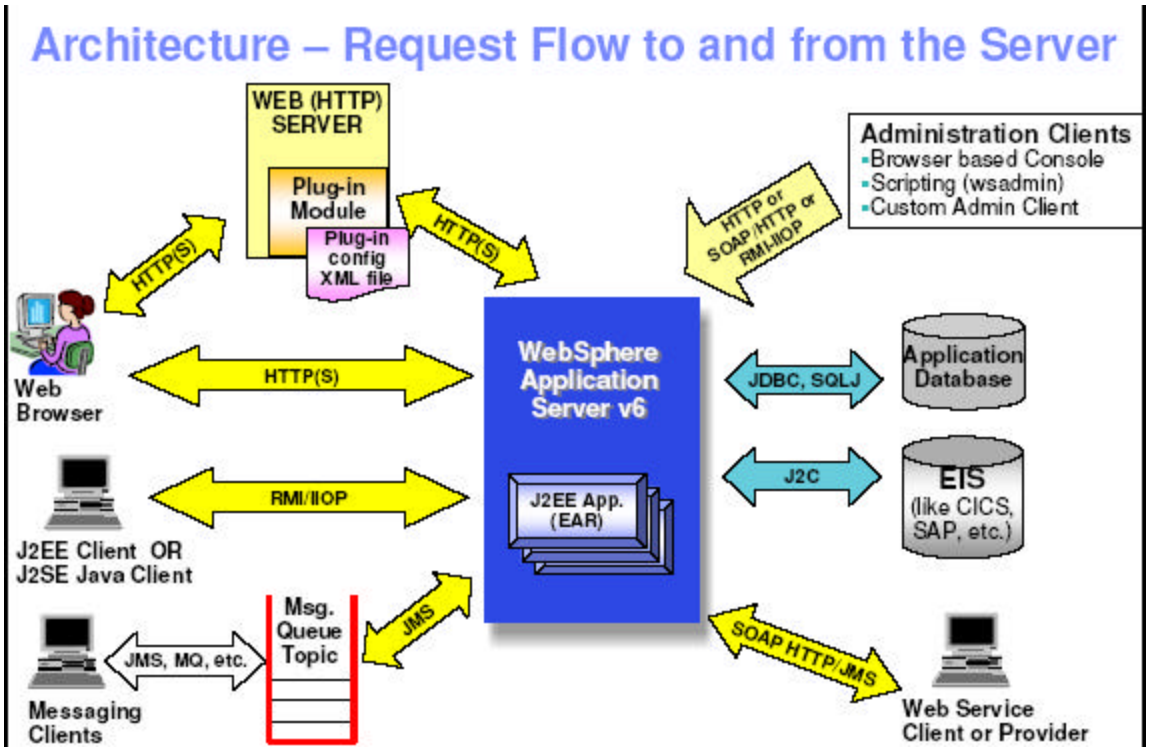
After a mediation handler is deployed to the bus, it is associated with the destination.
A mediation handler is a java program that performs mediation function and implements:

- com.bim.websphere.sib.mediation.handler.MediationHandler

Mediation handler can have properties that control its behavior.
Mediation handler is packaged for deployment with a supplied EJB (stateless session javabean) into an EAR file.

Articulate the various components of IBM WebSphere Application Server Network Deployment, V6.0 runtime architecture

Deployment Manager, Nodeagent, Application server.
The DM process manages the node agents. It holds the configuration repository for the entire management domain, called a cell. The DM is defined within a profile.
A node is a logical grouping of servers. Each node is managed by a single nodeagent process. Each node is defined within a profile. You can have multiple nodes on a single machine.
One of the benefits on running an unmanaged node is that you can create Web Server definitions.
An application server provides the functions that are required to support and host user applications. Each application server runs on only one node, but one node can support many application servers (JVMs).
WebSphere files are split into two components, Product files and Configuration files (profiles).
The deployment manager profile provides clustering and caching support, including failover support and WLM.



Browser administrative client connects to the admin console application running in the server:
http://<host>:9060/ibm/console
https://<host>:9043/ibm/console
User name is used to trace and save user-specific configuration data. The temp user workspace is saved under <WAS_INSTALL>/wstemp/<USER>/workspace

The node agent must be running before any application server on that node. The reason is that the node agent hosts the Location Service Daemon that is needed by the application server and application clients.

File Synchronization occurs either automatically or can happen manually via syncNode.sh script. File synchronization is a one-way process from the DM to the node agents and utilizes the HTTP(s) protocol. At synchronization time, the node agent send message digests to the DM, where they are compared against the digests in the master repository, consisting of:
Configuration files for all processes.
J2EE Application (Deployment descriptors files and binaries).

Workspace is a temporary space given to the user making Administrative Configuration changes – not needed for operational changes. Configuration XML files are copied from the master repository and cached into workspace. Different temporary space is maintained for each user. Workspaces are retained persistently and can be reused in the next login.

IBM HTTP server management and Tivoli Performance view monitor has been integrated in the Admin console.

Vertical scaling
Vertical scaling refers to configuring multiple application servers on a single machine and creating a cluster of associated application servers all hosting the same J2EE application(s).

In this case, the Web server plug-in routes the requests according to the application servers availability. Load balancing is performed at the Web server plug-in level based on a round-robin algorithm and with consideration of session state. Failover is also possible as long as there are active application servers (JVMs) on the system. Vertical scaling can be combined with other topologies to boost performance, throughput and availability.

Vertical scaling – many processes on one machine; single point of failure

Horizontal Scaling
Horizontal scaling exists when the cluster members are located across multiple machines. This lets a single application span over several machines, yet still presenting the application as a single logical image.

The Web server plug-in distributes requests to the cluster members on each node and performs load balancing and failover. If the Web server (Server A) goes down, then the WebContainer Inbound Chain of Server B or C could be utilized (limited throughput) meanwhile Server A or the Web server on Server A is repaired.
Be aware that this configuration introduces a single point of failure; when the HTTP server is out of service, your entire application is inaccessible from the outside network (internal users could still access the application server(s) using the WebContainer Inbound Chain). You can omit this SPOF by adding a backup Web server.
A Load Balancer, part of the WebSphere Edge Components, can be configured to create a cluster of Web servers and add it to a cluster of application servers.
Load balancing products can be used to distribute HTTP requests among Web servers that are running on multiple physical machines. The Dispatcher component of Load Balancer, which is part of the WebSphere Edge Components, is an IP sprayer that performs intelligent load balancing among Web servers based on server availability and workload capacity as the main selection criteria to distribute the requests
The Load Balancer Node sprays Web client requests to the Web servers. The Load Balancer is configured in cascade. The primary Load Balancer communicates to his backup through a heartbeat to perform failover, if needed, and thus eliminates the Load Balancer Node as a single point of failure. Both Web servers perform load balancing and failover between the application servers (cluster members) through the Web server plug-in.

The main theme with network deployment is distributed applications.

Horizontal scaling uses clustering and eliminates single points of process and hardware failure.

Describe workload management and failover strategies using IBM WebSphere Application Server, V6.0 High Availability Manager
A Core Group is a logical boundary that defines the set of processes that can provide each other with HA functionality. Node Agents, Cluster Members, and the DM can be members of a Core Group. A process can be a member of one and only one Core Group. Additionally, all members of a cluster must be members of the same Core Group. Members of a Core Group share information among one another, so that they are aware of the status of other Core Group members. Singleton services, such as WLM routing, running on one member of a Core Group can be failed over to any other member of the Core Group.
When the DM is installed, a default Core Group named "DefaultCoreGorup" is created. It has HA policies for the Transaction Manager and Messaging Engines predefined. As managed processes are added to the cell, they are automatically added to the DefaultCoreGroup.
Each Core Group has an HA Coordinator that manages the HA related work for the Core Group. It keeps track of what services are running in what processes and, in the case of failure, decides which process should restart that service. In cases where you have several clusters within your cell, it can be useful to configure more than one HA Coordinator in your cell, so that a single process does not get overloaded.
The Transaction Log Hot Standby allows hot failover of in-transit two-phase commit transactions. To set this, you need to check the box labeled, "Enable high availability for persistent services" when configuring your cluster.
An HA policy defines how failover occurs and which servers to use for failover. A Core Group can have different HA policies for different services. WLM clustering for transaction can use one HA policy, while messaging Engines can use another HA policy. By default, the Core Group has a clustered transaction manager policy and a messaging policy for Service Integration Buses.
In an HA policy, singleton services can be configured to run in three basic modes:

- Static
- 1 of N
- No Operation

In a Static policy, singleton services run on a single server, and the failure of that server will not result in another server recovering that service. The service will only be reactivated when the failed server is restarted.
In the "1 of N" policy the HA coordinator is enabled to manager failover. The coordinator determines which processes should run which services and assigns them accordingly. The coordinator will ensure that the singleton service is running on exactly one server at any given time. This is the suggested HA policy for most users. When configuring a "1 of N" policy you can choose "Preferred servers only", "Fail back" or "Quorum".
With Preferred servers only, singleton services will run only on servers in the preferred list, and if none of them are available, the singleton will fail.
With Fail back policy, the HA Manager that after a process has failed over, the process should be moved back to a preferred server when that server becomes available.
The "Quorum" policy specificies that singleton services should only be started if at least half of the servers in the Core Group are running. This means that if more than half of your servers fail, your singleton services will be automatically stopped.
The "No Operation" policy specificies that WAS will never activate singletons on its own. Instead, it will be told what to do using JMX commands. The value of this policy is that allows you to make the HA Coordinator aware of the actions taken by external clustering software, so that applications will always have access to the necessary resources.
Any of these options can be changed dynamically using wsadmin.

The HA Manager uses two parallel methods to monitor processes: TCP Keep-Alive sockets, and an active hearbeat. Both of these methods are used across the Core Group in a peer-to-peer fashion. You should tune your OS's TCP KEEP_ALIVE value to ensure that failures are detected in a reasonable time.
The second method used to determine server status is an active heartbeat. Each process sends a heart beat message to every other process once per 'N' seconds. If a peer fails to respond to 'M' consecutive heart beats, then services from that process will be failed over to other processes.
HA policies decide how and where services will be failed over to.

WLM is implemented by using Clusters of application servers.
What can be workload managed?

- HTTP requests - Spread across multiple web servers by an edge product.
- Servlet requests - Spread across multiple web containers by the web server plugin
- EJB requests - Spread across multiple EJB containers by the WLM service.
- Partitioned Messaging destinations - Spread across multiple Messaging Engines by the WLM survive
- Web services outbound requests - Routed directly by WLM service, no longer require external routing by the web server plugin

The Location Service Daemon process is responsible for the EJB routing table, which can have entries for servers in other clusters.

HA does not depend on the DM.
For failover of stateful session EJBs:
The DRS is used, which is similar to HTTP session failover.
This can be enabled on a per-Application basis.
WLM will fail beans over to a server that already has a copy of the session data in memory if possible.
Ability to collocate stateful session bean replicas with http session replicas with hot failover.

By definition, clusters are a set of application servers having the same applications installed, and grouped logically for WLM.
Cluster configuration is as follows:
Prefer local - Routes EJB client requests to local EJB, if possible.
Enable HA for persistent services(for transaction log recovery)
Backup Cluster - If the entire cluster fails, the backup cluster supplies failover routing information
Cluster member weights.

To install applications to a cluster select a cluster, rather than selecting a server. Required resources, such as databases, must be available to all cluster members. New in v6 there is a rollout update option available on clusters.

Basic WLM Request Routing
Load Balancer - The load balancer is an IP sprayer that makes intelligent load balancing decisions. Using the NDAdmin tool, you can set it up to route your HTTP servers based on round robin, statistical round robin, best, custom advisor, or content based routing. Once the request arrives at an HTTP server, the routing is weighted round robin - the only configuration option is how much 'weight' to give eachserver. The routing information, the list of available servers and their weights, is ultimately stored by the DM. When the HTTP server plugin is generated, servlet request routing weights are written into the plugin.xml file, which the HTTP server will reload at configurable intervals. When a client requests an IOR for an EJB, the LSD returns the IOR and a copy of the routing table. The client uses two in-memory copies of the table - one static, one dynamic. The static copy is only to cache a local copy of the weights.
If ant HTTP server fails, the load balancer will simply route around it. The plugin reads the cloneID from the session key, and can route the request to its originating server. If a server process fails, the HTTP server notes the failure and marks that application server as unavailable, then routes the request to the next cluster member. Sessions already in progress will have a server ID for that failed server; the HTTP server routes them to the next server. Session data can be handled in two ways. Session Persistence to a database, or internal messaging of session information. In v6, the scope of the

replication domain is the core group; DRS will coordinate with the WLM component to determine which cluster members should hold backup session information for a specific cluster member.
Other than the fact that Node Agents will not be notified of failed servers on other nodes, the impact of the DM going down is minimal. Singleton services, like the WLM routing component, are 'portable' in v6, they do not run in the DM.

There are two kinds of high availability: process high availability and data high availability.
Redundant hardware and clustering software are approaches to high availability.
We can divide availability into the following levels:

1. Basic systems. Basic systems do not employ any special measures to protect data and services, although backups are taken regularly. When an outage occurs, the support personnel restores the system from backup (usually tape).

2. Redundant data. Disk redundancy and/or disk mirroring are used to protect the data against the loss of a disk. Full disk mirroring provides more data protection than RAID-5.

3. Component failover. For an e-business infrastructure like WebSphere, there are many components. As we discussed above, an outage in any component may result in service interruption. Multiple threads or multiple instances can be employed for availability purposes. For example, if we do not make the firewall component highly available, it may cause the whole system to go down (worse than that, it may expose your system to hackers) even though the servers are highly available.

For WebSphere, we have process high availability (vertical scaling) and process and node high availability (horizontal scaling). Entity EJBs are persisted into the database. Highly available data management is critical for a highly available transactional system. Therefore, it is very important to balance the availability of all components in the WebSphere production system. Do not overspend on any particular component, and do not underspend on other components, either.

4. System failover. A standby or backup system is used to take over for the primary system if the primary system fails. In principle, any kind of service can become highly available by employing system failover techniques. However, this will not work if the software is hard-coded to physical host-dependent variables. We can configure the systems as active/active mutual takeover or active/standby takeover. Although the active/active mutual takeover configuration increases the usage of hardware, it also increases the possibility of interruption, and hence reduces the availability. In addition, it is not efficient to include all components into a single cluster system. We have a firewall cluster, LDAP cluster, WebSphere server cluster, and database cluster. In system failover, clustering software monitors the health of the network, hardware, and software process, detects and communicates any fault, and automatically fails over the service and associated resources to a healthy host. Therefore, you can continue the service before you repair the failed system. As we discussed before, as MTTR approaches zero, A increases toward 100%. System failover can also be used for planned software and hardware maintenance and upgrades.

5. Disaster recovery. This applies to maintaining systems in different sites. When the primary site becomes unavailable due to disasters, the backup site can become operational within a reasonable time. This can be done manually through regular data backups, or automatically by geographical clustering software.

Section 2 - Installation/Configuration of Application Server (13%)

WebSphere Application Server v6 Packaging

Content	WebSphere Application Server v6 – Express	WebSphere Application Server v6	WebSphere Application Server v6 Network Deployment
Core Application Server	Standalone Node	Standalone Node	Deployment Manager, Standalone Node, Managed Node
IBM HTTP Server v6 Web Server plug-ins	Yes	Yes	Yes
Application Client (not on zLinux)	Yes	Yes	Yes
Data Direct JDBC drivers [1]	Yes	Yes	Yes
Development and/or Deployment Tools [2]	Rational Web Developer, AST	AST, Rational Application Developer Trial	AST, Rational Application Developer Trial
Database included in the package	DB2 Express (Dev. Use only) [1]	DB2 Express (Dev. Use only) [1]	DB2
Edge Components	No	No	Yes
IBM Tivoli Directory Server (LDAP server)	No	No	Yes
Tivoli Access Manager Server	No	No	Yes
Production Ready Applications	IBM Business Solutions [2]	None	None

Adding WAS v6 application server or WAS v6 Express to Network Deployment cell is just a license upgrade – no additional code is needed.

Identify installation options and determine the desired configuration (e.g., silent install, required/desired plug-ins etc.)

The Update Installer updates the core product files in a WebSphere Application Server product. Service in a maintenance package might update the following files in the installation root directory:

- The SDK, Java technology edition, in the java/jre directory
- JAR files in the lib directory
- Scripts in the bin directory
- Profile templates

Log on as root on a Linux or UNIX operating system, or as a member of the administrator group on a Windows system. The migration tools in WebSphere Application Server Version 6.0.x support migration from WebSphere Application Server Version 4.0.x, Version 5.0.x, and Version 5.1.x.

For silent installations, modify the sample response files supplied, response.base.txt for base and response.nd.txt for network deployment. Then issue the following command to perform the silent installation:
./install.sh –options “myresponsefile.txt” –silent
Silent creation of profiles a response file which should be customized:

- DMGR – response.pct.NDdmgrProfile.txt
- Application Server – response.pct.NDStandaloneProfile.txt
- Custom – response.pct.NDmanagedProfile.txt

Installing Log Analyzer "silently" prevents installation messages from being displayed, but the file responsefile.txt for silent installation needs more information to install Log Analyzer. To silently install Log Analyzer, add the following option to this file: -P logAnalyzerBean.active="true"

Set the Performance and Analysis Tools property in the file responsefile.txt to true to install the Log Analyzer tool. The property in the responsefile.txt file is: -P performanceAndAnalysisToolsBean.active="true".

Some maintenance packages provide required service for existing profiles in addition to service for the core product files. Each maintenance package that has profile maintenance provides a script that changes the profile. The Update Installer prompts you to back up your configuration when installing a maintenance package that has required maintenance for profiles.

Some maintenance packages provide optional service for existing profiles. The readme file for the maintenance package describes whether the maintenance package contains optional service for existing profiles. If so, the readme file describes how to use the script provided with the maintenance package.

To apply a fixpack or interim fix on both DM and nodes:

1. Download the fixpack/fix to both the DM and all nodes.
2. Run the backupConfig command to back up the configuration of each profile that the maintenance package can update. Or archive the app_server_root/profiles directory to back up all of the profiles at once.
3. Stop all the JVM's.
4. Change directories to the app_server_root/bin directory. Issue the . ./setupCmdLine.sh. This is to set JAVA_HOME, verify echo $JAVA_HOME.
5. Change directories to the updateinstaller directory and use the update command to install the maintenance package. Install the maintenance package on the deployment manager node before installing the maintenance package on each application server node that you intend to update.
6. Change directories to the updateinstaller directory and run ./update.sh -silent -options responsefile

The updateinstaller/maintenance directory will have the maintenance .pak file that you are installing.

If you uninstall a maintenance package, the Update Installer does not uninstall the maintenance package from profiles. The reason for not removing the maintenance is that you might have configured the profile after installing the maintenance. To restore an original profile, use the restoreConfig command or copy the profile from the archived profile_root directory to replace the changed profile.

Coexistence

Coexistence is a state in which nodes from different versions of WebSphere Application Server can start and run in the same environment at the same time.

Coexistence, as it applies to WebSphere Application Server products, is the ability of multiple installations of WebSphere Application Server to run on the same machine at the same time. Multiple installations include multiple versions and multiple instances of one version. Coexistence also implies various combinations of Web server interaction.

Version 6.0 WebSphere Application Server products can coexist with the following supported versions:

- IBM WebSphere Application Server Advanced Server Single Edition and Advanced Edition Version 4.0.2 and later
- IBM WebSphere Business Integration Server Foundation Edition Version 4.1 and later
- IBM WebSphere Application Server Version 5.0 and later
- IBM WebSphere Application Server Network Deployment Version 5.0 and later
- IBM WebSphere Application Server Enterprise Version 5.0 and later
- IBM WebSphere Application Server Version 5.1.0 and later
- IBM WebSphere Application Server Network Deployment Version 5.1.0 and later

All combinations of Version 4.0.x products, Version 5.x products, and Version 6.0.x products can coexist so long as there are no port conflicts.

Migrating a V5 DM to a V6 DM
WASPreUpgrade copies configuration to backup copy
Create a V6 profile for the Dmgr
WASPostUpgrade copies and transforms configuration data from backup into new profile
Analyze migration log
Start V6 Dmgr

Migrating V5 Node in a Cell

1. V6 DM must be running
2. Stop servers of node and nodeagent
3. Run WASPreUpgrade - copies configuration backup directory
4. Create V6 Application Server Profile with same node name as V5 node name
5. Run WASPostUpgrade - copies and transforms configuration data from backup into new profile
6. Analyze upgrade log
7. Start V6 nodeagent and test

V5 DM is disabled after successful WASPostUpgrade - Reenable with script located in V5 DM bin directory, only as a last resort.

WASPreUpgrade <backupDirectoryName> <old_version_WebSphere_Dir> [administrationNodeName] <options>

- backupDirectoryName is the output backup directory
- old_version_WebSphere_Dir is the V4 or V5 WebSphere install directory
- Additional Optional V4 options
- administrationNodeName is the positional optional V4 AE administration node name
- -import V4 AE's xml configuration file
- -nameServiceHost used to call V4 XMLConfig
- -nameServicePort used to call V4 XMLConfig

WASPostUpgrade <backupDirectoryName> <options>

- backupDirectoryName is the output of WASPreUpgrade
- -profileName - V6 profile name
- [-portBlock - starting seed to create new ports]
- [-backupConfig - T/F decision to backup the new install config]
- [-replacePorts - T/F decision to add or replace virtual host ports]
- [-includeApps - T/F decision to migrate applications]

All web server plugin migration issues are manual. V6 plugin modules can send requests to V5 and V6 nodes.
Web server plugins are installed from the launchpad.sh . Main log file is located in $PLUGIN_HOME/logs/install/log.txt.
Other files located in $PLUGIN_HOME/logs/install/ are:

- log.txt
- masterConfigurationLog.txt
- install<WebServer>Plugin.log
- configure_<WebServer>_webserver.log
- installGSKit.log

Install WebSphere Application Server and create profiles

Profiles are the way you are allowed to run more than one application server on a single installation of WebSphere product files. Each profile uses the same product files. This creates an environment that is simpler to administer than multiple WAS installations. This also uses less disk space and simplifies product updates. There are two ways of creating and managing profiles, the Profile Creation Wizard (GUI) and the wasprofile script (Command line).

The vpd.properties file lists program components that are currently installed. The file helps ISMP and the installer programs of WebSphere Application Server products to recognize previous installations of WebSphere Application Server products and to control options for new installations.
ISMP uses a file called vpd.properties to trace WAS products that it installs on all distributed platforms except Solaris and HP-UX. The file lists program components that are currently installed, and files updated during install and uninstall. The file helps ISMP to recognize previous installs of WAS products and to control options for new installs.
The file is located in usr/lib/objrepos (AIX), root directory on Linux, and on C:\WINNT, or C:\WINDOWS.

Certain situations require you to edit and manually remove the entries in vpd.properties before reinstalling a WAS product:

- Bypassing the uninstaller program to uninstall a product manually.
- Uninstalling a product manually when the uninstaller program is not present or is not working.

Profiles are created based on templates supplied with the product. Each template consists of a set of files that provide the initial settings for the profile and a list of actions to perform after the profile is created. You can create a profile by using the Profile Creation Wizard, found in <was_root>/bin/ProfileCreator directory. A profile can also be manually created using the wasprofile command line tool. This will create a profile in silent mode using wasprofile –silent. Silent creation of profiles require a response file which should be customized:
Dmgr: responsefile.pct.NDdmgrProfile.txt
Application Server: responsefile.pct.NDStand-aloneProfile.txt
Wasprofile is a tool that should be used for managing profiles. The list of profiles and their properties can be found in <was_root>/properties/profileregistry.xml. Additional properties, such as log levels, can be found in <was_root>/properties/wasprofile.properties
When using the profile creation wizard you must use a consistent hostname structure, make sure all are either short or long hostnames. They can't be mixed.

The wasprofile script can:

- Create a new stand-alone application server profile.
- List all profiles
- Delete a profile
- The list of profiles and their properties can be found in <was_root>/properties/profileregistry.xml

To create a profile from the command line run the following script:

- <was_root>/bin/wasprofile

./wasprofile –create –profileName profile2 –profilePath "<profile_root>/profile2" –templatePath "<was_root>/profileTemplates/default" –nodeName was6hostNodeName –cellName was6hostNameCell –hostName was6hostName

Profiles give the ability to create multiple independent user configurations sharing the same install binaries.
Each profile define a WAS runtime environment, can be a stand-alone application server, a Dmgr, or a custom profile.
With profiles, there is now a separation of product files and user data. This requires less disk space and product update

time is simplified. Each profile instance is registered in a registry (XML file) called Profile Registry. Command line tools look at the registry to find the profile information. The registry file is stored in $WAS_HOME/properties/profileRegistry.xml . In order to start a server from a specific profile pass the -profile profileNmae to the startServer.sh script.
Profile Creator Tool - is located in $WAS_HOME/bin/ProfileCreator directory. Inside this directory are also three sample repsonse files for each of the profile types. You need to provide one of the types of profiles when creating one. PCT tool log files in the profile directory $WAS_PROFILE/logs/pctLog.txt.
Every time wasprofile is invoked. log files are created in $WAS_HOME/logs/wasprofile/ directory:

- wasprofile_create_AppServer01.log
- wasprofile_delete_AppServer01.log
- wasprofile_listProfiles.log

With the concept of profiles, there is no longer a need to install a separate product for the Deployment Manager; it can be installed by using the profile type 'Deployment Manager'.

To federate the profile into the cell:
1. Start the DM
2. Log onto the node and switch to the new profile bin directory <profile_name>/bin
3. Run ./addNode.sh <dmgr_hostName> <dmgr_soap_port>

This will create a <profile_root>/logs/addNode.log

Best Practices for addNode.sh script:
If you receive an OutOfMemory exception while using the addNode command, you may need to increase the heap size of the deployment manager. To increase the heap size of the deployment manager, adjust the Maximum Heap Size parameter using the administrative console. In the administrative console, go to System Administration > Deployment Manager > Java and Process Management > Process Definition > Java Virtual Machine > Maximum Heap Size.
In some instances it may take longer than anticipated for the deployment manager to respond to the addNode command. The default timeout value, which determines how long the client will wait for a server response, is appropriate in the majority of cases. However, you may require more time for the server to respond under heavier processing conditions. For example, if you include the -includeapps option and have a large number of applications, or the applications are very large, the default value of 180 seconds may be insufficient. To change the default timeout value, open the file $WAS_HOME/profiles/<profile name>/properties/soap.client.props in any ASCII text editor and find the following line (shown here with default value of 180 seconds):
com.ibm.SOAP.requestTimeout=180

If you need to change the default you can edit this line to set the timeout to a value more appropriate for your situation (Note: setting the above value to 0 will disable the timeout check altogether).

WebSphere installation variables:

<variable>	<value> Windows	<value> UNIX
<software_dir>	C:\Software	/cdrom/
<software_cds>	C:\Software_CDs	/cdrom/
<was_root>	C:\Program Files\IBM\WebSphere\AppServer	/opt/IBM/WebSphere/AppServer/
<profile_root>	C:\Program Files\IBM\WebSphere\AppServer\profiles	/opt/IBM/WebSphere/AppServer/profiles/
<his_root>	C:\Program Files\IBM HTTP Server	/opt/IBM/http_server/
<plugin_root>	C:\Program Files\IBM\WebSphere\Plugins	/opt/IBM/WebSphere/plugins/
<dbs>	C:\Program Files\IBM\SQLLIB	/opt/IBM/sqllib/

To being installation launch the ./LaunchPad.sh script. After the installation is complete, you then need to create a profile. Use the profile creation wizard to create application server run-time environments, called profiles.
After installation launch the administrative console, you should see the DefaultApplication installed. Next open a browser and point to http://localhost:9080/snoop

Verify the installation (e.g., ivt, verification using sample (snoop and/or hitcount)
A number of logs are created during the installation and profiles creation process.
Change to the <was_home>/logs and examine log.txt. This file records installation status.
Change to the <was_profile>/logs directory and examine wasprofile_create_profile1.log. This log records creation events that occur when creating the profile, profile1.
Change to the <profile_root>/profile1/logs:

- Examine the pctLog.txt. This log records installation events that occur when creating profiles with the Profile creation wizard.
- Examine backupConfig.log. This log records events that occur when creating a backup of the configuration directory structure.
- Examine ivt_config.log. This logs the messages during installation of the IVT test application. Verify that no errors occurred during installation.
- Examine ivtClient.log. This logs results from the installation verification command.
- Examine portdef.props. This properties file logs information about the default ports for an application server.

First Steps can be used to verify installation, there is one per profile.

The ivt tool verifies that the installation of the profile was successful and each profile has its own ivt utility that can be run. The ivt tool will start the application server automatically. After the server initializes the ivt tool runs a series of verification tests and displays pass or fail in the console window. It will also scan the SystemOut.log for errors and verify the functionality of the profile.

Troubleshoot the installation (e.g., identify and analyze log files)
First Steps is a post-installation ease-of-use tool for directing WAS elements from one place. This can be used to verify installation.
After installing the product, use the IVT tool. The IVT program scans product log files for errors and verifies core functionality of the product installation.

Application Server install logs
- $WAS_HOME/logs/log.txt
- $WAS_HOME/logs/ihs_log.txt
- $WAS_HOME/logs/WASPreUpgrade.log
- $WAS_HOME/logs/WASPostUpgrade.log
- $WAS_HOME/logs/instconfig.txt

Profile Creation
- $WAS_HOME/logs/launch_wsprofile.log
- $WAS_HOME/logs/wasprofile.log

Application Deployment
- $WAS_PROFILE/logs/<name>_deploy.log
- $WAS_PROFILE/logs/<name>_config.log

$WAS_PROFILE/logs/activity.log
$WAS_PROFILE/logs/wsadmin.traceout
$WAS_PROFILE/logs/pctLog.txt
$WAS_PROFILE/logs/$PROFILE/SystemOut.log
$WAS_PROFILE/logs/$PROFILE/SystemErr.log
$WAS_PROFILE/logs/$PROFILE/native_stdout.log
$WAS_PROFILE/logs/$PROFILE/native_stderr.log
$WAS_PROFILE/logs/$PROFILE/startServer.log

activity.log - binary log viewable with the log analyzer or the showlog command
The log analyzer is a GUI to the service.log and compares error messages to a database of known problems. The database can be updated with the latest symptoms using File > Update Database

JVM logs - SystemOut.log and SystemErr.log are self-managing and cal rool over based on time or file size. The number of historical files is configurable.

HTTP plugin trace can be set to Error, Warn, or Trace.

The collector tool is used to gather information for IBM support, information is collected into a JAR file. To use the Collector tool you must create a temp directory, outside of the WebSpehre directory. The collector tool will produce its output in the current working directory. This needs a JVM to run, and use the WebSphere installed JVM.

The waslogbr launches a stand-alone Log Analyzer, which is run against a symptom database. The symptom database should be updated often.

The genHistoryReport and genVersionReport can be used to generate html reports of the history and version of websphere.

The collector tool collects logs and configuratioin files into a single jar and is most often requested when engaging IBM support to troubleshoot an issue. The tool collects system information such as configuration, system OS, disk space, etc.

You can use the JspBatchCompiler to batch compile the jsp files so the initial client request is served much quicker on the production web server. Batch compiling makes the request much quicker because the jsp file is translated and compiled into a servlet.

The PropFilePasswordEncoder utility is used to replace encoded passwords in configuration files with clear text. The syntax is as follow:

PropFilePasswordEncoder -Filename -password properties list
EncAuthDataFile -Input file -output file

Section 3 - Application Assembly and Deployment and Server Resource Configuration (11%)

Describe the name service management of WebSphere Application Server (JNDI)
Develop your application using either JNDI or CORBA CosNaming interfaces. Use these interfaces to look up server application objects that are bound into the name space and obtain references to them. Most Java developers use the JNDI interface. However, the CORBA CosNaming interface is also available for performing Naming operations on WebSphere Application Server name servers or other CosNaming name servers.

Assemble your application using an assembly tool. Application assembly is a packaging and configuration step that is a prerequisite to application deployment. If the application you are assembling is a client to an application running in another process, you should qualify the jndiName values in the deployment descriptors for the objects related to the other application. Otherwise, you may need to override the names with qualified names during application deployment. If the objects have fixed qualified names configured for them, you should use them so that the jndiName values do not depend on the other application's location within the topology of the cell.
Naming is used by clients of WebSphere Application Server applications to obtain references to objects related to those applications, such as enterprise bean (EJB) homes.

These objects are bound into a mostly hierarchical structure, referred to as a name space. In this structure, all non-leaf objects are called contexts. Leaf objects can be contexts and other types of objects. Naming operations, such as lookups and binds, are performed on contexts. All naming operations begin with obtaining an initial context. You can view the initial context as a starting point in the name space.

The name space structure consists of a set of name bindings, each consisting of a name relative to a specific context and the object bound with that name.

You can access and manipulate the name space through a name server. Users of a name server are referred to as naming clients. Naming clients typically use the Java Naming and Directory Interface (JNDI) to perform naming operations. Naming clients can also use the Common Object Request Broker Architecture (CORBA) CosNaming interface.
Typically, objects bound to the name space are resources and objects associated with installed applications. These objects are bound by the system, and client applications perform lookup operations to obtain references to them. Occasionally, server and client applications bind objects to the name space. An application can bind objects to transient or persistent partitions, depending on requirements.

Users, groups, or the AllAuthenticated and Everyone special subjects can be added or removed to or from the naming roles from the WebSphere Application Server administrative console at any time. However, you must restart the server for the changes to take effect. A best practice is to map groups or one of the special-subjects, rather than specific users, to Naming roles because it is more flexible and easier to administer in the long run. By mapping a group to a naming role, adding or removing users to or from the group occurs outside of WebSphere Application Server and does not require a server restart for the change to take effect.

If a user is assigned a particular naming role and that user is a member of a group that is assigned a different naming role, the user is granted the most permissive access between the role that is assigned and the role the group is assigned. For example, assume that the MyUser user is assigned the CosNamingRead role. Also, assume that the MyGroup group is assigned the CosNamingCreate role. If the MyUser user is a member of the MyGroup group, the MyUser user is assigned the CosNamingCreate role because the user is a member of the MyGroup group. If the MyUser user is not a member of the MyGroup group, is assigned the CosNamingRead role.

The CosNaming authorization policy is only enforced when global security is enabled. When global security is enabled, attempts to do CosNaming operations without the proper role assignment result in a org.omg.CORBA.NO_PERMISSION exception from the CosNaming server.

In WebSphere Application Server, each CosNaming function is assigned to one role only. Therefore, users who are assigned the CosNamingCreate role cannot query the name space unless they also are assigned the CosNamingRead role. In most cases, a creator needs three roles assigned: CosNamingRead, CosNamingWrite, and CosNamingCreate. The CosNamingRead and CosNamingWrite roles assignment for the creator example in above have been included in

CosNamingCreate role. In most cases, WebSphere Application Server administrators do not have to change the roles assignment for every user or group when they move to this release from a previous one.

Use Application Server Toolkit (AST) for packaging J2EE applications, including Enhanced Ear Files
IBM Rational Web Developer
Used to be WebSphere Studio Site Developer (WSSD) in v5
Bundled with WebSphere Application Server v6 - Express

IBM Rational Application Developer
Used to be WebSphere Studio Application Developer (WSAD) in v5

Application Server Toolkit Application Server Toolkit (AST)
Packaged with WebSphere Application Server packages
Downloadable from the IBM support site
Supports Deployment and packaging of J2EE 1.4 applications for WebSphere Application Server v6

Enhanced EAR is a zip file that contains most of the application information needed to install in the Application Server J2EE EAR, Deployment information and some application resources (JDBC) and properties (like class loader)

Benefits: Improved productivity
Application resources/properties come with the application
Application install process creates the necessary resources within the server
Moving application from one server to another also moves the resources

Application scoped resources are tied to a specific application. The Enhanced EAR file includes embedded application resources so the application can be moved from one server to another. Application resources cannot be edited in the administrative console. The application needs the server resources not the server runtime. AST or wsadmin can be used to view/modify an application scope resource.

J2EE applications are EAR files and contain a deployment descriptor and one or more J2EE modules. Each module groups one or more application components that are of the same type, along with a deployment descriptor for that module.

EJB modules group related EJB's in a single module, and are packaged in JAR files. There is only one deployment descriptor for all of the EJB's in the module.
Web modules group Servlet class files, JSP's, HTML files and images. They are packaged in WAR files.
In WAS, the J2EE EAR file is enhanced with IBM Bindings and Extensions. These adapt the generic J2EE application to the IBM WAS environment.
IBM Bindings associate local names to deployment platform specific resources, for example security roles mapped to specific authentication credentials.
IBM Extensions support additional options such as access intent (caching and transaction isolation) and Web application reloading.
In the AST, the import wizard is used to import modules (EAR, files, EJB Jar, Application Client JAR, Web Module WAR) into a new or existing Enterprise Application. The J2EE perspective is used during the assembly/configure/deploy process.
The application.xml file defines all modules in the EAR file:

- EJB JAR's
- WAR's
- Resource Adapters
- Application Client JAR's
- Dependent Files
- Bind Application Security Roles

EJB Module Assembly
The AST import wizard imports into a new or existing Enterprise Application project. Adds EJB's in module (session bean, BMP entity, CMP entity, MDB). You can also configure the following using the deployment descriptor editor (EJB references, EJB relations, EJBQL queries, Security roles, method permission, access intent for entity beans).

When importing a WAR file containing a Web module into a Web project, you are given a chance to change the existing context root for the module.
After all modules have been imported and their deployment descriptors have been configured, all that is left to do is to export an EAR file which combines all the modules and their deployment descriptors.
When the AST starts, it asks which directory to use as the workspace. Each project should have its own separate workspace. Switch to the J2EE perspective to being assembling the application. Now create a new Enterprise Application project. Now import all EJB modules Next import all web modules and define the context root for it. Now click on the finish button. Now on the deployment tab of the deployment descriptor editor for the application you can define certain

resources that are included within the EAR. To add a data source, open the deployment descriptor editor, click on the deployment tab and then configure the data sources for the application. Select a JDBC provider, create a new J2C Authentication Data entry with UserID and Password. Inside the application deployment descriptor slect add to add a JAAS authentication alias. Enter the alias, userId and password. Now click finish. Now you can export the project topan EAR file.

Deployer provides
- JNDI names of EJB
- Target Mappings to server/cluster
- Can map application to a Web Server
- Web Services bindings

Deploying using wsadmin:
- Interactive shell prompts for binding information
  $AdminApp installInteractive C:\MyApp1.ear

- Non-interactive install
- Bindings could be pre-configured in EAR or specified on command
  $AdminApp install C:\MyApp1.ear

Use wsadmin -conntype NONE option if server is stopped.
To list installed Applications:
- $AdminApp list

To invoke help:
- $AdminApp help
- $AdminApp help installInteractive

To uninstall using wsadmin:
- $wsadmin -c "$AdminApp uninstall WebSphereBank"

Applications can be started/stopped.
Applications can be exported to the local file system
The table DDL can be exported to a file.

Classloader Information
Can be set at Application server level, application level or web module level.
Parent_First is the default level and this searches the immediate parent first.
Parent_Last tries to find and load the class from its own class loader and if the class isn't found it delegates to the immediate parent class loader and then the immediates parents.

Use WebSphere Rapid Deploy (WRD), Fine Grained Application Updates and Rollout Update
WebSphere Rapid Deployment is a collection of eclipse plugins. There are two syles to WebSphere Rapid Deployment: autoappinstall and freeform. WRD uses the eclipse framework and there is no GUI. There are two steps to initiate WRD, the first is to create a workspace for WRD to monitor and the second is to start monitoring the workspace. The tool is located under $WASPROFILE/bin/ and called deploytool. You will need to set the WORKSPACE variable and then run the wrd-config.sh script. You will need to specify the name of the project and the style of the project you want to use. There are also additional parameters that can be passed to wrd-config.sh
After WRD is configured run the $WASPROFILE/bin/wrd.sh script to enable monitoring of the specified workspace directory.

The WRD requires a workspace to be defined, after the workspace has been configured, a WRD monitor process is launched. This process detects changes in the workspace and acts accordingly, generating the needed artifacts and installing, updating or removing the artifacts from the running server. The wrd-config.sh script (located in <profile_root>/<Profile1>/bin) allows a user to create a set of common WRD-enabled workspaces without launching IRAD/AST in GUI mode. The script wrd.sh is used to start a WRD enabled workspace in headless mode.
To configure a new WRD project enter the following at a command line under:
- <profile_root>/<profile1>/bin
  wrd-config –project “MyProject” style “freeform” –runtime “was60” –runpath “<was_root>”

After WRD is configured, invoke wrd.sh to enable monitoring of specified directory
- <profile_root>/<profile_name>/bin/wrd.sh

Using the freeform style a developer creates individual J2EE artifacts (Java modules, static resources, deployment descriptors, etc) and needs to construct/install application:
1. Developer generates individual artifacts (.java, .jsp, .xml)
2. Developer places artifacts in monitored directory
3. WRD packages up artifacts in J2EE structure, generates required artifacts and installs/updates on WebSphere

Do not use hot deployment to update components in a production deployment manager managed cell. Hot deployment is well-suited for development and testing, but poses unacceptable risks to production environments. Full or partial resynchronization might erase hot deployed components. Also, running the restoreconfig command might overwrite

changes made to expanded application files. Further, hot deployed components are not migrated between versions of WebSphere Application Server. To add new components or modules to an enterprise application, reassemble the application EAR file so it has the new components or modules and then redeploy the EAR file.

To install and uninstall applications using wsadmin, $AdminApp is used
Ear is stored in config/cells/cellname/applications/appName.ear/

Deployment Descriptor (deployment.xml) is extracted at config/cells/cellname/applications/appName.ear/deployments

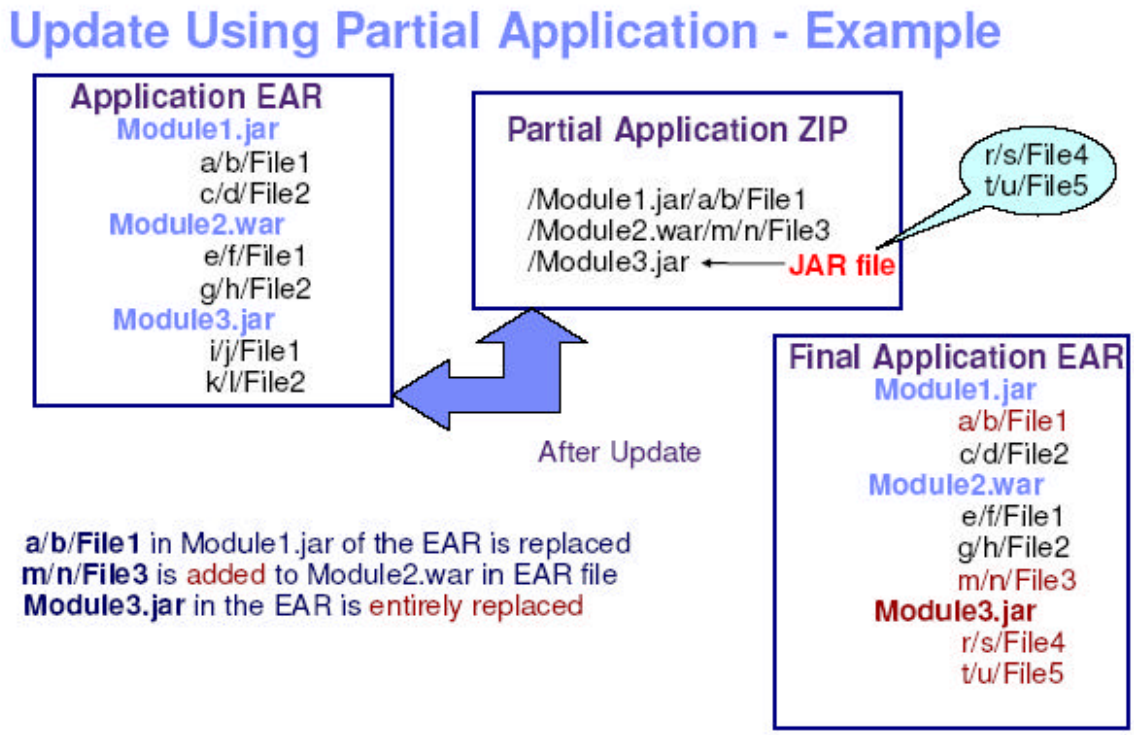
J2EE 1.2, J2EE 1.3 and J2EE 1.4 supported
J2EE 1.4 EAR files can contain J2EE 1.2 and J2EE 1.3 modules

Deployer provides necessary information:

- JNDI names of EJBs
- app-based name space bindings
- Target Mappings to server/cluster
- Can specify target as a web server (New to v6)
- Web services bindings

Applications >Enterprise Applications >Install an Application

Partial Application is a zip file containing application artifacts (it is not a valid j2ee archive).
During update, the contents are added or replaced within the deployed application EAR.
If the update zip contains a jar file, then the entire jar is replaced – its contents are not merged with the corresponding jar within the EAR.



In a Partial Application, File Deletion is specified using a special meta-data file META-INF/ibm-partialapp-delete.props

Module Start supported on all module types. (EJB, Connector, WAR..) Starting a module is useful when using Application Update to add a module to an EAR.

Uninstalling apps using wsadmin
wsadmin –c "$AdminApp uninstall WebSphereBank.ear"

Using wsadmin to update a module:
$AdminApp update appname {options}

Update Scenario	Default Behavior
Module Addition (EJB,Web..)	Start new module if application is running. Other modules are not affected. However, if security roles are defined in the new module DD, then app is restarted.
Web Module Deletion	Stop web module if app is running and remove it. Other modules are not affected.
Web Module Updates	For metadata changes, web module is recycled. All other changes are applied dynamically.
All other changes	Restart the application, if running.

Application Management uses JMX-based System management component to manage applications within WebSphere.

Fine grained application update support of application subcomponents as well as the entire application. Using Fine Grained Application updates, the smallest change is a single file, the largest is a whole application.

Partial Deployment

Partial application is a zipped file containing application artifacts, it is not a valid J2EE archive. During the update, the contents are added orreplaced within the deployed application EAR. If the update file contains a jar file, then the entire jar file is replaced.

The entire application or any of its components can be updated. For individual application files and updates, the file and the URI relative to the EAR root is provided. Individual application modules can be added, removed or updated. A partial application can be updated via a zip file which contains the application artifacts.

The partial application is a zipped file containing application artifacts, and is not a valid J2EE archive. It has the same hierarchical structure as they appear in the application EAR. During the update, the contents are added or replaced within the deployed application EAR. If the update zipped file contains a jar file, then the entire jar is replaced.

In a partial application, file deletion is specified using a special metadata file: META-INF/ibm-partialapp-delete.props. This can be at tha application scope or module scope.

Update API is added to AppManagement MBean and exposed via:

- wsadmin - $AdminApp update appname contenttype {options}
- Administrative Console
- Programming MBean interface.

Rollout update (ripple) feature provides 100% application availability.

- This can be selected in the Application Update user Interfaces
- When this is selected the application update is applied on each node in turn.

Define and map security roles (e.g., J2EE security)

J2EE security is concerned with controlling access to application resources. not system resources.

J2EE application level security is specified using security roles. Security roles allow developers to specify security at an abstract level. Security roles are applied to the Web and EJB application components, EJB methods or Web URI's.

Security can be specified in the following ways:

- Declaratively at assembly time, through the deployment descriptors.
- Programmatically using standard API's at development time.

Binding of users and groups to J2EE security roles is usually done at the application deploy (install) time.

Security roles have security permissions on the EJB methods or Web URI's, and these are set during the assembly of the components. The security users and groups are binded to the security roles during deployment.

Security can be applied to EJB's in the following two ways:

- Access control can be applied to individual session and entity bean methods so that only callers who are members of particular security roles can call those methods.
- Session and entity bean methods which need to be aware of the role or identity of the caller can programmatically call the J2EE API methods isCallerInRole() and getCallerPrincipal() to determine a caller's role and principal respectively. When using isCallerInRole(), the security role references are used, and these are later mapped to security roles.

The mapping of users and groups to J2EE security roles can take place during or after application installation. After installation, using the administrative console, go to the application under additional properties, select "Map security roles to users/groups." Then select the role and either:

Lookup and apply users or groups.
Map to everyone
Map to All authenticated

Declarative J2EE security - The developer does not need to code for security, since it will all be handled at deployment and runtime; this is called declarative security

Define J2EE security roles for EJB modules.

- Security role references - used to provide a layer of indirection between security roles named in the EJB java code and security roles that are defined at application assembly time.
- Configuring method access control
- EJB Run-As delegation policy
- Bean Level delegation
- Method Level delegation
- Run-As mapping

Programmatic J2EE Security - Programmatic security becomes useful when the application server provided security infrastructure cannot supply all the functionalities needed for the application. Using the Java APIs for security, developers could implement security for the whole application without using the application server security functions at all. Programmatic security also gives developers the option to implement dynamic security rules for your applications.

EJB security methods:

- java.security.Principal getCallerPrincipal() - The getCallerPrincipal method allows the developer to get the name of the current caller. To do this, you need to call getName() on the java.security.Principal object returned.
- Boolean isCallerInRole(String roleName) - The isCallerInRole method allows the developer to make additional checks on the authorization rights of a user which are not possible, or more difficult, to perform through the deployment descriptor of the EJB.

Define JDBC providers and data sources (e.g., resource scoping)

Resources and Environment variables can be defined at different scopes:

- Cell – Resources available to all servers within the cell.
- Node – Resources available to all servers within the node.
- Server – Resources available only to the servers.
- Cluster – Resources available to all cluster members of that cluster.
- Application – Resources are tied with the application and are available to the server running that application.

Creating JDBC Provider Resources:

- Create JDBC provider before defining data sources
- One JDBC provider needed for each unique database driver.
- JDBC providers can be defined at cell, node or server scope.

Create J2C Authentication Alias

- Click New to create a new alias
- Alias name
- User id and password

Installed applications that must interact with relational databases use JDBC providers for data access. Together, the JDBC provider and data source objects are functionally equivalent to the J2EE Connector Architecture (JCA) connection factory (which provides access to non-relational databases).
A data source is associated with a JDBC provider that supplies the specific JDBC driver implementation class. The data source represents the J2EE Connector Architecture (JCA) connection factory for the relational resource adapter. Application components use the data source to access connection instances to a specific database. A connection pool is associated with each data source.

<u>Steps to create a new data source</u>

Security > Global Security > Authentication > JAAS Configuration > J2C Authentication Data > New

Alias = db2user
UserID = wsbeta
Password = password

Click apply and save to master repository.
Now go to Resources > JDBC Providers > New > Select Scope

Database Type = DB2
Provider Type = DB2 Universal JDBC Driver Provider

Implementation Type = XA data source
Click Next tab
Name = Example DB2 JDBC provider
Classpath = ${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar, db2jcc_license_cu.jar, db2jcc_license_cisuz.jar
Implementation class name = com.ibm.db2.jcc.DB2XADataSource
Click ok
Now click on JDBC Providers > Example DB2 JDBC provider > Data sources
Name = BANKDS
jndi name = jdbc/bank
Choose to use CMP or not.
DB2 Universal data source properties
Database Name = BankDB
Driver Type = 2
port number = 50000
Authentication
Component-managed authentication alias = node1/db2user
Container-managed authentication alias (deprecated in v6) - node1/db2user
Authentication alias for XA recovery - Use component-managed authentication alias.
Click OK. Then click save to master repository
Select Datasource and then Test Connection.

Define SIB Resources
Creating a new bus member:
Service Integration > Buses > New
name = mybus
Click ok.
Additional Properties > Bus members > Add
Choose server scope. Accept defaults. Click Finish
Now go to
Buses > mybus > Additional Properties > Destinations > New
Select Destination Type - Queue
Identifier - BankJSQueue
Click Next
Assign the Queue to a bus member - node:server1
Click Next. Click Finish. Click Save to master repository.

Configure J2C resource adapters, connection factories (resource scoping) and Message Driven Bean Activation Spec)
A resource adapter is a system-level software driver that a Java application uses to connect to an enterprise information system (EIS). A resource adapter plugs into an application server and provides connectivity between the EIS, the application server, and the enterprise application.
Resource adapter archive file
A Resource Adapter Archive (RAR) file is a Java archive (JAR) file used to package a resource adapter for the Java 2 Connector (J2C) Architecture for WebSphere Application Server.

A RAR file can contain the following:
- Enterprise information system (EIS) supplied resource adapter implementation code in the form of JAR files or other executables, such as dynamic link lists (DLL).
- Utility classes.
- Static documents, such as HTML files, images, and sound files.
- J2C common client interfaces, such as cci.jar.

Configuring Java 2 Connector connection factories in the administrative console
Steps for this task
- Click Resources.
- Click Resource Adapters.
- Select a resource adapter under Resource Adapters.
- Click J2C Connection Factories under Additional Properties .
- Click New.
- Specify General Properties .
- Select the authentication preference.
- Select aliases for component-managed authentication, container-managed authentication, or both.
- If none are available, or you want to define a different one, click Apply > J2C Authentication Data Entries under Related Items.
- Click J2C Auth Data Entries under Related Items.

- Click New.
- Specify General Properties.
- Click OK.
- Click OK.
- Click the J2C connection factory you just created.
- Under Additional Properties click Connection Pool.
- Change any values desired by clicking the property name.
- Click OK.
- Click Custom Properties under Additional Properties.
- Click any property name to change its value. Note that UserName and Password if present, are overridden by the component-managed authentication alias you specified in a previous step.
- Click Save.

Section 4 - Security (11%)

Implement security policies (e.g., authentication and authorization (using different security registeries), etc.)

Enabling Global Security using LDAP User Registry

1. Set up WebSphere v6 to use LDAP registry, then set up Global security
2. Specify the user ID/password that is used to run the application server, this must be a valid user in the LDAP directory.
3. Type in the LDAP directory vendor and LDAP hostname and LDAP port 389 or 636
4. Type in the Base DN name for the users - o=ibm,c=us
5. Type in the Bind DN name used when doing LDAP searches for the users. Some LDAP servers do not support anonymous LDAP binding for search, so you may need to provide the Bind DN and password. cn=user2,o=ibm,c=us
6. Turn on SSL if you need SSL communication between WebSphere and LDAP.
7. Open up the LDAP key store with ikeyman and extract the key database certificate file as a .arm file.
8. Now load the certificate extracted in the previous step and import it into the jks key store that WebSphere is using.
9. Now Global Security can be turned on and the servers started. (The two authentication mechanisms that can be used are either SWAM or LTPA).
10. Click save. The cell/cellName/security.xml file will be updated

Configure the Custom Registry for Authentication

1. Click on the 'Security' then the 'Global Security' tab. Under the 'User Registries' tab click on 'Custom'.
2. Specify the Server user ID - must be a valid Server user ID and password in the Custom Registry. This is used for authentication during server startup and is 1st user with all Administrative rights.
3. Specify any Custom Properties needed by the custom registry. It will be passed to the implementation via java.util.Properties in the Registry initialization method.
4. Now you can turn on Global Security, Specify the authentication mechanism (LTPA or SWAM both support user registries).
5. Under the Active User Registry specify Custom User Registry.
6. Save the changes - These changes are made to the security.xml file - Start the servers.
7. To add new users, enter the 1st user admin id to log into the console. Next go to 'System Administration' then 'Console settings' and there you can add users or groups.
8. Click save. The cell/cellName/security.xml file will be updated

The Custom Registry class name must be in the WebSphere extension class loader path.

Enable Global Security using LTPA and Local OS Registry

1. Security then Global Security tab
2. Enter a valid Administrator user in the Local OS registry
3. Next step is to create the LTPA keys, since we are using LTPA as the authentication mechanism.
4. Specify the password used to generate the LTPA keys, there is also the Timeout in which the LTPA token is invalidated.
5. The import/export keys is to distribute the same LTPA keys on the App. servers of the distributed cells that participate in the same LTPA token forwarding and SSO.
6. Now create the SSO domain, if no domain is used then the hostname is used and SSO will work only for that server.
7. Turn on the interop Mode when using a mix of v4/v5 servers in the single SSO domain.

8. 'Web inbound security attribute propagation' is used when there is a need to propagate security attributes when forwarding LTPA tokens - this is helpful when using Reverse Proxy where you have the proxy server propagate its attirbutes to the server.
9. In the authentication mechanism choose LTPA, and in the user registry choose Local OS.
10. Click save. The cell/cellName/security.xml file will be updated.

The LTPA keys are automatically generated the first time that you enable security. After you enable security, WAS can generate a new set of keys in two ways. If you need to change the password, make the change and click OK or apply to generate the keys. You do not need to click generate keys. If you do not need to change the password, click generate keys. The new set of keys are not used until they are saved.

If global security is turned on, you must specify the authentication method (LTPA or SWAM) and the user registry (Local OS, LDAP, or a custom registry). In addition you will need to supply a valid user id and password which is already present in the user registry.

Trust association allows a 3rd party reverse proxy security server to act as a front-end authentication server for web http requests into Websphere.
LTPA authentication process and calls.
When using LTPA, a token (called LTPA token) is generated with user information, an expiration time and is signed by the keys. LTPA protocol uses cryptographic keys to perform data integrity (signing) and data confidentiality (encrypting) on user data, that passes between the servers.

Protect WebSphere resources
Managed resources in WebSphere are represented by JMX beans. All these management interfaces in WebSphere are protected by role-based access control.
CSIv2 takes care of passing credentials between EJB containers
LTPA takes care of passing credentials between the web containers by enabling single sign-on for the cell.
A realm is a collection of users that are controlled by the same authentication policy.

The Global security settings for a cell are almost the same as the global security settings for a server. The difference is that only LTPA is available as an authentication mechanism, as this allows credential-forwarding.

After configuring security for a cell stop and restart the components in the following order:
- Stop the application servers
- Stop the node agents
- Restart the deployment manager
- Start the node agents
- Start the application servers

Resources such as JMS and JDBC can be authenticated and authorized by the following types:
Application: res-auth=Application
Container (WebSphere): res-auth=Container

Component Managed Authentication
Applies only when res-auth=Application.
If credential (used ID/pwd) are specificed in the getConnection() method, those credentials are going to be used.
If no credential specified in getConnection(), then the value in CMA alias specified with the resource used.
If application managed authentication is specified, either the user ID/pwd must be specified in the method or as the CMP alias in the resource. (If not, an Exception will be thrown if the back end resource requires authentication).
Component Managed Authentication behavior is the same as WAS v5.
Tools | Resource Reference or the CMP EJB CMP Connection Factory Binding.
Authorization: Per_Connection_Factory
On Data Source Resource panel: Component Managed Authentication Alias (used if application does not provide ID/pwd).

Container Managed Authentication
Container managed authentication can be used with or without specifying a JAAS login.
JAAS login configuration is specified in the application used Resource Reference or CMP Connection Factory Binding (for CMP EJB's)
Tools | Resource Reference or the CMP EJB CMP Connection Factory Binding.
Authorization: Container
Authentication method: JAAS login Configuration
On the Data Source Resource Panel: Specify Container Managed Authentication Alias
Default JAAS login configuration - Mapping Configuration Alias: NONE (Any other value will throw an Exception).

Container Managed Authentication: JAAS login
Configuration on Application side:
Tools | Resource Reference or the CMP EJB CMP Connection Factory Binding
Authorization - Container Authorization type: Container
Authentication Method - JAAS login Configuration: Default
Authentication Alias (data): Specify J2C authentication alias (user/pwd)
Configuration on Resources Side:
On Data Source Resource Panel, set:
Container Managed Authentication Alias: Ignored
Mapped Configuration Alias (JAAS login configuration): Ignored

There are different resource authentication mechanisms:
Application Managed
Container Managed

Web clients use the HTTP or HTTPS protocol to send the authentication information. Java clients use CSIv2/SAS, TCP/IP, or SSL to authentication.
There are three ways of authenticating:

- Basic Authentication
- Credential Token
- Client Certificate Authentication

Protecting EJB's
EJB applications programmatic API's:
IsCallerInRole(string role-name)
Returns true if the bean caller is granted the specified security role.
If the caller is not granted the specified role, or if the caller is not authenticated, it returns false.
If the specified role is granted Everyone access, it always returns true.
Must have security role reference defined in the deployment descriptor.

getCallerPrincipal():
Returns the java.security.Principal object containing the bean caller name.
If the caller is not authenticated, it returns a principal containing UNAUTHENTICATED name

Messages arriving at MDB have no client credentials. When an MDB needs to call a secure EJB, it needs security credentials. Provide Run-As identity for the MDB to authenticate with.

Protecting Web Components
The web component needs to let the server know how the web client should provide client authentication. This can be done using Basic Authentication (Not secure), Client Authentication or Form-based authentication. Web resources is a set of URL patterns and HTTP methods. This is specified in the web deployment descriptor. Unlike EJB's where the constraint is placed on EJB methods, with web components it is on the web resource.
Binding the Roles to the users/groups is done in websphere after deployment.
Web Applications programmatic API's:
isUserInRole(string role-name): Returns true is the remote user is granted the specified security role. Returns false, if the remote user is not granted the specified role, or no user is authenticated.
getUserPrincipal(): Returns the java.security.Principal object containing the remote user name
getRemoteUser(): Returns the user name the client used for authentication.

Changing Identity: Run-As is a way for a servlet or JSP to change the identity when calling downstream processes or EJB's.

Java 2 Security provides a policy-based, fine-grain access control mechanism to control the Java code access to system level resources. All Java code runs under a security policy that grants the code access to certain resources.
Examples of permissions protected by Java 2 Security:

- File Access
  - canRead(), FileInputStream(), RandomAccessFile(), isDirectory(), isFile(), length(), canWrite(),
  - FileOutputStream(), mkdir(), renameTo(), createTempFile(), delete(), deleteOnExit()
- Network Access
  - send(), receive(), getLocalAddress(), getHostName(), getLocalHost(), getAllByName()
- Java VM
  - classLoader(), loadLibrary(), checkPermission(), checkLink(), checkExit()
- Program Threads
  - stop(), resume(), suspend(), interrupt(), setPriority(), setName(), setDaemon()
- System Resources
  - getPrintJob(), setProperty(), getProperty(), setDefault(), getFont(), getEventQueue()

- Security Aspects
    - getFields(), getMethods(), getConstructors, setPublicKey(), addCertificate()

CSIv2 configuration

1. Configure client's CSIv2 outbound authentication protocol
2. Configure CSIv2 server's inbound authentication protocol
3. Configure servers CSIv2 outbound authentication protocol
4. Configure CSIv2 server's inbound authentication protocol

Inbound configuration is used for server side EJB, and outbound configuration is used by EJB client running in WAS.

Resources such as JMS and JDBC can be authenticated and authorized by the following type:

Application: res-auth=Application

Container (WebSphere): res-auth=Container

Component Managed Authentication

If credentials are specified in the getConnection() method, those credentials will be used.

If no credential specified in getConnection(), then the value in component-managed authentication alias specified with the resource is used.

If application managed authentication is specified, either the userID/password must be specified in the method or as the component-managed authentication alias with the resource. If not an exception will be thrown if the back end resource requires authentication.

SIB Security

Authentication occurs when client creates a connection to SIB resources. UserID/pwd are authenticated using the configured User registry of the application server.

When a JMS client connects to a bus, the user/pwd must be correctly configured:

Component-managed: specified by the application

Connection Managed: specified in the connection factory used to create the connection to the bus.

V6.0 Support for JACC, the Java Authorization Contract with Containers as required for J2EE 1.4

CORBA Security - Any calls made among secure ORB's are invoked using IBM proprietary Secure Association Service (SAS) or J2EE standard CSIv2 authentication protocol that sets up the security context and the necessary quality or protection. After the session is established, the call is passed up to the enterprise bean layer.

Global security is either ON or OFF for the entire Cell in a ND package. The Authentication type and registry applies to the entire cell also.

The main difference between Java 2, JAAS, and J2EE Security is how the security is enforced.

Java 2 Security is enforced by the JVM and by the server with the permissions specified in policy files.

JAAS Security is enforced programmatically from within an application.

J2EE Security is enforcable by the J2EE application server or programmatically from within an application.

CSIv2 is the authentication protocol, used by EJB and EJB client over RMI/IIOP.

J2EE 1.4 Security Features

Java 2 Security - Access to System Resources

- Enforces access control, based on the location of the code and who signed it. Not based on the principal.
- Defined in a set of Policy files.
- Enforced at runtime.
- Authorization is performed using J2EE security roles. Security roles are applied to the Web and EJB application components (EJB methods or Web URI's). Binding users and groups to J2EE security roles is usually done at application install time.

JAAS Security - Authentication and Authorization

- Enforce access control based on the current Principal or Subject.
- Defined in Application Code.
- Enforced Programmatically.
- Used for any type of Java Code - Stand-alone application, Applet, EJB, Servlet.

J2EE Security Roles - Authorization of J2EE application artifacts.

- Role based security - Roles defined in the J2EE EAR file
- Defined in application configuration settings (Deployment Descriptors).
- Enforced by runtime, programmatically, or both.

CSIv2 - Used for Authenticating EJB's, replacing IBM proprietary SAS and z/SAS protocols.

SSL is used by multiple components within the application server:

- HTTP server and web container

- ORB component using IIOP over SSL
- LDAP client using LDAP over SSL

JAAC allows application servers to interact with 3rd party authorization provider using standard interfaces to make authorization decisions.

WebSphere components that can use SSL are:
- HTTP Transport
- ORB
- LDAP Client
- SOAP port

Define and implement WebSphere administrative security
You should secure the administrative console to restrict access to the configuration.
Define users and groups and assign roles to them.
Four available roles are:
Monitor – Can view config files and view current state
Configurator – Can change config files and view current state
Operator – Can view config files and stop/start server
Administrator – Can change config files and stop/start server

Mapping a role to everyone means that anyone can access resources protected by this role, and essentially, there is no security.

Roles are mapped to all authenticated users, and all authenticated users in the selected user registry are granted access to the role.

As stated above, Global security is either ON or OFF for the entire Cell in a ND package. The Authentication type and registry applies to the entire cell also.

Configure WebSphere plug-in to use SSL
Set the authentication mechanism as client-cert
- Create a self-signed certificate for the web server plugin
- Create a self-signed certificate for the embedded http server in the web container
- Exchange public keys between the peers
- Modify the web server plugin file to use SSL/HTTPS
- Modify the web container to use SSL/HTTPS

The Web server plug-in requires a keyring to store its own private and public keys and to store the public certificate from the Web container's keyfile (Generating a self-signed certificate for the web server using the ikeyman IBM tool).

Section 5 - Workload Management/Scalability/Failover (13%)

Singleton services that require failover can be divided into two categories, user resources and system resources. User resources are artifacts related to a user application, such as a transaction log or a messaging engine. These resources can fail over only to another member of the same cluster, because only members can fail over to another member of the same cluster, because only members of the same cluster are guaranteed to be running the same application. System resources, such as WLM routing, can be run on any process with the same Core Group. A Core Group is a boundary that defines the set of processes that can provide each other with HA. In the default and most common configuration, all processes are members of a single Core Group.
What can be Workload Managed:
- HTTP Requests – Load Balancer – HTTP Server plugin
    - Servlet Requests HTTP Server plugin – Web containers
        - EJB Requests – Web Container or Java Client – EJB containers

- HTTP requests - spread across mutliple web servers by an edge product. External to WebSphere Application Server

- Servlet requests - spread across multiple web containers by the web server plugin. Application servers are clustered. Server weighted round robin routing. plugin-cfg.xml provides the routing table to the HTTP server.

- EJB requests - Spread across multiple EJB containers by the WLM service. Location service daemon provides routing table to the ORB. Stateful session beans use DRS, which is similar to HTTP session failover. WLM will fail beans over to a server that already has a copy of the session data in memory if possible.

- Partitioned messaging destinations - Spread across multiple ME's by the WLM service

- Web Services outbound requests - routed directly by WLM service, no longer require external routing by the web server plugin

High availability does not depend on the deployment manager.
High Availability
Singleton Services that require failover fall into two categories:
- User resources - application related resources - ME, 2phase commit XA.
- System Resources - used internally by WAS - WLM routing

A Core Group defines a set of processes that can cooperate to provide each other with high availability. Processes can be DM, Node Agents, Application Servers, Cluster Members. A process is a member of exactly one Core Group. All members of a cluster must be within the same Core Group. WLM information is shared automaticallt between Core Group members.
Each Core Group has an HA Coordinator that coordinates all HA activities among the Core Group processes. Any process in the core group can be the HA coordinator. This is selected using an internal algorithm. An HA policy defines how failover occurs and which servers to use for failover. A Core Group can have different HA policies for different services.

Clusters are a set of Application servers having the same applications installed. and grouped logically for WLM. Clusters are contained within a cell, and clusters may span physical machines/nodes.
Cluster members can be configured for WLM with weighted values.

Basic WLM Request Routing.
- Load Balancer
  Routing decision table stored internally. Configurable with the NDAdmin tool. Multiple intelligent routing options.

- HTTP Server Plugin
  Routing table part of the plugin-cfg.xml file. Configured with administrative Web console or wsadmin scripting tool.

- WLM-aware client
  Includes Web containerm Java Client, EJB. Routing table is supplied by the LSD. Configured with admin console or wsadmin scripting tool.

With web container failover, in flight sessions may be persisted to a database or replicated in memory using DRS.
With EJB container failover, if failure occurs before any work was initiated on the cluster member ORB automatically reroutes EJB request. If no other cluster member is available, it throws "NO_IMPLEMENT". If "NO_IMPLEMENT" is thrown no recovery is possible.

The need to keep the DMGR is only for configuration changes and JMX routing. It is no longer a SPOF for WLM routing, as in v5.
Consequences of a DMgr failure:
- Unable to broadcast configuration changes to Node Agents
- Administrative Console unavailable.
- w sadmin unavailable
- No changes to the cell configuration.

Federate nodes (including custom profiles)
A WebSphere cell defines an administrative domain. A deployment manager provides centralized administration of all resources in the cell. It is created as a profile. A node is a logical grouping of servers. Each node is managed by a single node agent and each node is defined by a profile. Each profile uses the same set of product files. Multiple profiles can be installed on a single machine. An application server in the ND product can run in a deployment manager cell as a managed node or its own as a stand-alone application server. Multiple application server profiles can be created on a single machine. Each application server profile can be federated into a cell. If there are multiple base profiles on a single machine, they can be federated into the same cell, different cells, or remain stand-alone. Once in a cell, it becomes it's own node which activates a node agent. The deployment manager profile is used to create a deployment manager process and it can exist on an independent machine, or on a machine with other profiles.
A custom profile creates a WebSphere node without any application. By default it is automatically federated into a cell during profile creation. Use the administrative console to create servers and clusters on the federated node. There are multiple ways to create a profile:
- <was_root>/bin/ProfileCreator directory
- Manually via the wasprofile command line tool

Use can create profiles in silent mode via the –silent switcha dn this requires a response file which should be customized:
- responsefile.pct.NDdmgrProfile.txt
- response.pct.NDStand-aloneProfile.txt
- responsefile.pct.NDmanagedProfile.txt

You can use the –listProfiles switch passed to the wasprofile command to list the profiles. The list of profiles and their properties can be found in <was_root>/properties/profilesregistry.xml

Adding a node to a cell can be done via the administratrive console or the addNode.sh script. The process of adding a node to a cell:

- Creates a backup of current configuration
- Connects to the deployment manager
- Configures the node agent
- Add's node's applications to cell configuration if –includeapps is passed.

After the node has been added

- Use the startNode.sh script to start the node agent, if the node is not started.
- Use the syncNode.sh script to synchronize the node with the cell configuration.

When a node is removed from a cell, the profile reverts back to the configuration it had when it was initially federated into a cell. So any applications or configuration changes that were made while it was part of a cell are lost.

By default after running the addNode utility the node is started. You can pass the -noagent option to the addNode utility so the nodeagent won't start. If this is passed a file synchronization won't take place. You add a node to cell during the creation of a custom profile. The add node utility now has a resolution algorithm for port conflicts. This algorithm looks at the ports used by all the profiles of the current installation only. The application server that you wish to federate to the cell must be running before it can be added.
The add node utility should be run from the stand-alone application server and the dmgr and dmgr port (8879 default) must be passed. When adding an application, if an application with the same name already exists in the cell, a warning is displayed and the application is not installed.
To add the node agent as a Windows service, the -registerservice option can be passed.
There is a new option -portprops <file> that can be used to list specific ports to be used by the addnode utility.
The federated node inherits global security settings of the DM Cell.

The DM must be running to add or remove a node. The nodeagent must be running to remove a node from a cell.
Log files will be in the node profile's log directory:
$WAS_PROFILE/logs/addNode.log
$WAS_PROFILE/logs/removeNode.los

Node names in the cell must be unique.

Create clusters and cluster members
Only available with the Network Deployment insallation.
Steps for this task
Go to the Cluster Members page. Click Servers > Clusters in the console navigation tree. Then, click a cluster in the collection of clusters and click Cluster Members. The Cluster Members page lists members of a cluster, states the nodes on which members reside, and states whether members are started, stopped or encountering problems.
Click New and follow the steps on the Create New Cluster Members page.

1. Type a name for the cluster member (application server).
2. Select the node on which the server will reside.
3. Specify the server weight. The weight value controls the amount of work directed to the application server. If the weight value for the server is greater than the weight values assigned to other servers in the cluster, then the server receives a larger share of the servers' workload. The value can range from 0 to 100.
4. Specify whether to generate a unique HTTP port.
5. Specify whether to create a replication entry for the server.
6. Specify the server template.
7. Click Apply to finish the cluster member. Repeat steps 1 through 7 to define another cluster member.
8. Click Next.
9. Review the summary of information on new cluster members and click Finish.
10. Click Save on the administrative console taskbar and save your administrative configuration.

To examine a cluster member's settings, click on the member's name under Member Name on the Cluster Members page. This displays the settings page for the cluster member instance.

It is far easier to build a highly available environment with WAS v6.0. You need only install WAS on your servers. Because no external clustering software is required, it is more practical to deploy a large network of blade servers. However, you will need to ensure that any resources on which WAS depends are made highly available. This means storing transaction logs on a highly available Network Attached Storage system, and making sure that your database servers are clustered.

With WAS v6.0 the Deployment Manager is no longer a SPOF for WLM routing. However, still requires a shared file system or shared devices with external cluster software to be highly available. The DM no longer runs the WLM routing service, so it is no longer a single point of failure for WLM. It is only needed for configuration changes and JMX routing. In v5.0 the node agent had to be failed over to support the recovery of transaction logs or JMS messaging engines. In v6.0 this is no longer necessary because the HA manager can recover the transaction logs or the messaging engine on another server, provided that the transaction logs or the message store are accessible to all servers. You should still use

the “rc” or a similar service to start your node agent in the case of a failure, since the Location Service Daemon still only runs inside the node agent.

There are different ways to add a Node to a cell. It can be done from the DM Admin Console, or it can be done from the stand alone node running the addNode script. You can pass the includeapps switch if you want to include the applications. The default is ‘no’. From the Admin Console you navigate to System Administration > Nodes > Add Node. Here you will need to enter the hostname and JMX port of an instance to be federated Node to the Cell. To use the addNode script run: addNode.sh/bat <dmgr_host> [dmgr_port] [options} – default port is 8879. Newly added node inherits security from the cell.
A node can be removed from the cell using command “removeNode.sh/bat”. If security is on the user/pwd will be needed for the command. This stops all running server processes in the node. Backed-up original configuration will be restored, this is the backup configuration created during the addNode.

Clusters cannot be mixed between distributed systems and z/OS.
To create a cluster:

1. Servers – Clusters – New cluster
2. Enter cluster name
3. Select node
4. Set routing weight
5. generate unique ports
6. apply and regenerate plugin
7. Synchronize changes with nodes

Now you can install Enterprise applications to the cluster.

Basic WLM Request Routing
Routing Decision Points

1. Load Balancer – Roputing table stored internally. Configured using Load Balancer Tool. Multiple intelligent routing options.
2. HTTP Server plugin – Routing table part of plugin-cfg.xml. Configured using admin console or wsadmin.
3. WLM aware client – Web container, Java client, EJB. Routing table supplied by name server. Configured using admin console or wsadmin. Prefer local (yes or no)

Weighted Routing Example:

After	Server1	Server2
Request 0	4	1
Request 1	3	1
Request 2	3	0
Request 3	2	0
Request 4	1	0
Request 5	0	0
Reset	4	1

Once all values = zero a reset is done. Round robin is used to distribute the hits. The plugin reduces to the lowest common denominator. EJB WLM uses two tables for Prefer Local and Remote.

It is possible for the distribution of hits to become unbalanced. This could happen if one server happens to get a bunch of requests that require affinity. In this case, the server in question could be doing more work than intended. This algorithm corrects this unbalance by preventing that server from taking any new hits until such a time that it has rebalanced itself relative to the work done by the other servers. So an imbalance can occur. The system automatically corrects itself while still ensuring that requests with affinity go to the right server.

Web container failover – If a plugin detects that a web container is unavailable, it marks that container as down and does not any additional attempts to communicate with it for 2 minutes. Once the 2 minutes are over, it attempts communication again and continues this process if the container is still marked as down. In-flight session may be persisted to a database or can be replicated in JVM memory.

EJB Container failover – Client code and ORB plugin can route to next available cluster member.
If a failure occurs before any work was initiated on the cluster member:

- ORB automatically reroutes EJB request.
- If no other cluster member is available, it throws a NO_IMPLEMENT exception. Failure

If failure occurs after EJB method call initiated work:

- System exceptions are thrown
- Client needs to determine appropriate recovery action – Reissue request, rollback transaction, etc.
- If NO_IMPLEMENT exception is thrown, no recovery is possible.

HTTP Session Management – HTTP is a stateless protocol

- Sessions allow you to maintain state information across a series of HTTP requests from the same client.
- Java servlets specification defines session management process for Web applications.

Session manager stores session information, and sends client a unique session ID through:

- Cookie in HTTP header
- URL rewriting in parameter on links or forms.

Session affinity – An application may retain state information for a user's session in memory. HTTP server plugin routes subsequent servlets requests consistently to the same application server after the session is created. Uses server ID with session ID in cookie or URL. Session affinity routes session's requests to same server.

Session Persistence – Session objects are cached in server memory by default, and therefore are lost if server fails. Two methods to enable session persistence are using a database or using memory-to-memory replication.
Using a database for session persistence, session objects are persisted using a JDBC datasource. DB2 is included in WAS package for session persistence. This has more overhead.
Memory-to-Memory replication – Sessions are copied to another server using WebSphere Platform Messaging. This could fail if all copies of the session are lost. Session persistence via database or memory-to-memory.

Memory-to-Memory session replication configuration:
Three replication modes – Server mode, Client mode, both. In server mode only stores backup copies of other WAS session and does not send out copies of any session created in that particular server. In client mode, it only broadcasts copies of the sessions it owns and does not receive backup copies of sessions from other servers.

Replication Domains configuration:
The number of replicas can be: Single Replica, Entire Domain, Specified. Single Replica means that for every session object, there is one backup copy stored on a different server. It is also possible to have the sessions backed up on every member of the domain. This would not tend to scale well since every member has to store every other member's session objects.

Service Integration HA
To create clustering, create a Bus Member to assocaite the cluster to a Bus. Workload sharing is dictated by how many ME's you have added to the cluster bus member. By default when you add a cluster to a Bus one ME is created. Add additional ME's to the cluster Bus Member.

Scalability is achieved by deploying multiple ME's per bus within a cluster. With this flexibility, there are one or more concurrently active ME's running in a cluster on the same bus. The ME's are monitored by the HA Manager. Depending on the HA policies, if a server fails, each ME that is running on that server is activated on another server.
Any cluster member in the cluster can run the ME. HA Manager decides which server runs the ME at any given time. If the ME fails, the HA Manager activates an ME on another server in the cluster. WLM ensures that messaging activity for the ME is directed to the appropriate server. For messaging failover to work properly, the data store needs to be accessible to all of the cluster members. Destinations can be associated with a Bus Member of type 'Cluster'.
The different types of HA policies are:

- static - cannot be failed over
- 1 of N (Default) - The HA Manager elects one cluster member to host ME.
- OS Managed - Failover is done via 3rd party HA framework to specify where to place the ME (JMX commands).

Simulating Failure to test HA
Send a JMX command using wsadmin to a ME MBean. You can inject a local or global error.
Get MBean name for the ME

- set mbean_name[$AdminControl queryNames "type=SIBMessagingEngine, Name messaging_engine_name"]
- $AdminControl invoke mbean_name injectFault LocalError
- $AdminControl invoke mbean_name injectGlobalError

The results will be sent to the SystemOut.log

Evaluate session state failover options (in memory, database persistence, messaging, partitioning)
HTTP sessions are identified by session IDs. A session ID is a pseudo-random number generated at the runtime. Session hijacking is a known attack HTTP sessions and can be prevented if all the requests going over the network are enforced to be over a secure connection (meaning, HTTPS).

IBM WebSphere Application Server provides a service for managing HTTP sessions: Session Manager.
Steps for this task

- Plan your approach to session management, which could include session tracking, session recovery, and session clustering.
- Create or modify your own applications to use session support to maintain sessions on behalf of Web applications.
- Assemble your application.
- Deploy your application.
- Ensure the administrator appropriately configures session management in the administrative domain.
- Adjust configuration settings and perform other tuning activities for optimal use of sessions in your environment.

By configuring your application server to use the WLM even distribution of HTTP requests function, HTTP session objects can be evenly distributed by workload management (WLM) to the servants in your configuration. You can use this task to distribute HTTP session objects in a round-robin manner among several servants instead of the normal situation where there is a servant affinity, and HTTP session objects reside in one or two servants.

One of the important functions that the plugin provides, in addition to failover and wlm, is the ability to manage HTTP sessions. Only a single cluster member may control/access a given session at a time. After a session has been created, all following requests need to go to the same application server that created the session (known as session affinity). If the application server is unavailable when the plugin attempts to connect to it, the plugin will choose another cluster member and attempt a connection. Once a connection to a cluster member is successful, the session manager will decide what to do with the session. The cluster member will find that it does not have the session cached locally and thus will create a new session. To avoid the creation of a new session, a distributed session can be used to access sessions from other application servers. There are two mechanisms to configure distributed sessions in WAS v6:

- Database Persistence - Session state is persisted to a database shared between the clustered application servers.
- Memory-to-Memory replications - Session state is persisted, based on DRS. It provides in-memory replication of session state between clustered application servers.

There are three methods of identifying a user's session to the application server:

- Cookies - plugin uses JSESSIONID to manage requests. lifetime is governed by the cookie lifespan
- URL rewriting
- SSL ID

The session manager also allows the administrator to permit an unlimited number of sessions in memory. If the administrator enables the Allow overflow setting on the session manager, the session manager permits two in-memory caches for session objects. The first cache contains only enough entries to accommodate the session limit defined to the session manager, 1000 by default. The second cache, known as the overflow cache, holds any sessions the first cache cannot accommodate, and is limited in size only by available memory. The session manager builds the first cache for optimized retrieval, while a regular, un-optimized hash table contains the overflow cache.

Create and configures DRS (Data Replication Service) replication domains

The Data Replication Service is an internal component of the WebSphere Application Server. It is used by other components to move data from place to place. The most visible use of DRS is for replicating persistent HTTP session data so that if an application server fails, the request can be routed to another application server, and the session data will be available there.

In order to minimize the impact of a failure, DRS coordinates with the WLM routing algorithm to assure that requests and data end up in the same place.

A new feature in v6 is the capability to capture the state of a stateful session bean and enable failover to another instance of that bean in another application server.

DRS is used by multiple WAS components:

HTTP session memory-to-memory replication
Dynamic cache replication
Stateful session EJB state replication

A replication domain consists of servers and cluster members that have the capability to replicate information from one cluster member to any other cluster member.

When creating a replication domain, create a replication domain and define how many replicas of the data it should contain.

In WAS v6 one change includes the cooperation between the DRS and the WLM subsystem to coordinate which cluster members serve as backups for other cluster members. Another improvement is that the underlying mechanism has been rewritten using a proprietary transport to move data. This reduces overhead and improves overall system performance. The change in the underlying communication mechanism removes the need for replicators. The only configuration decision you can make is how many backup copies of the data will be needed. The default is one. This new communication mechanism benefits from a faster transport mechanism, the channel framework, which eliminates the one-thread-per-queue limitation. Sitting on top of a more robust transport also removes the need for manual partitioning.

When you create a cluster, creating a replication domain is as easy as selecting a checkbox. Because DRS is used for both cache replication and session data, you can configure cache replication under Server, then container service, then select dynamic cache replication.

In the single replica(default) topology, each server in the domain will hold a replica of the data from one other server.
In the Entire Domain topology, data flows from each process to every other process, so that for each replicated object, there are four remote copies and the local original. This topology is overkill for HTTP session replication, but it is the only allowable configuration for cache replication. When caching dynamic content, the cache is only useful if it is available on all the machines where a request could arrive.

When troubleshooting a DRS problem, you should examine the SystemOut.log for all servers in the cluster, and their server.xml files, which contain the DRS configuration information. For further tracing use the string: com.ibm.ws.drs.*

Suggested best practices include creating a distinct domain for HTTP and EJB session data, and another domain for Cache replication. The number of replicas you configure will have an impact on failover and on performance. Using a small number improves performance. Increasing the number of replicas may reduce the time it takes a session to move to another server, but it does so at the cost of voerall performance. If congestion messages appear in the logs, increase the Transport buffer size to 50MB(default is 10MB) for each server.

DRS
DRS is a mechanism for moving data around within WAS processes for replication purposes for specific functions.
DRS is used by multiple WAS components:

- HTTP Session memory-to-memory replication
- Dynamic Cache replication
- Stateful Session EJB state replication

DRS coordinates with WLM.
In WAS v6 You create a Replication Domain and define how many replicas of the data it should contain.
A Replication Domain consists of server or cluster members that have the capability of sharing WAS internal data (HTTP Session, Cache) within the domain.

Section 6 - Maintenance and Performance Tuning (19%)

Manage application configurations (e.g., application bindings, tune HTTP session configuration parameters such as timeout value, persistence, etc.)
Session tracking –

- Enable SSL ID tracking Enable Cookies Enable URL rewriting | Enable protocol switch rewriting

Maximum in-memory sessions
Session timeout

HTTP Session Management Configuration
WAS session support generates a unique session ID for each user, and returns this ID to the user's browser via a cookie. The default cookie name required by the servlet specification is JSESSIONID. You can configure the exchange of cookies between the browser and the server will only occur if HTTPS sessions is used. The cookie maximum age can be configured to be the current browser session or a user specified maximum age.
URL encoding is another way to send back session. This requires the developer to use special encoding API's. The URL encoding stores the session identifier in the page returned to the user.

Session Management can be configured to use a database or memory-to-memory replication.

WebSphere session support keeps the user's session information about the server. WebSphere passes the user an identifier known as a session ID, which correlates an incoming user request with a session object maintained on the server.
WebSphere supports three approaches to tracking sessions:

- SSL session identifiers
- Cookies
- URL rewriting

Many Web applications use the simplest form of session management: the in-memory, local session cache. The local session cache keeps session information in memory and local to the machine and WebSphere Application Server where the session information was first created.
Local session management does not share user session information with other clustered machines. Users only obtain their session information if they return to the machine and WebSphere Application Server holds their session information about subsequent accesses to the Web site. Most importantly, local session management lacks a persistent store for the sessions it manages. A server failure takes down not only the WebSphere instances running on the server, but also destroys any sessions managed by those instances. WebSphere allows the administrator to define a limit on the number of sessions held in the in-memory cache from the administrative console settings on the session manager. This prevents the sessions from acquiring too much memory in the Java VM associated with the application server.

The session manager also allows the administrator to permit an unlimited number of sessions in memory. If the administrator enables the Allow overflow setting on the session manager, the session manager permits two in-memory caches for session objects. The first cache contains only enough entries to accommodate the session limit defined to the session manager, 1000 by default. The second cache, known as the overflow cache, holds any sessions the first cache cannot accommodate, and is limited in size only by available memory. The session manager builds the first cache for optimized retrieval, while a regular, un-optimized hash table contains the overflow cache.

For best performance, define a primary cache of sufficient size to hold the normal working set of sessions for a given Web application server.

General Properties for session Management

- Maximum in-memory session count
- Allow OVerflow - Allowing an unlimited amount of sessions can potentially exhaust a system memory and even allow for system sabotage. Someone could write a malicious program that continually hits your sit, creating sessions, but ignoring cookies or encoded URL's and never utilizing the same session from one HTTP request to the next.
- Session timeout
- Security Integration
- Serialize session access

Perform WebSphere backup/restore and Archive Configuration tasks

backupConfig - a simple utility to back up the configuration of your node to a file. By default, all servers on the node stop before the backup is made so that partially synchronized information is not saved. Saved as a zip file.
You can run this from the server installation or a network deployment installation.
The –nostop option tells the backupConfig command not to stop the servers before backing up the configuration.

You create the backup of the profile, by going to <profile_root>/profileName/bin and running ./backupConfig.sh

restoreConfig - a simple utility to restore the configuration of your node after backing up the configuration using the backupConfig command. By default, all servers on the node stop before the configuration restores so that a node

#restoreConfig <backup_file> [options]
-nowait
Tells the restoreConfig command not to stop the servers before restoring the configuration.
-quiet
Suppresses the progress information that the restoreConfig command prints in normal mode.
-location <directory_name>
Specifies the directory where the backup file is restored. The location defaults to the app_server_root/config directory.
-logfile <fileName>
Specifies the location of the log file to which information gets written.
-profileName
Defines the profile of the Application Server process in a multiple profile installation. The -profileName option is not required for running in a single profile environment. The default for this option is the default profile.
-replacelog
Replaces the log file instead of appending to the current log.
-trace
Generates trace information into the log file for debugging purposes.
-username <name>
Specifies the user name for authentication if security is enabled in the server. Acts the same as the -user option.
-user <name>
Specifies the user name for authentication if security is enabled in the server. Acts the same as the -username option.
-password <password>
Specifies the password for authentication if security is enabled in the server.
-help
Prints a usage statement.
-?
Prints a usage statement

#backupConfig <backup_file> [options]
-nostop
Tells the backupConfig command not to stop the servers before backing up the configuration.
-quiet
Suppresses the progress information that the backupConfig command prints in normal mode.
-logfile <fileName>
Specifies the location of the log file to which information gets written.
-profileName <profileName>
Defines the profile of the Application Server process in a multi-profile installation. The -profileName option is not required for running in a single profile environment. The default for this option is the default profile.
-replacelog
Replaces the log file instead of appending to the current log.
-trace
Generates trace information into the log file for debugging purposes.
-username <name>

Specifies the user name for authentication if security is enabled in the server. Acts the same as the -user option.
-user <name>
Specifies the user name for authentication if security is enabled in the server. Acts the same as the -username option.
-password <password>
Specifies the password for authentication if security is enabled in the server.
-help
Prints a usage statement.
-?
Prints a usage statement.

syncronization does not occur during the restoration. If the configuration directory already exists, it will be renamed before the restoration occurs.
You can run this from the server installation or a network deployment installation.

By default all servers on the node stop before the backup is made so that partially saved information is not saved.

If the config directory already exists, it is renamed before the restore occurs. When a restore is made, all servers on the node are stopped to prevent a synchronization from taking place and are not restarted. They will need to be manually started after the restore.

You can run the syncNode command when the DM is not running in order to forace a node to be in sync with the master configuration. This can be useful if there is a problem with the node and it won't start.

Configuration Archive
New to WAS v6 is the configuration archive, this is a complete or subset of WAS configuration. The configuration information is virtualized to make it portable. This removes any install specific information, like hostname.
This is stored in a file with .car extension. You can export and import a car file. The export converts the real configuration into templates. It breaks any relationship of configurations in the archive and the system configuration. When a cluster is exported, the relationship between the cluster and its members is preserved. Configuration may also refer to system resources outside scope:
database driver library, database name, etc.
You can export a server or websphere profile.
Here is a wsadmin example:
$AdminTask exportServer { -archive /opt/websphere6/archives/myArchive.car -nodeName node1 -serverName server1}
$AdminTask exportWasprofile { -archive C:\myCell.car }

Importing a configuration archive is the process of updating/creating system configuration with contents in configuration archive. The import operation compares node properties between source and target.
Here are some wsadmin import examples:
$AdminTask importServer { -archive /opt/websphere6/archives/myArchive.car [-nodeInArchive node1] [-serverInArchive server1] [-nodeName node1] [-serverName server1] }
$AdminTask importWasprofile { -archive C:\myCell.car }

Monitor size of logs/files and backup/purge as needed

A self-managing log file writes messages to a file until reaching either the time or size criterion. At the specified time or when the file reaches the specified size, logging temporarily suspends while the log file rolls over, which involves closing and renaming the saved file. The new saved file name is based on the original name of the file plus a timestamp qualifier that indicates when the renaming occurs. Once the renaming completes, a new, empty log file with the original name reopens and logging resumes. All messages remain after the log file rollover, although a single message can split across the saved and the current file.

Troubleshooting > Logs and Trace > server > JVM Logs > Configuration

- File Name
- File formatting
- Log file rotation
- File Size
- Maximum Size
- Time
- Start Time
- Repeat Time
- Repeat time
- Maximum Number of Historical Log Files

Manage the plug-in configuration file (e.g., regenerate, edit, propagate, etc.)
If you edited the plugin-cfg.xml files on federated nodes, turn off automatic synchronization of nodes on the File Synchronization Service page to prevent the edited files from being overwritten. Because a plugin-cfg.xml file on a Network Deployment machine is stored in the configuration repository, the plugin-cfg.xml file is overwritten on all federated nodes in your network whenever the Network Deployment machine updates the configuration repository on these nodes. If you need to regenerate the plugin-cfg.xml file on the Network Deployment machine:
Save your edited plugin-cfg.xml files on the federated nodes.
Force node synchronization of each federated node.
For managed Web Server nodes, the plugin xml file is automatically managed by the DM.

You run the GenPluginCfg.sh script, then perform a full synchronization so the plugin-cfg.xml file is replicated in all the WebSphere Application Server nodes.

Web server plug-in request routing properties
To view this administrative console page, click Servers > Web Servers > Web_server_name Plug-in Properties > Plug-in server_cluster_name Properties.

- Load balancing option
- Retry interval
- Maximum size of request content
- Remove special headers
- Clone separator change

If you select a Web server plug-in during installation, the installer program configures the Web server to identify the location of the plugin-cfg.xml file, if possible. The Web server is considered installed on a local machine if it is on the same machine as the application server. It is considered installed on a remote machine if the Web server and the application server are on different machines.

If the Web server is installed on a remote machine, the plug-in configuration file, by default, is installed in the
plugins_root/config/web_server_name directory.
If the Web server is installed on a local standalone machine in a Network Deployment environment, the plug-in configuration file, by default, is installed in:
profile_root/config/cells/cell_name/nodes/web_server_name _node/servers/web_server_name
If the Web server is installed on a local distributed machine in a Network Deployment environment, the plug-in configuration file, by default, is installed in the
profile_root/config/cells/cell_name/nodes/node_name/servers/web_server_name directory.

For remote Web servers, you must copy the file from the local directory where the Application Server is installed to the remote machine. This is known as propagating the plug-in configuration file. If you are using IBM HTTP Server V6 for your Web server, WebSphere Application Server can automatically propagate the plug-in configuration file for you to remote machines provided there is a working HTTP transport mechanism to propagate the file

Web server plug-in request and response optimization properties
To view this administrative console page, click Servers > Web Servers > web_server_name Plug-in Properties > Request and Response.

- Enable Nagle Algorithm for the IIS Web Server
- Chunk HTTP response to the client
- Accept content for all requests
- Virtual host matching
- Application server port preference
- Priority used by the IIS Web server when loading the plug-in configuration file

Web server plug-in caching properties
To view this administrative console page, click Servers > Web Servers > Web_server_name Plug-in Properties > Caching Properties.

- Enable Edge Side Include (ESI) processing to cache the responses
- Enable invalidation monitor to receive notifications
- Maximum cache size

Web server plug-in tuning tips
- Balancing workloads
- Improving performance in a high stress environment:

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\tcpip\Parameters\TcpTimedWaitDelay
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\tcpip\Parameters\MaxUserPort

Servers

- Application servers
- Generic servers
- JMS Servers
- Web servers
    - Generate Plugin
    - Propagate Plugin
- Clusters
- Cluster topology
- Core groups
- Core group settings
- Core group bridge settings

Tune performance of WebSphere Application Server (e.g., configure caching, queuing and pooling parameters, tune JVM heap size, etc.)

Generic JVM arguments
Quickstart (-Xquickstart) Avoiding class verification (-Xverify:none) Class garbage collection (-Xnoclassgc) Garbage collection threads (-Xgcthreads)
Heap compaction (-Xnocompactgc) Initial system heap size (-Xinitsh) Thread local heap size (-Xmc) Local cache object allocation (-Xml)
Initial heap size Maximum heap size
Disable just in time (JIT) compiler
Garbage collection policy -Xgcpolicy:[optthruput|optavgpause]

To reduce memory usage consider lowering the EJB Inactive pool cleanup interval. This is the interval at which the EJB container examines the pools of available bean instances to determine if some instances can be deleted.

Java Tuning
Enterprise applications written in Java involve complex object relationships and utilize large numbers of objects. Although Java automatically manages memory associated with an object's life cycle, understanding the application's usage patterns for objects is important. In particular, ensure the following:

The application is not over-utilizing objects
The application is not leaking objects (that is, memory)
The Java heap parameters are set to handle the use of objects

Understanding the effect of garbage collection is necessary to apply these management techniques.

The JVM allocates areas of storage inside the Java heap, where objects, arrays, and classes are stored. An allocated object is considered live when there exists at least one reference to it, that means, it is used by someone, commonly another object. Thus the object is also considered reachable. When this object is no longer used by anyone, all references should have been removed, it is now considered garbage, and its allocated storage area should be reclaimed for reuse. This task is performed by the Garbage Collector. When the JVM is unable to allocate an object from the current Java heap because of lack of free, contiguous space, a memory allocation fault occurs (allocation failure) and the Garbage Collector is invoked. (The GC can also be invoked by a specific function call: System.gc(). However, when System.gc() is invoked, the JVM can simply 'take it under advisement' and choose to defer the GC operation until later if the JVM has more pressing needs to attend to.) The first task of the GC is to make sure to acquire all locks required for garbage collection, and then stops all the other threads. Because of this garbage collection is also referred to as stop-the-world (STW) collection. Garbage collection will then take place in three phases: mark, sweep, and optionally compact.

Mark phase
In the mark phase, all reachable objects that are referenced either directly by the JVM, for example through threads stacks, or in turn by other objects, will be identified. Everything else that is not marked is considered garbage.

Sweep phase
All allocated objects that are not marked are swept away, that is, the space used by them is reclaimed.

Compaction phase
When the garbage has been removed from the heap, the GC can consider compacting the heap, which is typically riddled with holes caused by the freed objects by now. When there is no chunk of memory available big enough to satisfy an allocation request after garbage collection, the heap has to be compacted. Because heap compaction means moving objects around and updating all references to them, it is extremely costly in terms of time, and the GC tries to avoid it if possible. Modern JVM implementations try to avoid heap compaction by focusing on optimizing object placement in the heap.

Heap thrashing

Avoid heap thrashing at all costs. It is caused by a heap that is barely large enough to avoid expansion but not large enough to satisfy future allocation failures. Usually a garbage collection cycle frees up enough space for not only the current allocation failure but a substantial number of future allocation requests. But when heap is getting thrashed, each garbage collection cycle frees up barely enough heap space to satisfy just the current allocation failure. The result is that the next allocation request leads to another garbage collection cycle, and so on. This scenario can also occur due to lots of short-lived objects.

EJB Container

Cache settings - Cache size and cleanup interval

The cleanup interval specifies the interval at which the container attempts to remove unused items from the cache in order to reduce the total number of items to the value of the cache size.

The cache size specifies the number of buckets in the active instance list within the EJB container.

ORB Thread Pool Size

The degree to which the ORB thread pool value needs to be increased is a function of the number of simultaneous servlets (that is, clients) calling enterprise beans and the duration of each method call. If the method calls are longer or the applications spend a lot of time in the ORB, consider making the ORB thread pool size equal to the Web container size. If the servlet makes only short-lived or quick calls to the ORB, servlets can potentially reuse the same ORB thread. In this case, the ORB thread pool can be small, perhaps even one-half of the thread pool size setting of the Web container.

Web Container

To route servlet requests from the Web server to the Web containers, a transport connection between the Web server plug-in and each Web container is established. The Web container manages these connections through transport channels and assigns each request to a thread from the Web container thread pool.

Thread pool

The Web container maintains a thread pool to process inbound requests for resources in the container (that is servlets and JSPs). Checking the Allow thread allocation beyond maximum thread size box on the Thread Pool Configuration page allows for an automatic increase of the number of threads beyond the maximum size configured for the thread pool. As a result of this, the system can become overloaded because too many threads are allocated.

HTTP transport channel Maximum persistent requests

The maximum persistent requests is the maximum number of requests allowed on a single keep-alive connection. This parameter can help prevent denial of service attacks when a client tries to hold on to a keep-alive connection.

HTTP transport channel Read timeout

Specifies the amount of time, in seconds, the HTTP transport channel waits for a read request to complete on a socket after the first read request occurs. The read being waited for could be an HTTP body (such as a POST) or part of the headers if they were not all read as part of the first read request on the socket.

Application assembly performance checklist

Enterprise bean modules:

- Entity EJBs - Bean cache
- Method extensions - Isolation level
- Method extensions - Access intent
- Container transactions

Web modules:

- Web application - Distributable
- Web application - Reload interval
- Web application - Reload enabled
- Web application - Web components - Load on startup

Managing connections

The goal is to minimize the number of connections needed for an end-to-end system, as well as to eliminate the overhead of setting up connections during normal operations. To reduce the overhead associated with establishing connections between each layer, a pool of pre-established connections is maintained and shared among multiple requests flowing between the layers.

For instance, WebSphere Application Server provides database connection managers to allow connection reuse. It is important to note that a session may use multiple connections to accomplish its tasks, or many sessions may share the same connection. This is called connection pooling in the WebSphere connection manager.

Another pool of connections is at the Web container level. The application server environment pre-allocates this pool to allow for quick access to requests. Once a request has been satisfied, the container releases the connection back to the pool. There are configuration options for allowing this pool to grow beyond the maximum, and to limit how far beyond that it can grow. These, however, can create additional memory requirements. Care should be taken when adjusting the parameters related to the Web container thread pool.

The key issue is with maintaining a session's identity when several sessions share a connection. Reusing existing database connections conserves resources and reduces latency for application requests, thereby helping to increase the number of concurrent requests that can be processed. Managing connections properly can improve scalability and response time. Administrators must monitor and manage resources proactively to optimize component allocation and use.

Dynamic Cache

The Dynamic Cache Service improves performance by caching the output of servlets, commands and Java Server Pages (JSP) files. WebSphere Application Server consolidates several caching activities, including servlets, Web services, and WebSphere commands into one service called the dynamic cache. These caching activities work together to improve application performance, and share many configuration parameters, which are set in an application server's dynamic cache service. The dynamic cache works within an application server Java Virtual Machine (JVM), intercepting calls to cacheable objects, for example through a servlet's service() method or a command's execute() method, and either stores the object's output to or serves the object's content from the dynamic cache. Because J2EE applications have high read/write ratios and can tolerate small degrees of latency in the currency of their data, the dynamic cache can create an opportunity for significant gains in server response time, throughput, and scalability.

Policy files used to store caching rules (cachespec.xml). This provides performance enhancements by reusing dynamic application data. Application Server service used to temporarily store dynamically created J2EE application content. There is an API available to programmatically. You enable the service then define policies. You will need to write code within the application when you want to cache WebSphere command objects or Java objects. WebSphere command objects can be database queries or file system lookups. The cachespec.xml file defines how an item should be cached, what it is dependent on and when it should be invalidated. File is located in either or both:

- $PROFILE_HOME/properties/ - This is server wide
- Web module WEB-INF directory or EJB META-INF directory. This is the recommended way to configure this.

There is a dynamic cache PMI module to monitor the key areas of the dynamic cache service.
You will need to install cachemonitor.ear and to access it point a browser to http://$HOST_NAME:$PORT/cachemonitor
The following data can be cached:

- Servlets/JSP/Struts/Tiles
- WebSphere command objects
- Web Services responses
- Java Objects

http://www.research.ibm.com/journal/sj/432/bakalova.pdf

Use Integrated Tivoli Performance Viewer/Runtime Advisor to gather information about resources and analyze results

PMI externalzies performance data. Performance is collected on:

- Customer's application resources (EJB's, Servlets/JSPs, Web Services)
- Application Server's run-time resources (JVM memory, thread pools, database connection pools)
- System Rrsources (CPU usage, disk usage, free memory)

Implements J2EE 1.4 Performance Data Framework

Tivoli Performance Viewer and Performance Advisors allow you to analyze and visualize PMI data. Displays data collected from local/remote Application Servers. Also provides configuration advice via Performance Advisor section. This is now integrated into the Administrative.

Performance Advisors provide configuration advice based on collected PMI data. Advice is based on basic rules of thumb for tuning WAS. Administrator must manually apply recommendations.

You can monitor garbage collection statistics using object statistics in the Tivoli Performance Viewer or the verbose:gc configuration setting.
Default value: NewSize=2m, MaxNewSize=32m, SurvivorRartio=2
Recommended value: -XX:newSize=640m -XX:MaxNewSize=640m -XX:SurvivorRatio=16 (for JVM with more then 1 GB heap size) or set 50 to 60% of total heap size to new generation pool.

Performance Monitoring Infrastructure collects performance data from a running application server.
PMI data falls into three categories:

- Application resources - EJB's, Servlets, Web Services
- Run-time resources - JVM memory, thread pools, database connection pools.
- System resources - CPU, Free memory

From the data, WAS v6 organizes PMI metrics into categories:

- Basic - J2EE components, processor usage, HTTP session info
- Extended - Basic + Application server resources (WLM, Dynamic Cache)
- Custom - Fine-grained control

The PMI service is enabled by default in v6, at the Basic setting.

Tivoli Performance Viewer
TPV collects and visualizes performance data and configuration advice via the Performance Advisor section.

Monitoring and Tuning > Performance Viewer > CUrrent Activity
server1 > Advisor
Settings
User
Log
Summary Reports
Servlets
EJB's
EJB Methods
Connection Pool
Thread Pool
Performance Modules
SIB Service
Enterprise Beans
Dynamic Caching
JDBC Connection Pools
HA Manager
JCA Connection Pools
JVM Runtime
Object Pool
ORB
Servlet Session Manager
System Data
Thread Pools
Transaction Manager
Web Applications
Web Services
Web Services Gateway

PMI settings for a server are accessed through:
Monitoring and Tuning > Performance Monitoring Infrastructure > $SERVER_NAME> Runtime
The performance advisors should be enabled once the peak load has been reached, and not during the ramp up time.

Examples of advice that Performance Advisor would give in certain situations:
Data Sources
Situation: Prepared statement discard rate is too high and heap space is available.
Advice: Increase state cache size

Thread Pools (ORB, Web Container, Data Source)
Situation: If number of connections is low (= to the min)
Advice: Decrease pool size.

Situation: If all DS connections are heavily used and heap space is available.
Advice: Increase max pool size

Situation: Size of the pool is fluctuating a lot (high variance), possibly indicating batch processing and wasted resources
Advice: Decrease Pool Size

JVM Heap Size
Situation: If heap is too small
Advice: Increase the pool size

Unbounded Thread Pools
Situation: Threads added to an unbounded pool are not pooled.
Advice: If the average number of threads is higher than the pool size, then the pool should be increased in order to allow better pooling.

Sessions
Situation: Read/Write time/size is too large
Advice: Warn of application problem

Situation: Number of live session is greater than the session cache and memory is available.
Advice: Increase session cache

Situation: Requests turned down because there is no room for new sessions.
Advice: There are either too many active sessions or the cache size is too low.

Request metrics provide data about each individual transaction, correlating this information across the various WebSphere components to provide an end-to-end picture of the transaction. The transaction flow with associated response times can help you target performance problem areas and debug resource constraint problems. The flow can help determine if a transaction spends most of its time in the Web server plugin, the web container, the EJB container or the backend database. The response time that is collected for each level includes the time spent at that level and the time spent in the lower levels. Once you have isolated problem areas, use request metrics filtering mechanism to focus specifically on those areas.

Tune data source configuration (e.g., connection pooling, timeouts, etc.)

Connection pool settings

Scope	Cell	Node	Server	Connection Timeout
Max Connections	Min Connections	Reap Time	Unused Timeout	Aged Timeout
Purge Policy				

The datasource connection pool is used for direct JDBC calls within the application, as well as for enterprise beans using the database. Better performance is generally achieved if the value for the connection pool size is set lower than the value for the Max Connections in the web container.
Deadlock can occur if the application requires more than one concurrent connection per thread, and the database connection pool is not large enough for the number of threads. To prevent deadlock, the value set for the database connection pool must be at least one higher than the number of waiting threads in order to have at least one thread complete its second database connection.

Database connections will only return to the connection pool after timeout. New connections that have waited too long throw a connectionWaitTimeoutException(). A connectionWaitTimeoutException() is thrown due to waiting on a full connection pool. Poorly-written applications often do not properly release database connection, they forget to call connection.close(). To enable connection leak diagnostics set the log detail level to ConnLeakLogic=finest.

A connectionWaitTimeoutException() can be thrown when:

- The maximum number of connections for a given pool is set too low.
- The connection timeout is set too low.
- The application does not close some connection, or it returns connections back to the pool at a very slow rate.

Poor database or network performance often indicated by:

- Many used connections
- Many threads waiting for database connection pool
- Poor overall response time

The prepared statement cache size setting defines the maximum number of prepared statements cached per connection. The more prepared statements your application has, the larger the cache should be.

ORB Thread pool acts as a queue for incoming requests
Break CMP enterprise beans into several enterprise bean modules during assembly.

Configure class loader parameters

Classloaders

Each class loader has a delegation (search) mode that may or may not be configurable.
When searching for a class, a class loader can search the parent class loader before it looks inside its own loader or it can look at its own loader before searching the parent class loader.
Delegation algorithm does not apply to native libraries.

Delegation values:

- PARENT_FIRST - Delegate the search to the parent class loader FIRST before attempting to load the class from the local class loader.
- PARENT_LAST - First attempt to load classes from the local class path before delegating the class loading to the parent class loader. This allows an application class loader to override and provide its own version of a class that exists in the parent class loader.

Whenever the class successfully loads, the JVM caches the class and associates the class to its class loader.
Native libraries are non-Java code used by Java via the JNI interface. They are platform specific files, for example .dll in windows, and .so or .a in unix. The JVM uses the caller's class loader to load the native library. If that fails, it then uses the JVM System class loader. If both fail to load, an UnsatisfiedLinkError will result.
Application Module class-loader policy options - defined globally at Application Server level

- Single - All applications share a single application module class loader. No application isolation.
- Multiple(default) - Each application gets its own application module class loader - provides application isolation.

Web Module class-loader policy - defined on per Application

- Application - All application web modules are loaded by the application module class-loader.

- Module(default) - Each application web module will have its own class-loader, and they are children of the application module class-loader.

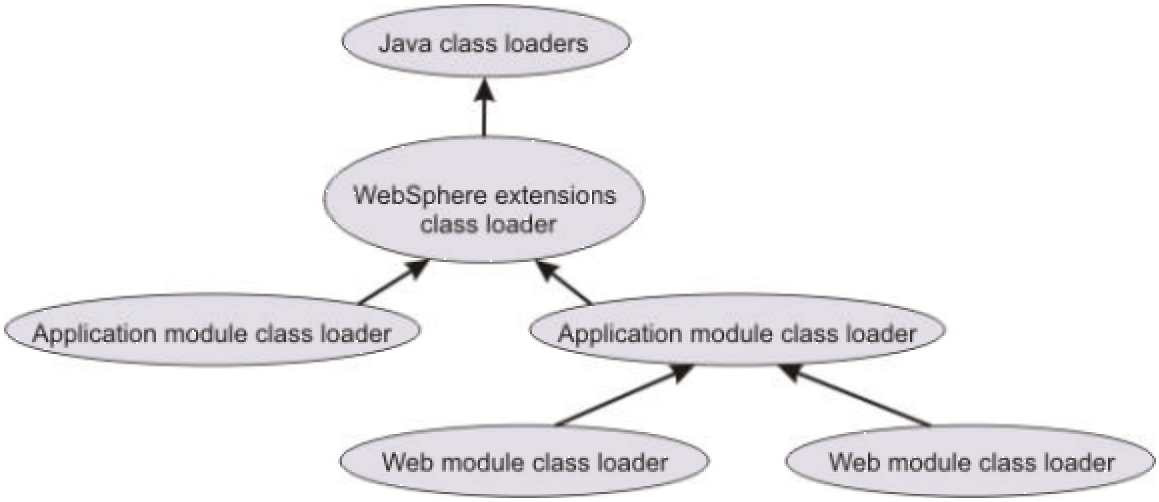
JVM class loading violations:
- ClassNotFoundException
- NoClassDefFoundError

Class preloading speeds up the startup of the Application Server process. The 1st time the application server process starts up, the name of each class loaded and the name of the JAR file containing the class are written to a preload file. By default class preloading is on, and to turn it off -Dibm.websphere.preload.classes=false to the generic JVM properties. The preload file is located in $WAS_HOME/logs/preload and the preload file can be deleted from here to be regenerated on next startup of the JVM.

Common problems with class loaders:
- ClassCastException - Duplicate class loaded by two different class loaders and have dependencies
- ClassNotFoundException - Classes not in the classpath; Loaded classes that have dependencies that are lower in the hierarchy.

Single application class loader policy - Each Application EJB's, RAR's, Dep JAR's can reference other classes in other Applications. Cannot start and stop individual Applications without effecting others.
Multiple application class loader policy - has an advantage of being able to restart each application without effecting others. Classes in one EAR cannot reference JARs in another EAR.

The runtime environment of WebSphere Application Server uses the following class loaders to find and load new classes for an application in the following order:
- The bootstrap, extensions, and CLASSPATH class loaders created by the Java virtual machine
- A WebSphere extensions class loader
- One or more application module class loaders that load elements of enterprise applications running in the server
- Zero or more Web module class loaders.



If the class loaders that load the artifacts of an application are not configured properly, the JVM might throw a class loading exception when starting or running that application. Class loading exceptions describes the types of exceptions caused by improperly configured class loaders and suggests ways to use the Class Loader Viewer to correct configurations of class loaders. The types of exceptions include:
- ClassCastException
- ClassNotFoundException
- NoClassDefFoundException
- UnsatisfiedLinkError

Section 7 - Problem Determination Applications (15%)

Configure, review and analyze logs (e.g., web server, IBM WebSphere Application Server, V6.0, etc.)
WebSphere v6 uses a standard Java logging package: java.util.logging.

Begin troubleshooting by trying to isolate the location of the problem. Test individual components and verify which are working as expected. This should help determine where the failure is.
The advanced log format is available as an output format for the trace log and system out log. The thread name is now included in this format to enable easier correlation with other types of diagnostic data.
The Log Analyzer tool formats the activity log and provides a structured way to look at problems and messages. With AST you can also analyze Log and Trace.
Log files:
<was_root>/profiles/profileName/logs/ServerName/

- SystemOut.log and SystemErr.log – Standard JVM output and error log.
- startServer.log and stopServer.log – startup and shut down of the application server.
- Activity.log – Events that show a history of activities – Use log analyzer to read output from this file.
- Trace.log – Output from diagnostic trace. Destination and name are configurable.
- http_plugin.log – Located on web server - <plugin_root>/logs/<webserver_name>

Logs are self-managing, they can roll over based on time or file size. Number of historical log files is now configurable.

The DM must be running to add or remove a node. The nodeagent must be running to remove a node from a cell.
Log files will be in the node profile's log directory:
$WAS_PROFILE/logs/addNode.log
$WAS_PROFILE/logs/removeNode.los

Found in $PROFILE_HOME/logs/

Group	Log Files	Description
Service log	Activity.log	Binary log contains data from each JVM in a node for analysis using Log Analyzer
JVM los	SystemOut.log, SystemErr.log	Contains all messages sent to the Java System.out and System.err buffers
Native process logs	Native_stderr.log, native_stdout.log	Contains messages sent to stdout and stderr from native code segments, including the JVM
Embedded HTTP server logs	http_access.log, http_error.log	Contain all requests to the embedded HTTP server
HTTP server plug-in log	http_plugin.log	Contains data about the operation of the HTTP server plugin module
Command-line program logs	startServer.log, addNode.log	Contain data about the execution of individual command-line utilities

Installation Logs - Found in $WAS_HOME/logs

- log.txt - Main installation log
- ihs_log.txt - Main IBM HTTP installation log
- WASPreUpgrade.log - Log actions of the migration tools that backup and restore existing configuration
- WASPostUpgrade.log - Log actions of the migration tools that backup and restore existing configuration

When enabling trace, configuration tab will take effect at the next server startup. The Runtime tab effects the currently running server. The runtime tracing is stored in memory buffer and performance is not effected as much as when constantly being written to disk.

The Thread Analyzer gathers and analyzes thread dumps from an application server. This can be used to analyze performance bottlenecks due to either configuration or application problems. It can be used to determine if threads are being blocked by monitors. It can also determine percentage of work being done on the application server. In addition it can provide a recommendation based on analysis.
Thread analyzer can be downloaded from:
http://www.ibm.com/developerworks/websphere/downloads/thread_analyzer.html

Java Heap Dumps print a record of all objects in the Java heap to a text file. Garbage Collection is forced before a dump. HeapRoots is a free tool for reading heap dumps and can be downloaded at:
http://www.alphaworks.ibm.com/tech/heaproots

IBM diagnostics guide for the IBM Java SDK 1.4.1 can be downloaded from:
http://www.ibm.com/developerworks/java/jdk/diagnosis/diag141sr1.pdf

HeapWizard, a Graphical analysis of Java Heap dumps can be downloaded from:
ftp://ftp.software.ibm.com/software/webspehre/info/tools/heapwizard

ClassLoaderViewer, an admin console plugin for displaying the classloader hierarchy, can be downloaded from:
http://www.ibm.com/developerworks/websphere/library/techarticles/0312_cocasse/0312_cocasse.html

Use Log Analyzer tool to identify problems
The Log Analyzer takes one or more service or activity logs, merges all of the data, and displays the entries. Log Analyzer has a special feature enabling it to download the latest symptom database from the IBM Web site.
In cases where transferring the file is impractical or inconvenient, use the alternate viewing tool, showlog, to view the service or activity log file.

The log analyzer looks at error conditions in the activity log entries to diagnose problems and suggest possible solutions. Requires the symptom database to be updated from the Internet. There is also a log analyzer component as part of the AST. You start the log analyzer from the <profile_root>/bin/waslogbr.sh
The activity.log is located on: <was_root>/profiles/profileName/logs directory.
Load activity.log, Run analysis on it. Any records flagged with a checkmark have a corresponding record in the symptom database.
Raw data is formatted into Units of Work. Units of Work contain individual lines of detailing activity.
Analyzing log records means to compare log records using specified symptom databases that are loaded into memory. The solution is reported in the analysis result pane, giving advice on resolving the reported problem.

The Log Analyzer tool is used to view the activity.log. This file contains audit, warning, and fatal messages pertaining to the WebSphere environment. From the admin console navigate to Troubleshooting – Logs and Trace – server1 – IBM Service Logs. You can disable the service log and Maximum file size (In MB).
There is one service log for all WebSphere Application Server Java virtual machines (JVMs) on a node, including all application servers and their node agent (if present). A separate activity log is created for a deployment manager in its own logs directory.

Log Analyzer is a GUI for the activity.log. The log analyzer compares error messages to a database of known problems. The database can be updated by downloading the latest symptom database from IBM.

JVM logs are managed by the runtime. The can be configured for time-based and/or size-based rollover. You can configure how many number of historical logs to keep.

The collector tool is used to gather data for IBM support, intended to reduce the number of round trips between customers and IBM. Gathers all relevant data about WebSphere and the environment and collects it into a jar file. Create a temporary directory outside of WebSphere and make it the current working directory. This will produce the output working directory. Must be run as root and have Java 1.2.2 or higher in the path.

Use trace facility (e.g., activating, selecting criteria, starting/stopping, JRAS, etc.)
Enabling Trace
Trace for an application server process is enabled while the server process runs by using the administrative console. You can configure the application server to start in a trace-enabled state by setting the appropriate configuration properties. You can only enable trace for an application client or stand-alone process at process startup.
A Trace string is passed to the trace service. (com.ibm.ejs.ras.*=all=enabled).
You can apply changes to a running server.
Use Scope settings to filter the contents of an administrative console collection table to a particular cell, node, or server

Trace can be started:
- While server is running using Runtime Diagnostic Trace
- When server is started using Configuration Diagnostic Trace

Trace output can be directed to
- Memory ring buffer – dumped after trace stops.
- File

Log Detail Level is set to *=info by default, this has no trace output.
Trace strings have been moved to a separate panel in the console(Log Detail Level)
AST can be used to analyze trace output.
Log Levels control which events are processed by Java Logging. Log detail level affects tracing and regular logging. Setting levels below info reduces the amount of data in logs.
*=off disables logging altogether. Trace levels are Fine, Finer, Finest and do not appear in the trace file unless logging in enabled. Therefore, if you do not enable diagnostic trace, setting the log level detail to Fine, Finer, or Finest does not have an effect on the data that is logged.

Timestamps are real machine time values. Events prior to an exception are probable causes. Events after exception are recovery attempts.

Reading a log/trace file

	Entry to a method (debug)
<	Exit a method (debug)
A	Audit
W	Warning
X	Error
E	Event (debug)
D	Detail (debug)
T	Terminate (exits process)
F	Fatal (exits process)
I	Information
O	Program Output
C	Configuration

Timestamp. Thread id, component, message type, Message

Filter and review console messages
Use the WebSphere Status area of the administrative console to view error and run-time event messages returned by WebSphere Application Server.

The WebSphere Status area displays along the bottom of the console and remains visible as you navigate from the WebSphere Home page to other pages. The area displays two frames: WebSphere Configuration Problems and WebSphere Runtime Messages. Click Previous or Next to toggle between the frames.

Click the icon in the upper-right of the area to refresh the information displayed. You can adjust the interval between automatic refreshes in the Preferences settings.

WebSphere Configuration Problems
Displays the number of workspace files. This frame also displays the number of problems with the administrative configuration for the user ID.
Click on the number to view detailed information on the problems.

WebSphere Runtime Message
Displays the number of messages returned by WebSphere Application Server as well as the number of error messages(x icon), warning messages (! icon), and informational messages (i icon).

Use First Failure Data Capture (FFDC) Tool
The First Failure Data Capture tool preserves the information generated from a processing failure and returns control to the affected engines. The captured data is saved in a log file for use in analyzing the problem.
It runs as part of WAS and does not effect performance.
There are no administrative tasks to manage FFDC.

The FFDC configuration properties files are located in the properties directory under the WebSphere Application Server product installation. There are three properties files, but only the ffdcRun.properties file should be modified. You can set the ExceptionFileMaximumAge property to configure the amount of days between purging the FFDC log files. The value of the ExceptionFileMaximumAge property must be a positive number. The FFDC feature does not affect the performance of the WebSphere Application Server product.

The Collector tool collects information about the WAS installation and configuration so that it may be sent to IBM support for analysis
Packages information in JAR file
JAR file is sent to IBM support
Run collector from the app_server_root/bin/ directory:
app_server_root/bin/collector.sh -profileName profile_name
This script must be run as root user, or administrator user on a windows machine.
The collector program creates the Collector.log log file and an output JAR file in the current directory.

The name of the JAR file is composed of the host name, cell name, node name, and profile name:
host_name-cell_name-node_name-profile_name.JAR

Use the JNDI dumpNameSpace utility
It is often helpful to view a dump of the name space to understand why a naming operation is failing.
You can use the dumpNameSpace tool to dump the contents of a name space accessed through a name server.
The JNDI dumpnamespace utility is useful to ensure correct association of named objects:

- Data sources
- EJB's
- JMS resources
- Other resources

./dumpNameSpace.sh –bootstrap_port
./dumpNameSpace.sh –factory com.ibm.websphere.naming.WsnInitialContextFactory
The dumpNameSpace tool only dumps the objects that are in the local namespace of the process against which the command was run.
If the object a client needs to access does not appear, use the administrative console to verify that:

- The server hosting the target resource is started.
- The Web module or EJB container, if applicable, hosting the target resource is running.
- The JNDI name of the target resource is correct and updated.

If the problem resource is remote, that is, not on the same node as the Name Server node that the JNDI name is fully qualified, including the host name.
View detailed information on the runtime behavior of the Naming service by enabling trace on the following components and reviewing the output:

- com.ibm.ws.naming.*
- com.ibm.websphere.naming.*

If you see an exception that appears to be CORBA related ("CORBA" appears as part of the exception name) look for a naming-services-specific CORBA minor code, further down in the exception stack, for information on the real cause of the problem.
For a list of naming service exceptions and explanations, see the class
com.ibm.websphere.naming.WsnCorbaMinorCodes in the API documentation.

Name Space Bindings are special user-defined bindings
Basic components of WAS Naming are:

- Global Name Space
- WAS JNDI Provider
- EJB and Resource References
- Java:comp/env names
- Handling of local EJB's
- Administrative Name Space Bindings

Naming Problem Determination

- ./dumpNameSpace -port {BOOTSTRAP_PORT} -startAt eis
- ./dumpNameSpace -port {BOOTSTRAP_PORT} -startAt jdbc
- ./dumpNameSpace -port {BOOTSTRAP_PORT} -startAt nodes/Node1/servers
- ./dumpNameSpace -port {BOOTSTRAP_PORT} -report long

Perform troubleshooting tasks (e.g., thread dump, JVM core dump, and remote debugging, etc.)
IBM Heap Dump is created in <profile_root> directory when an out of memory exception is thrown. This can be disabled with and environment variable: -IBM:HEAPDUMP_OUTOFMEMORY=false;
To enable heap dump add the following environment variable: IBM_HEAPDUMP=true;
Heap dump is in .phd (portable heap dump) format. To access heap dump you can use HeapRoots
IBM_JAVA_HEAPDUMP_TEXT=true formats heap dump as text (classic heap dump format).
SIGQUIT (kill -3) also generates heap dump.
Using wsadmin to generate the dump:

- Set jvm [$AdminControl completeObjectName type=JVM,process=<server_name>,*]
- $AdminControl invoke $jvm dumpThreads

Application threads can hang for a number of reasons, including infinite loops or deadlocks.
There is a component known as the ThreadMonitor that monitors the web container, ORB and Async Bean thread pools for hung threads. The thread monitor doesn't try to deal with the hung threads. It issues notifications so that the administrator and/or developer can deal with the issues. When a hung thread is detected, three notifications are sent: a JMX notification for JMX listeners, PMI thread pool data is updated for tools like the Tivoli performance viewer, and a message is written to the SystemOut.log. When the thread pool gives work to a thread, it notifies the thread monitor. Thread monitor notes thread ID and timestamp. Thread monitor compares active threads to timestamps. Threads active

longer than the time limit are marked potentially hung. Performance impact is minimal. Hung thread detection is enabled by default, a thread must be active for at least 10 minutes before it is marked as potentially hung. Two custom properties can be configured for Hung thread detection:
In the admin console go to Servers – Application servers – server1 – Server Infrastructure – Administration – Custom

- Com.ibm.websphere.threadmonitor.interval 20
- Com.ibm.websphere.threadmonitor.threshold 120

You can adjust the thread monitor settings by using the wsadmin scripting interface. These changes take effect immediately, but do not persist to the server configuration , and are lost when the server is restarted.

Poorly written applications often do not properly release database connections. Connections should be closed in the finally { } block and the connection.close( ) method should be called.
Orphaned connections only return to the pool after they time-out. This can cause a backup of connections waiting for old connections to time out. New connections that have waited too long throw a connectionWaitTimeoutException.
WebSphere is smart enough to eventually time out orphaned connections and return them to the pool, but for an application that makes frequent use of database connections, this might not be enough. New connections can get queued up waiting for the database while old connections are waiting to be timed out. This can cause connection Wait exceptions.
Connection leaks have traditionally been hard to diagnose because the error messages do not usually provide specific enough information about the source of the problem. Usually a source code review is needed to find points in the code where connections are not properly closed.

Connection Manager Diagnostics – When activated, enables a connection manager wrapper that holds the stack trace of all getConnection( ) calls in a throwable object. When an exception is thrown due to waiting on full connection pool, print stack traces of all open connections. This has a lower performance impact than the connection manager tracing.
When a thread times out waiting on a connection from a full connection pool, it throws a connectionWaitTimeoutException. When this exception is thrown, the wrapped prints out the stack traces for every open connection. This feature is useful because it shows you the call stacks for all open connections at the time of the exception. This enables you to significantly narrow your search area when you look at the application's source code to try and find the responsible code.

Connection Leak Diagnostics – Enabled using a standard trace string: -ConnLeakLogic=finest. You must enable Log in the Diagnostic Trace Service page in the admin console.
To set this trace string using Jython scripting:

1. Identify the server and assign it to the server variable:
   Server=AdminConfig.getid('/Cell:MyCell/Node:mynode/Server:server1/') print server
2. Identify the trace service belonging to the server and assign it to the tc variable:
   Tc=AdminConfig.list('TraceService', server) print tc
3. Set the trace string: AdminConfig.modify(tc, [['startupTraceSpecification', '*=info:ConnLeakLogic=finest']])

Some basic troubleshooting procedures:
Verify the server is running: <profile_root>/profile1/bin/ serverStatus.sh server1
You can log into the admin console and navigate to Troubleshooting – Logs and Trace – server1
If you select JVM logs you can configure:

- Log file location
- Formatting
- Log file rotation – Size (In MB) or Time (start time and repeat time) and Maximum number of historical log files.

Any changes need to be saved to the master configuration and the JVM restarted.

The ConnectionWaitTimeout exception indicates that the application has waited for the number of seconds specified by the connection timeout setting and has not received a connection. This situation can occur when the pool is at maximum size and all of the connections are in use by other applications for the duration of the wait. In addition, there are no connections currently in use that the application can share because either the connection properties do not match, or the connection is in a different transaction.

IllegalConnectionUseException
This error can occur because a connection obtained from a WAS40DataSource is being used on more than one thread. This usage violates the J2EE 1.3 programming model, and an exception generates when it is detected on the server. This problem occurs for users accessing a data source through servlets or bean-managed persistence (BMP) enterprise beans

ConnectionWaitTimeoutException accessing a data source or resource adapter
some possible causes are:
The maximum number of connections for a given pool is set too low.
Connection Wait Time is set too low. Current demand for connections is high enough such that sometimes there is not an available connection for short periods of time.
You are not closing some connections or you are returning connections back to the pool at a very slow rate. This situation can happen when using unshareable connections, when you forget to close them, or you close them long after you are finished using them, keeping the connection from returning to the pool for reuse.
You are driving more load than the server or backend system have resources to handle. In this case you must determine which resources you need more of and upgrade configurations or hardware to address the need. One indication of this problem is that the application or database server CPU is nearly 100% busy.

To correct these problems, either:
Modify an application to use fewer connections
Properly close the connections.
Change the pool settings of MaxConnections or ConnnectionWaitTimeout.
Adjust resources and their configurations.

com.ibm.websphere.ce.cm.StaleConnectionException: [IBM][CLI Driver] SQL1013N The database alias name or database name "NULL" could not be found. SQLSTATE=42705
This error occurs when a data source is defined but the databaseName attribute and the corresponding value are not added to the custom properties panel.

java.sql.SQLException: java.lang.UnsatisfiedLinkError:
This error indicates that the directory containing the binary libraries which support a database are not included in the LIBPATH environment variable for the environment in which the WebSphere Application Server starts.

"J2CA0030E: Method enlist caught java.lang.IllegalStateException" wrapped in error "WTRN0063E: An illegal attempt to enlist a one phase capable resource with existing two phase capable resources has occurred" when attempting to execute a transaction.
This error can occur when last participant support is missing or disabled. last participant support allows a one-phase capable resource and a two-phase capable resource to enlist within the same transaction.

java.lang.UnsatisfiedLinkError:xaConnect exception when attempting a database operation
This problem has two main causes:
The most common cause is that the jdbc driver which supports connectivity to the database is missing, or is not the correct version, or that native libraries which support the driver are on the system's path.

To determine which JDBC driver you are running:
java com.ibm.db2.jcc.DB2Jcc –version
Hung thread detection
When the thread pool gives work to a thread, it notifies the thread monitor. The thread monitor notes thread ID and timestamp. The thread monitor compares active threads to timestamps. Threads active longer than the time limit are marked "potentially hung". Performance impact is minimal < 1%. If a thread previously reported to be hung completes its work, a notification is sent. The monitor has a self-adjusting system to make best effort to deal with false alarms. To configure hung thread detection create custom properties on the application server:

- com.ibm.websphere.threadmonitor.interval = 180 seconds
- com.ibm.websphere.threadmonitor.threshold = 600 seconds
- com.ibm.websphere.threadmonitor.false.alarm.threshold = 100 number of flase alarms

A Java thread dump will show the stack traces for all threads. Find the hung threads using the thread ID, use the stack trace to investigate the cause of the hang.

IBM Heap Dump
Prints a record of all objects in the Java Heap to a text file.
Displays size, address, and references for each object.
Garbage Collection is forced before a dump.
Compare heap dumps over time to determine if memory is leaking.

IBM heap dump is created in $WAS_PROFILE when an OutOfMemoryException is thrown. This can be disabled with an environment variable: IBM_HEAPDUMP_OUTOFMEMORY=false
To generate an IBM Heap Dump add IBM_HEAPDUMP=true for a .phd format or IBM_JAVA_HEAPDUMP_TEXT=true for the classic heap dump format. A SIGQUIT (kill -3) generates a heap dump in $WAS_PROFILE. One can also be triggered via wsadmin:
> set jvm [$AdminControl] completeObjectName type=JVM,process=serverName, *]
> $AdminControl invoke $jvm dumpThreads

New version of memory tools support .phd
IBM RAD V6
HeapRoots v2.0.5
HeapAnalyzer

Database Connection Leaks
Poorly-written applications often do not properly release database connections

- Forget to call connection.close()
- Most often in the exception case
- Connections should be closed in a finally{} block

Connections will only return to the pool after timeout
Can cause a backup of new connections waiting for old connections to time out
New connections that have waited too long throw a connectionWaitTimeoutException

When diagnostic functionality is activated, enables a connection manager tracer that holds the stack trace of all getConnection() calles in a throwable object.
When a connectionWaitTimeoutException is thrown due to waiting on a full connection pool, print stack traces of all open connections.
Lower performance impact than connection manager tracing
Enabled using a standard log detail level string: ConnLeakLogic=finest

Section 8 - Systems Administration Tasks (*covers one time administrative tasks) (7%)

Perform cell administration (e.g., deployment manager, federate nodes; create clusters etc.)
DMgr and Node Agent uses system Filetransfer application to transfer the file over Http(s) Updates are pulled by the Node Agent during file synchronization Only one-way synchronization: Changes saved at the DMgr level are propagated down Uses HTTPs if security is turned on. Default file sync interval is 60 seconds.
At synchronization time, Node agent sends document digests to the DMgr, where they are compared against the digests in the master repository, consisting of:

- Configuration files for all processes
- J2EE Application (Deployment Descriptors files and binaries)

Only changes made by administrative clients will be marked for synchronization

File Synchronization Process in a ND Cell:

1. Node Sync initiates synchronization operation
2. Cell Sync reads master configuration repository and compares to node copy information
3. Cell Sync returns update information and files
4. Node Sync writes updates to local configuration files (all changes at once)

Deployment Manager, Node Agents & Application Server can start in any order with the following exception
Node Agent must start before any application server on that node Reason: Node Agent hosts the Location Service Daemon that is needed by the application server and application clients.

You can add a node from the DM admin console or from the new node by running addNode.sh. You can pass the -includeapps to include the applications. By default the application are not installed.
addNode.bat(sh) <dmgr_host> [dmgr_port] [options]

- Host name of the deployment manager is required
- Dmgr_Port defaults 8879, the SOAP connector port

Once this is run from the node, the wsadmin scripts should be updated to reflect the new cell name. The http plugin xml file should be changed to point to the plugin file in the DM master configuration.
The node will inherit security settings from the DM.
The administrative console and wsadmin are the two ways that the environment is administered. These tools talk directly to the DM, and not the application servers directly. The communication of these commands flows from the tools to the DM to the node agents, to the application servers. This allows administration of multiple nodes from a single focal point (The DM).

<u>Resource Scope</u>
Version 5

- Cell, node, application server layer

By default the system will look at the server layer first for resources, then it checks the node layer next and finally the cell layer.

New in Version 6 is the cluster layer. Resources available to all cluster members. New cluster members automatically gets the cluster scoped resources.

A variable defined at the finest scope takes precedence.

A stand-alone application server master repository will not have the following files, that a cell would ahve:

- Cluster Definition
- DM node or other created nodes
- Node Agent or other application servers.

A stand-alone node application server repository is similar to a Cell level Master repository.

Cell Directories:

- config/cells/cellName/
  - applications/
  - buses/
  - clusters/

coregroups/
nodegroups/
nodes/

Cell Files:

- admin-authz.xml - Authorization for Administration
- cell.xml - defines cell information. like discovery protocol
- corebridge.xml - Core group bridge (communication between core groups) definitions.
- filter.xml - Java 2 Security filters so as to prevent permissions to key resources.
- multibroker.xml - DRS settings
- namestore.xml - Cell wide persistent info of namespace
- naming-authz.xml - Authorization for naming
- pmirm.xml - Performance monitoring instrumentation
- resources.xml - cell wide resource definitions (JDBC, JMS, etc)
- security.xml - Global security.xml
- variables.xml - cell wide env variable definitions
- virtualhosts.xml - global virtualhosts - apps are ties to a specific virtualhost
- ws-security.xml - default cell wide global web services security bindings

Node Files:

- app.policy, library.policy, spi.policy - Java 2 Security policy files for apps, shared libraries, J2C resources
- installedChannels.xml - defines channel (communication protocols like TCP, SSL, HTTP, HTTP tunnel) defined for the webspehre processes.
- namestore.xml - persistent JNDI definitions for entire node.
- node.xml - defines node info, like name, discovery protocol
- node-metadata.properties - properties of the node - product version, OS
- perftuners.xml - performance tuning rules
- resources.xml - node level resource definitions (JDBC, JMS)
- resourcesPME.xml - node level PME resource definitions
- serverindex.xml - defines all servers in the node, the server type and its ports
- variables.xml - node level env definitions

Application Server Files:

- hamanagerservice.xml - defines the core group that the server belongs to for HA, and defines HA fields.
- pmi-config.xml - Performance monitoring instrumentation
- resources.xml - server level resource definitions (JDBC, JMS)
- server.xml - defines server info, like channels, components and services, JVM process definition
- server-pme51.xml and server-pme.xml - PME definitions
- sib-authorizations.xml - SIB authorization for the server
- sib-engines.xml - SIB messaging engine definition
- sib-service.xml - SIB services - enabled or not
- variables.xml - server level variable definitions
- ws-security.xml - server level web services security bindings - overrides cell level identical bindings.

Dmgr and Node Agent use the Filetransfer application to transfer the configuration files over HTTP(S) for synchronization.

A benefit to enabling the 'Startup synchronization' option is that the server starts up with the latest configuration files. A full synchronization means that a full comparison is done, and not that all files are pushed. Only changed files are synchronization.

A synchronization can be done in one of three ways, using wsadmin, the administrative console, or the syncNode.sh script.
Only changes made by administrative clients will be marked for synchronization, changes made with a text editor will not be marked for automatic synchronization.

Perform administrative console and wsadmin tasks

wsadmin/Java admin client can use Soap/Http/RMI-IIOP to connect to the Admin services/JMX
The administrative console is a standard J2EE application installed during the install time.
The IBM HTTP server management and Tivoli Performance View monitor has been integrated into the admin console.
wsadmin is based on the Bean Scripting Framework (BSF).
wsadmin acts as an interface to Java objects for access by scripts: Objects communicate with MBeans (JMX management objects)
Objects perform different operations:

- AdminConfig - Enables manipulation of configuration data for a WebSphere installation. You can create/change WAS static configuration.
- AdminApp - Allows manipulation of application objects (installing, uninstalling, editing and listing). Install, modify or administer applications.
- AdminControl - Enables manipulation of MBeans running in a WebSphere server process. Works with live running objects, performs traces and data type conversion.
- AdminTask - Enables the execution of available admin commands. Access administrative commands to provide an alternative way to access configuration commands.
- Help - Provides Help info for the wsadmin scripting objects. Provide a means to obtain interface information about MBeans running in the system.

There are three ways to invoke wsadmin:
- wsadmin – interactively
- wsadmin –c command
- wsadmin –f scriptfile

wsadmin defaults to jacl scripting language; to use jython: wsadmin –lang jython

Use the AdminConfig object to:
- List configuration objects and their attributes
- Create configuration objects
- Modify configuration objects
- Remove configuration objects
- Obtain help

Use the AdminApp object to:
- Install and uninstall applications
- List installed applications
- Edit application configurations
- Obtain help

Use the AdminControl object to:
- List running objects and their attributes
- Invoke actions on running objects
- Obtain dynamic information about Mbeans that represent running objects
- Obtain help

Use the AdminTask object to access a set of administrative commands that provide an alternative way to access the configuration commands and the running object management commands. The administrative commands run simple and complex commands.
To preload a profile script: wsadmin –profile <profile_script_name>
Certain default behaviors for wsadmin can be changed by editing:
<profile_root>/<profileName>/properties/wsadmin.properties
Wsadmin –f scriptFile is faster than wsadmin –c command
If security is enabled, authentication information must be supplied for server communication. User id and password can be supplied via:
- Command Line parameters
- sas.client.props file for connecting with RMI
- soap.client.props file for connecting with SOAP

Use examples:
./wsadmin.sh –help
./wsadmin –c "\$AdminControl getPort"
./wsadmin –f <software_dir>/wsadmin/jacl/class_sample.jacl
./wsadmin.sh
wsadmin>$Help AdminApp
wsadmin>$AdminApp help edit

Do not perform backupConfig at the Node level of a cell.

syncNode
Forces full synchronization between the node and the Deployment Manager
Used as error recovery tool, in case the node level configuration is damaged to the point where the Node Agent does not start
In normal case, use the Administrative Console or wsadmin to force synchronization

cleanupNode <cellhost> <cellport> - Executed at the DMgr

In the administrative console, under nodes, the synchronize button compares node and master copies of the repository and sends changes to the node if needed. Full resynchronize re-sends a complete copy of the master repository to the node.

A Node group is a collection of WAS nodes. A node group establishes a boundary for cluster creation. All cluster members of a cluster must be on nodes that are members of the same node group. Nodes that are organized into a node group need enough capabilities in common to ensure that clusters formed across those nodes can host the same application in each cluster member. A node must be a member of at least one node group and can be a member of more than one node group. To delete a node group it must be empty.
The file transfer service for the node agent uses the http protocol.

versionInfo - Provides IBM WebSphere Application Server Version Report
genHistoryReport, genVersionReport - Generate html history/version reports
collector - Gather logs & configuration files in one .jar

PropFilePasswordEncoder and EncAuthDataFile
If you have to replace encoded passwords in config files with clear text, this tool will re-encode passwords
Syntax:
PropFilePasswordEncoder Filename, password properties list
EncAuthDataFile Input file, output file

Administrative Client Workspace
A temporary space is given to the user when making configuration changes. The xml configuration files are copied from the master repository to the workspace. WebSphere uses optimistic concurreny updates, if two users make changes, websphere will warn that the files has been updated. Files in the workspace are not removed automaticallt. This workspace is located in wstemp.
There are four security roles available:
Monitor (View only)
OPerator (View, start/stop jvms)
Configuratior (View, configure)
Administrator (View, start/stop, and configure)

The wsadmin command line scripting tool is based on Bean Scripting Framework and supports JACL and Jython languages.

The administrative console is accessible via a new port, 9060 and 9043 (SSL). A username is used to track and save user-specific configuration data. It allows recovery from unsaved previous sessions. The user workspace is located in the profile "wstemp" directory:
<$PROFILE_HOME/profileName/wstemp/,generated-hashed-name>/workspace

./startManager.sh -profileName <Dmgr_profile_name>
./startServer.sh -profileName <Dmgr_profile_name>
No need to provide the -profileName if run from the profile bin directory.

WASService.exe (Windows only) - is a command line tool that allows adding any WAS process as a windows service.
WASService.exe -add <service_name> -serverName <Server> -profilePath <servers_profile_path> <options>
WASService.exe -remove service_name
WASService.exe -start service_name
WASService.exe -stop service_name
WASService.exe -status service_name

These is a sample rc.was shell script that can be found in <WAS_ROOT>/bin
The system administrator has to manually add an entry into the inittab.

Configure WebSphere JMS providers

Configuring Buses
Service Integration > Buses

Configuring Bus Members
Service Integration > Buses > Additional Properties
A Bus Member is added to the bus and a messaging engine is created for that cluster.

Configuring Messaging Engines
Service Integration > Buses > BUS_NAME > Bus Member > BUS_MEMBER_NAME

New messaging engines can be created in clusters for scalability reasons. Messaging Engines manage messaging resources.

Messaging Properties:
- General Properties
  - Initial State - determines if the messaging engine is started with server.
  - High Message Threshold
- Message Points
  - Mediation points
  - Queue Points
  - Publication Points
- Additional Properties
  - Datastore
  - SIB link
  - WAS MQ client links
  - WAS MQ links
  - Custom properties

Configuring the Datastore
- Create the database
- Create the tables
  - can be automatically generated when the datastore is started
  - Use the SIBDDLGenerator command to get the DDL.
  - sibDDLGenerator -system db2 -version 8.1 -schema SIB -user test > DDL.txt
- Create the datasource
- Configure the Datastore

Creating Bus Destinations
- Select Queue or Topic

Service Integration Bus resource management is performed using either the Admin console or wsadmin.
- Buses
- Bus Members
- Messaging Engines
- Destinations

High Level Administration
- Administrative unit is the cell - A bus us fully contained within a cell.
- Bus Level - Manage SIB resources - Bus Members, Messaging Engines and destinations.
- Infrastructure Management -
  - Define and deploy Messaging Engines to servers or clusters
  - Associate queues with Bus Members
  - Assign persistent stores to Messaging Engines
  - Define links to another Bus, or to a WAS MQ Queue Manager.

WAS v6 provides a pure Java JMS 1.1 provider that is installed as part of the base server. It runs completely inside the application server JVM.
Persistent messages are stored either in an Embedded Cloudscape database or an external database of choice via a JDBC driver.
Each application server, or cluster can host a messaging engine. Messaging engines can be interconnected to form a messaging bus.

For WebSphere, SIBus consists of a Bus Member, Messaging engines in the server or cluster, Destinations that are linked to messaging engines. When SIBus is used for JMS applications, it is referred to as a messaging bus.

Bus Members:
Bus Members of SIBus are application servers and/or clusters on which the messaging engines are defined.
When a new bus member is defined, one messaging engine is automatically created on the corresponding application server or cluster.
For an ND cell, you can add additional ME's to a cluster to provide scalability.

Messaging Engines:
ME's run inside the application server JVM, and manage messaging resources.
Each ME has its own set of tables in a data store (JDBC database).
Queue-like destinations are associated with one or more ME's.
ME's provide a connection point for clients to put or get messages.

A default bus with a single default ME is created automatically at installation time of the standalone application server. Within each bus, each ME has a unique identity made up of bus name and server name.

Bus Destinations

A Bus Destination is a virtual place within an SIBus, to which applications attach to exchange messages.
Bus Destinations can be permanent or temporary.

- Temporary - Created and deleted automatically for API specific destinations. Created programmatically, usually to specify a JMSReplyTo destination within a message.
- Permanent - Created by an administrator and deleted only when manually deleted.

Types of Destinations

- Queue - For point-to-point messaging
- Topicspace - For publish-subscribe messaging.
- Alias - Destination that is an alias for another target destination.
- Foreign - Destination that identifies a destination on another bus.
- Exception - Destination that is used to handle messages that cannot be sent to intended bus destination.

Each ME has a default exception destination, SYSTEM.DEFAULT.EXCEPTION.DESTINATION.me_name that can be used to handle undeliverable messages for all bus destinations that are localized to the messaging engine.

Messages are transported through a bus. A bus is a virtual messaging environment that spans servers. If you want to send or receive messages through a bus you have to connect to it. Servers can be added to a bus, and in doing so you start/create a ME in that server. ME's are invisible to a customer and are noted/logged in the servers SystemOut.log (Message ID SIAS). SIBDestinations are defined on the bus, rather than on the ME. Queues are localized to a particular ME. messages sent to that Queue will be stored in a database on one particular ME within a bus.
For point-to-point messaging, the administrator selects one bus member, an application server or cluster, as the assigned bus member that is to implement the runtime state of a queue destination. This action automatically defines a queue point for each messaging engine in the assigned bus member.
For a queue destination assigned to an application server, all messages sent to that destination are handled by the messaging engine of that server, and message order is preserved.

Message Store

The message store is a sub-component of the messaging engine. This is used for in-flight messages and hold a number of other pieces on information. A message store can be either persistent or non-persistent. ME requires a persistent back-end data store, even for non-persistent messages.
Each ME has its own data store for storing messages, transaction states and delivery records.
Add cluster as a bus member and ME is automaticallt created. There is only one active ME at any given time, the HA Mgr decides which server the ME runs on. In case of active ME failure, HA Mgr fails over the ME to another standby server.
The default topology consisting of just one ME in a bus is adequate for many applications. Advanctages in deploying more than one ME and linking them together are:

- Spreading messaging workload across multiple servers.
- Placing message processing close to the applications that are using it.
- Improving availability in the face of system or link failure
- Accommodating firewalls or other network restrictions that limit the ability of network hosts all to connect to a single ME.

A bus can connect to other buses, which are referred to as foreign buses. If ME's are on different buses, applications can use those different buses, each with its own topology and set of resources. To create a link to a foreign bus, the administrator first creates a virtual link from the local bus to the foreign bus, then creates a physical gateway link from a messaging engine in the local bus to the foreign bus.

Protocol attach is the only way WebSphere MQ applications can interact with Platform Messaging applications. Platform Messaging will not directly support any WebSphere MQ programming interface, such as MQI or the AMI. Connectivity between a ME and an MQ Queue Manager is established by defining an MQLink. One of the primary functions of the MQLink is to convert between the formats and protocols used by WebSphere MQ and Platform Messaging, this is handled by component called the JS/MQ protocol adapter.
When you have a bus, every destination on that bus must have a unique name, so, to a certain extent, the bus is a namespace. If you have a second bus, you could have another destination with the same name as one in the first bus.
When you go to the level of the JMS Proxies in the JNDI namespace, a JMS destination specifies only the destination name, not the bus name. The Connection Factory specifies the name of the bus to connect to. So two different connection factories specify different bus names.
WebSphere MQ and Platform Messaging are separate products and do not share any modules or configuration data.
Connectivity between ME and MQ Queue Manager is established by defining a WebSphere MQLink. WebSphere MQLink converts between the formats and protocols used by MQ and Platform Messaging.
An MQ Queue Manager cannot attach to the bus using any communications protocol other than TCP/IP.
A messaging engine cannot participate in an MQ cluster..

You can have multiple MQLinks out of a bus, but each link goes to a different queue manager, and further these queue managers should not be interconnected. The link engine can be part of a cluster, but the issue is in handling failover. The ME hosting the MQLink must keep a fixed host/port address because that is qhat MQ expects, and so you have to marry the new WAS HA support with more traditional HACMP like HA solutions.

The messaging support of the default messaging provider is only accessible from WAS web, ejb and client containers, and is interoperable with WebSphere MQ. If access is required to heterogeneous non-JMS applications, WebSphere MQ clustering, or other WebSphere MQ functions, you should install WebSphere MQ as a messaging provider.

A JMS client obtains connections to a service integration bus using a suitably configured JMS connection factory.
A JMS client will always perform the same steps in order to connect to a JMS provider:

1. Obtain a reference to a JMS connection factory from a JNDI name space.
2. Invoke the createConnection method on the JMS connection factory.

The important point to remember is that the JMS connection factory object will always execute within the same process as the JMS client. However, the JMS client, and therefore the JMS connection factory, might be execute inside of a WebSphere Application Server process, or they might be executing within a stand-alone JVM.
The connection factory is only able to determine the location of messaging engines that are defined within the same WebSphere cell. If the target bus is defined within another cell then a list of suitable provider endpoints must be configured on the connection factory.

Creating JMS resources using administrative clients.
To use a JMS Queue or Topic, you need 2 different layers of administratively created objects:
Bus Destination - Queue or TopicSpace - This is the destination on the bus and is invisible to the application
JMS Destination - JMS Queue or JMS Topic - This destination acts as a proxy or pointer to the destination on the bus.

Configuring JMS resources - Resources > JMS Providers > Default Messaging

Creating JMS Connection Factory
Used to obtain connections for both point-to-point and pub-sub messaging styles.
A unified Connection Factory is new in JMS 1.1
Enter Name, JNDI Name, Message Reliability, Bus Name, Remote Target Type, Target Significance
You can also create a Queue Connection Factory and a Topic Connection Factory separately.

Creating JMS Queue
Represents the Queue that applications interact with. This allows the application to interact with the Bus level queue by working with the JMS Queue Java object. Specify the Name, JNDI Name, Bus Name, Queue Name, Delivery Mode, Time to Live, and Priority.

Creating JMS Resources with wsadmin
Most of the new SIB administrative commands use the new $AdminTask wsadmin commands. The $scope parameter specifies the level at which the resource is visible.

EJB 2.1 MDB's - Message Driven Beans run inside WAS, and get messages from Queues or Topics by way of Activation Specifications.
EJB 2.0 uses Listener Ports to connect an MDB to a destination. Listener ports are also used if WAS MQ is the JMS Provider.
The Bus uses JCA connectors to connect the MDB to the destination. Create an Activation Specification object and bind it into the Namespace. When the MDB is started the Activation Specification connects it to a destination.

MDB Administration with wsadmin
The Activation Specification contains the JNDI name of a JMS Destination to which an MDB will listen.
Creating Activation Specification using wsadmin:

- set params [list -name $activationSpecName -jndiName $activationSpecJNDIName -busName $BusName -destJNDIName $destinationJNDIName -destinationType $destinationType]
- $AdminTask createSIBJMSActivationSpec $scope $params

When installing the application, bind the MDB to the JNDI name of the Activation Specification.

Support for EJB 2.0 MDB's
Existing EJB 2.0 MDB's can be deployed against a Listener Port - Same as WAS v5
EJB 2.0 MDB's can also use Activation Specifications - Application code needs to be modified.
EJB 2.1 MDB's should use Activation Specifications

Example SIB Configuration
Setting up a Service Integration Bus – Setting up a SIB is the first step in creating the infrastructure required to participate in messaging. A SIB is made up of a group of interconnected servers and clusters that have been added as members of the bus. Applications connect to a SIB at one of the messaging engines associated with its bus members.

Service Integration -> Buses -> New -> msgBus

1. Service Integration -> Buses -> msgBus -> Additional Properties -> Bus Members -> Add
   Select from the cluster drop-down menu and also add the jndi name
   Creating a Bus member automatically creates a messaging engine.
2. Create a DB2 JDBC provider that will be used by the data source for the messaging engine
   Resources -> JDBC Providers -> New -> DB2 Legacy CLI-based Type 2 JDBC Driver -> XA data source
3. Add a datasource for the messaging engine
   DB2 JDBC Provider -> Data sources -> New
   Name = Messaging Engine Datasource
   JNDI = jdbc/newsMsgEng
   Enter MESENG for the database name
4. Create a SIB destination name LstnrQueue
   Service Integration -> Buses -> msgBus -> Destinations -> New -> Queue enter LstnrQueue -> Next
   Assign the queue to the Cluster bus member -> Next -> Finish

Setting up a JMS destination – The SIB and bus destination are now set up, these are required before a JMS destination can be defined for the MDB to use. Setting up the JMS resources requires two setup steps to create the:
JMS connection factory
JMS queue

1. Create a new JMS connection factory called LstnrJMSCF and associate it with the SIB created earlier, msgBus
2. Resources -> JMS Providers -> Default Messaging -> Cluster scope -> Connection factories -> JMS connection factory -> new -> LstnrJMSCF and for jndi: jms/LstnrJMSCF
   Select msgBus for the bus name
3. Create a JMS Queue named LstnrJMSQueue – This is the association between the JMS queue and the SIB destination
4. Resources -> JMS Providers -> Default Messaging -> Destinations -> JMS queue -> New -> LstnrJMSQueue and for jndi: jms/LstnrJMSQueue
   Select msgBus for the Bus name
   Select LstnrQueue for the Queue name

Setting up the Activation Specification – The activation specification is what a MDB uses to establish communication with the JMS destination. Activation Specs enable EJB 2.1 MDB's to use JCA 1.5 resource adapter to receive JMS (and non-JMS) messages.

1. Create the LstnrAS activation spec for the MDB used
2. Resources -> JMS Providers -> Default Messaging -> Activation Specifications -> JMS activation specification -> New -> App Listener Activation Spec for the name
3. Enter eis/LstnrAS for the jndi name
4. Destination type of Queue
5. Enter jms/LstnrJMSQueue for the destination JNDI name.
6. Select msgBus for the bus name

Section 9 – Miscellaneous Information

WAS v6 default port definitions

Port Name	WAS base	WAS ND	File
HTTP_Transport	9080	N/A	Serverindex.xml virtualhosts.xml
HTTPS_Transport	9443	N/A	Serverindex.xml virtualhosts.xml
HTTP admin console port	9060	9060	Serverindex.xml virtualhosts.xml
HTTPS admin console port	9043	9043	Serverindex.xml virtualhosts.xml
Internal JMS Server	5557	N/A	Server.xml
JMSSERVER_Queued_Address	5558	N/A	Serverindex.xml
JMSSERVER_Direct_Address	5559	N/A	Serverindex.xml
BOOTSTRAP_Address	2809	9809	Serverindex.xml
SOAP_Connector_Address	8880	8879	Serverindex.xml
DRS_Client_Address (Decprecated)	7873	7989	Serverindex.xml
SAS_SSL_ServerAuth_Listener_Address	9401	9401	Serverindex.xml
CSIV2_SSL_ServerAuth_Listener_Address	9403	9403	Serverindex.xml
CSIV2_SSL_MutualAuth_Listener_Address	9402	9402	Serverindex.xml

IBM HTTP Server Port	80	N/A	Virtualhosts.xml plugin-cfg.xml
IBM HTTPS Server admin port	8008	N/A	Plugin-cfg.xml
CELL_Discovery_Address	N/A	7277	Serverindex.xml
ORB_Listener_Address	9100	9100	Serverindex.xml
DCS_UNICAST_ADDRESS	9353	9352	Serverindex.xml
SIB_ENDPOINT_ADDRESS	7276	7276	Serverindex.xml
SIB_ENDPOINT_SECURE_ADDRESS	7286	7286	Serverindex.xml
SIB_MQ_ENDPOINT_ADDRESS	5558	5558	Serverindex.xml
SIB_MQ_ENDPOINT_SECURE_ADDRESS	5578	5578	Serverindex.xml
CELL_DISCOVERY_ADDRESS	N/A	7277	Serverindex.xml
CELL_Multicast_Discovery_Address	N/A	7272	Serverindex.xml
Node_Multicast_IPV6_Discovery_Address	5001	5001	Serverindex.xml

WAS v6 Nodeagent default port definitions

Port name	Value	File
BOOTSTRAP_Address	2809	Serverindex.xml
ORB_Listener_Address	9900	Serverindex.xml
SAS_SSL_ServerAuth_Listener_Address	9901	Serverindex.xml
CSIV2_SSL_MutualAuth_Listener_Address	9202	Serverindex.xml
CSIV2_SSL_ServerAuth_Listener_Address	9201	Serverindex.xml
NODE_Discovery_Address	7272	Serverindex.xml
NODE_Multicast_Discovery_Address	5000	Serverindex.xml
Node_IPV6_Multicast_Discovery_Address	5001	Serverindex.xml
DRS_UNICAST_ADDRESS	9354	Serverindex.xml
DRS_Client_Address	7888	Serverindex.xml
SOAP_Connector_Address	8878	Serverindex.xml

The SOAP_CONNECTOR_ADDRESS port is used by SOAP clients such as the wsadmin command to connect to the server to perform administration tasks.

There are four ways of authenticating users using servlets and JSP:

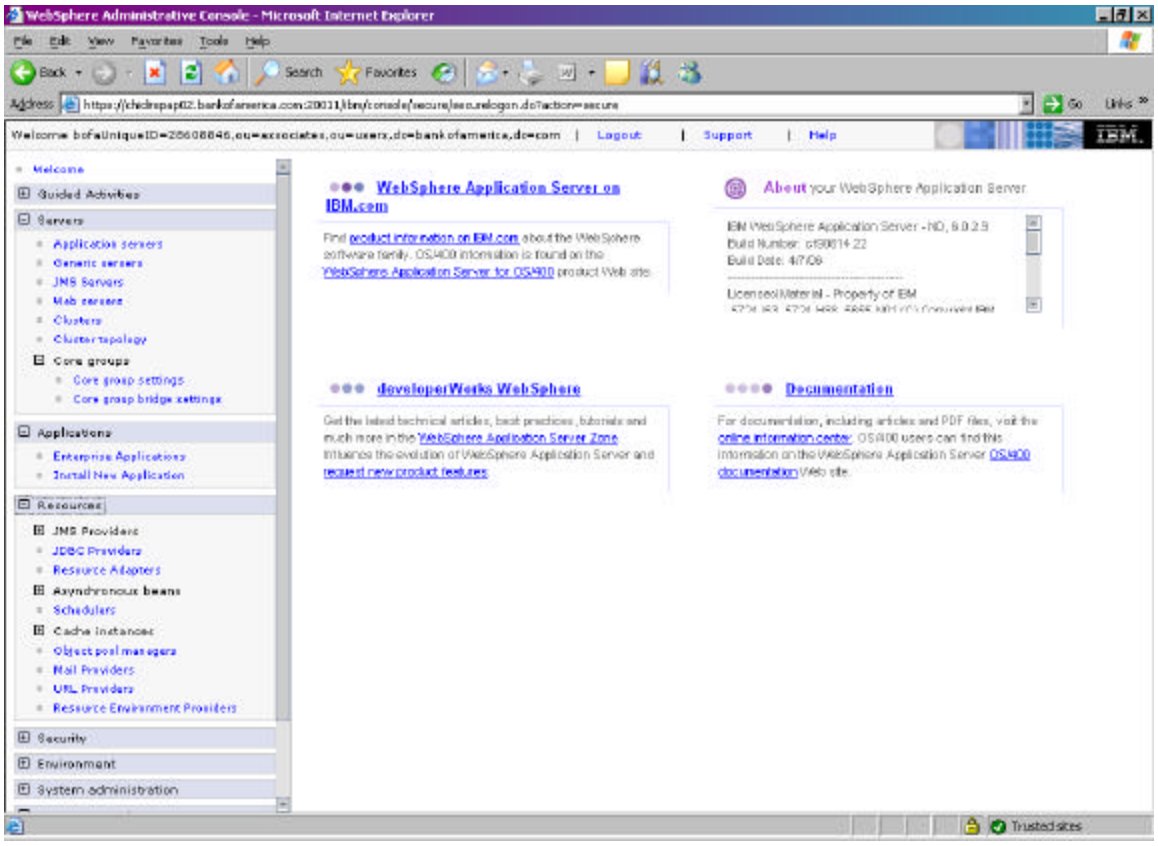
- Basic Authentication – When a user attempts to access a JSP page, the system will request a user name and password (using a built-in interface that depends on the Web Browser) and will allow several attempts to supply correct credentials before providing an error page.
- Form-Based authentication – Similar to basic authentication except that you can create your own login interface by using HTML. This is often more useful than basic authentication because you can create login interfaces that require something other than, or in addition to, the traditional user name and password.
- Digest Authentication – Similar to basic authentication except that passwords are encrypted using a hash formula. This makes digest authentication more secure than basic authentication, as only the password's hash value are transmitted over HTTP. However, digest authentication is still rarely used, simply because there are better and more complex ways of providing security (such as HTTPS and SSL).
- Client Certificate Authentication – This requires that each client accessing the resource has a certificate that it sends to authenticate itself. This type of authentication requires SSL.

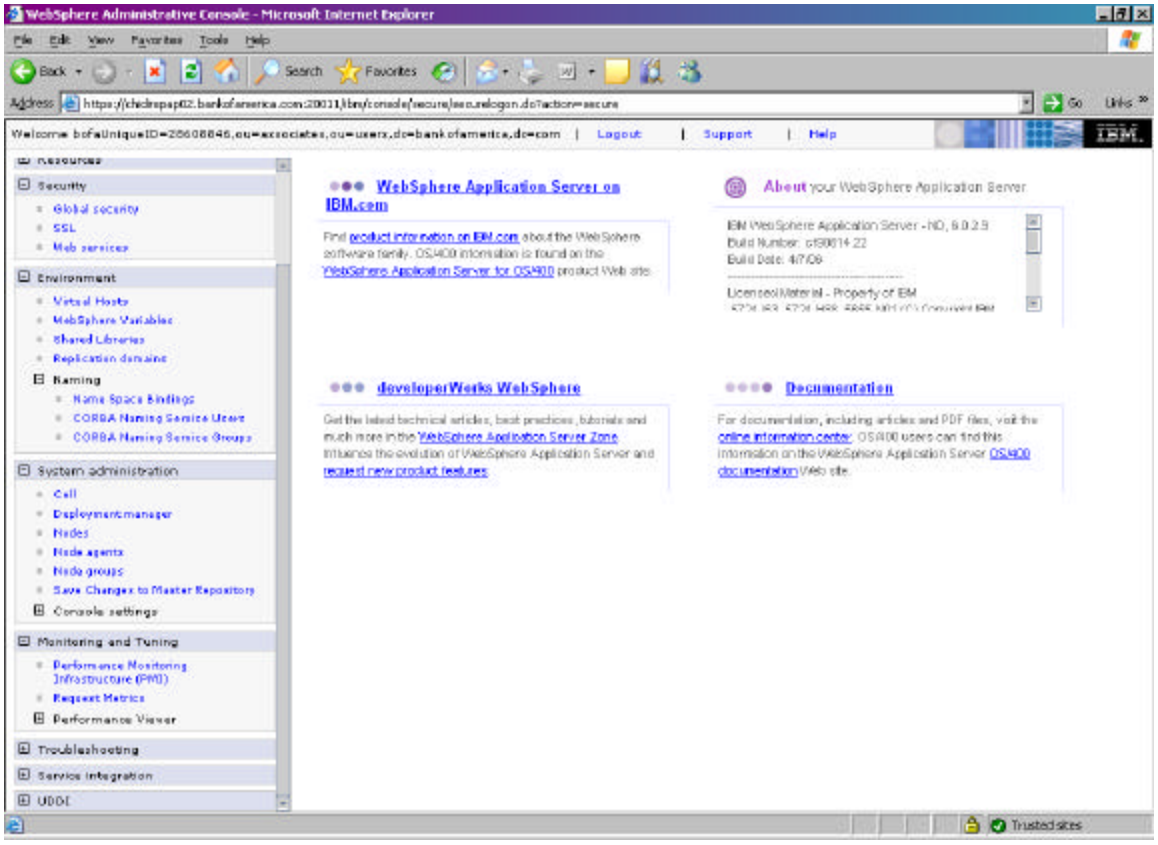
J2EE Platform Roles:

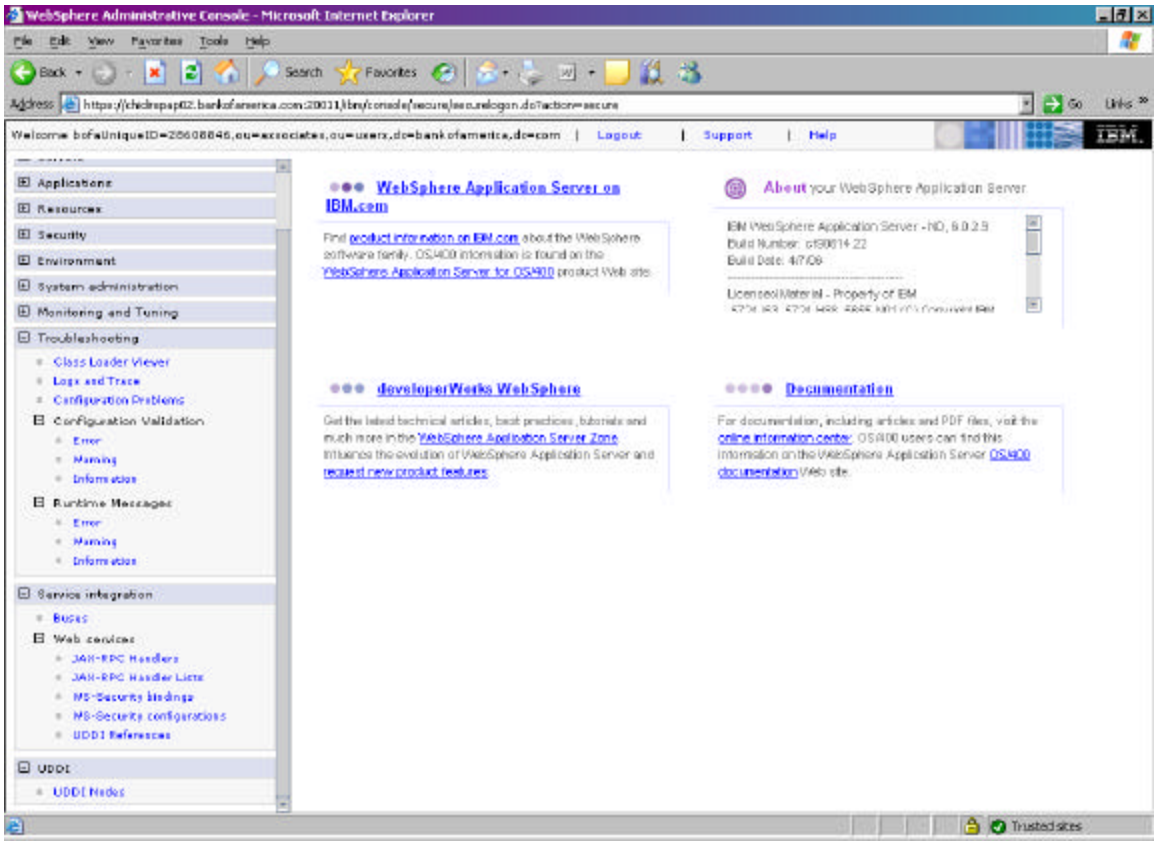
- Application Component Provider – Produces application building blocks(HTML, EJBs, JSPs, etc)
- Application Assembler – Takes components developed by component providers and assembles them into a complete J2EE application.
- Deployer – Deploys, configures and runs EJBs and web applications.
- System Administrator – Configures and administers the infrastructure.

The ikeyman utility is used to create key databases, public and private key pairs, and certificate requests.

When installing an enterprise application, the starting weight configuration specifies the order in which application are started. The application with the lowest starting weight is started first.


WebSphere Administrative Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address https://chclnspapt02.bankofamerica.com:20011/ibm/console/secure/securelogon.do?action=secure
Go Links
Welcome bofaUniqueID=28608846,ou=associates,ou=users,dc=bankofamerica,dc=com | Logout | Support | Help
IBM
Welcome
Guided Activities
Servers
Application servers
Generic servers
JMS Servers
Web servers
Clusters
Cluster topology
Core groups
Core group settings
Core group bridge settings
Applications
Enterprise Applications
Install New Application
Resources
JMS Providers
JDBC Providers
Resource Adapters
Asynchronous beans
Schedulers
Cache instances
Object pool managers
Mail Providers
URL Providers
Resource Environment Providers
Security
Environment
System administration
WebSphere Application Server on IBM.com
Find product information on IBM.com about the WebSphere software family. OS/400 information is found on the WebSphere Application Server for OS/400 product Web site.
About your WebSphere Application Server
IBM WebSphere Application Server - ND, 6.0.2.9
Build Number: cf30814.22
Build Date: 4/7/06
Licensed Material - Property of IBM
developerWorks WebSphere
Get the latest technical articles, best practices, tutorials and much more in the WebSphere Application Server Zone. Influence the evolution of WebSphere Application Server and request new product features.
Documentation
For documentation, including articles and PDF files, visit the online information center. OS/400 users can find this information on the WebSphere Application Server OS/400 documentation Web site.
Trusted sites


WebSphere Administrative Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address https://chclnspapt02.bankofamerica.com:20011/ibm/console/secure/securelogon.do?action=secure
Go Links
Welcome bofaUniqueID=28608846,ou=associates,ou=users,dc=bankofamerica,dc=com | Logout | Support | Help
IBM
Resources
Security
Global security
SSL
Web services
Environment
Virtual Hosts
WebSphere Variables
Shared Libraries
Replication domains
Naming
Name Space Bindings
CORBA Naming Service Users
CORBA Naming Service Groups
System administration
Cell
Deployment manager
Nodes
Node agents
Node groups
Save Changes to Master Repository
Console settings
Monitoring and Tuning
Performance Monitoring Infrastructure (PMI)
Request Metrics
Performance Viewer
Troubleshooting
Service integration
UDDI
WebSphere Application Server on IBM.com
Find product information on IBM.com about the WebSphere software family. OS/400 information is found on the WebSphere Application Server for OS/400 product Web site.
About your WebSphere Application Server
IBM WebSphere Application Server - ND, 6.0.2.9
Build Number: cf30814.22
Build Date: 4/7/06
Licensed Material - Property of IBM
developerWorks WebSphere
Get the latest technical articles, best practices, tutorials and much more in the WebSphere Application Server Zone. Influence the evolution of WebSphere Application Server and request new product features.
Documentation
For documentation, including articles and PDF files, visit the online information center. OS/400 users can find this information on the WebSphere Application Server OS/400 documentation Web site.
Trusted sites



Resources
The information contained within this document is not the work of the author of this document.
The information contained within is a collection from the following IBM sources:

IBM Infocenter - http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp
IBM Education Assistant - http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/index.jsp
IBM Training Class - Administration of WebSphere Application Server V6

IBM Redbooks
WebSphere Application Server V6 System Management & Configuration Handbook
ISBN - 0738492019
IBM Form Number - SG24-6451-00

IBM WebSphere Application Server V6.1 Security Handbook
ISBN - 0738496707
IBM Form Number - SG24-6316-01

WebSphere Application Server V6 Scalability and Performance Handbook
ISBN - 0738490601
IBM Form Number - SG24-6392-00