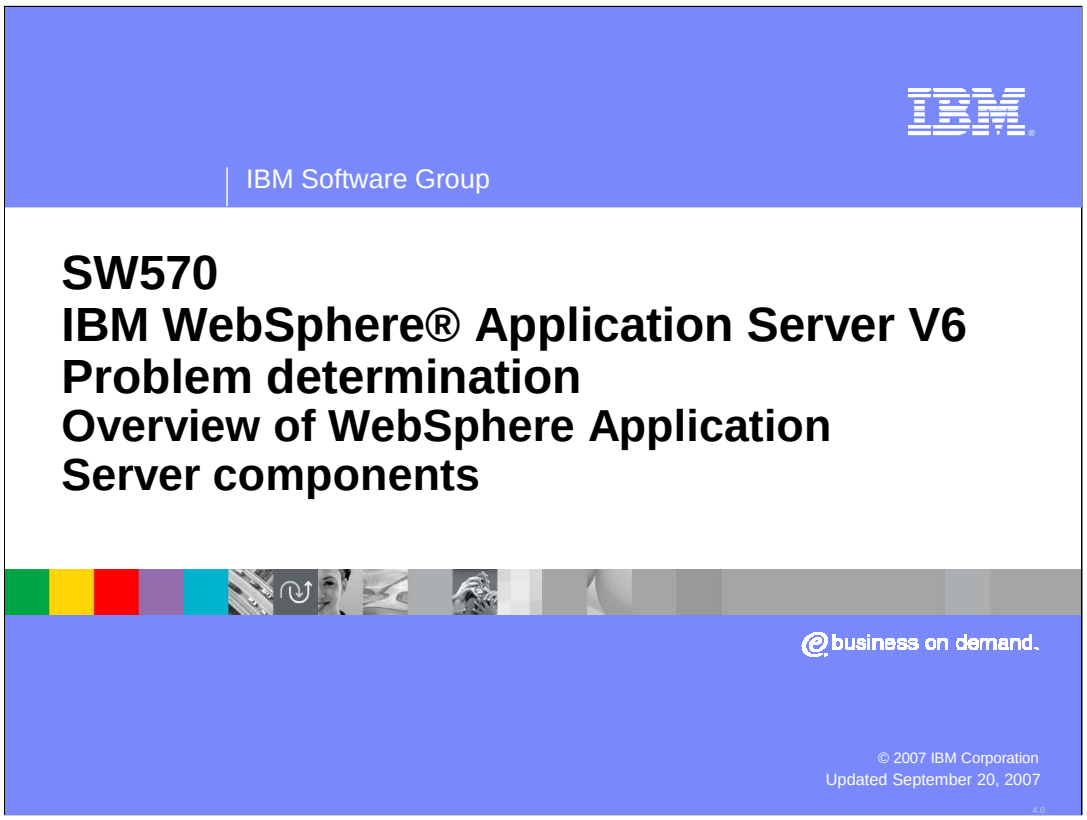
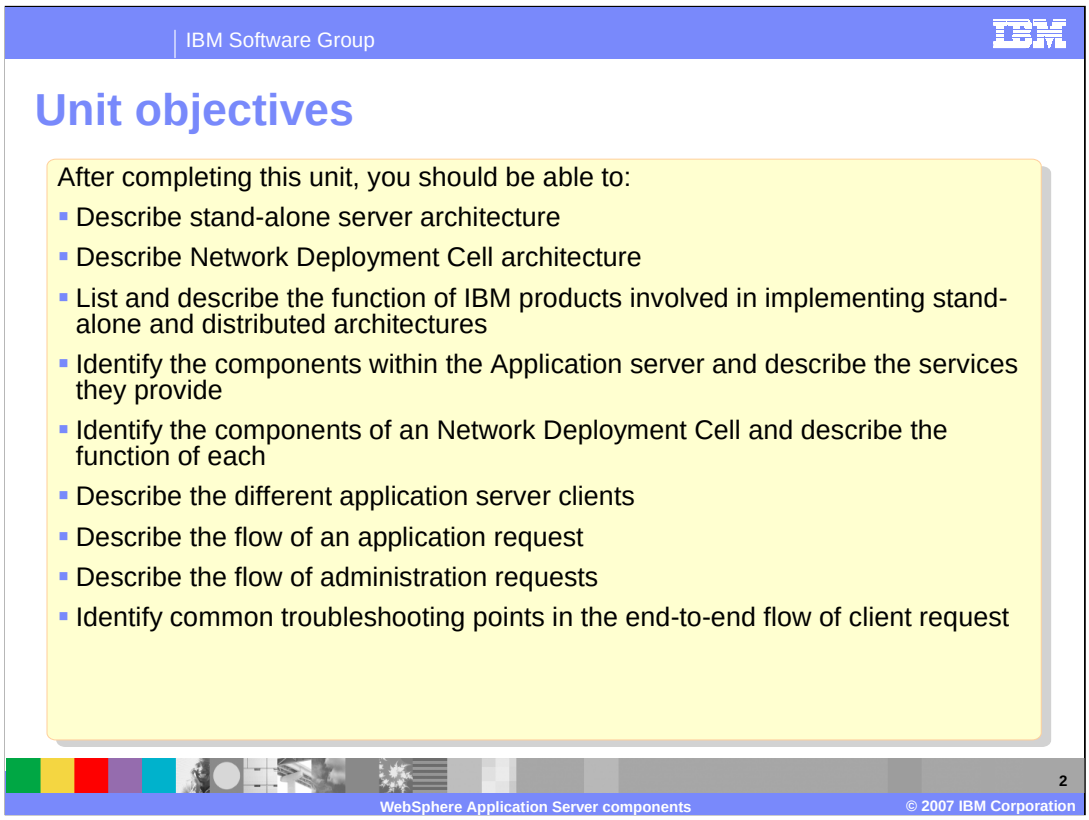


This module provides an overview of the topology of a system and identifies and describes key components as well as troubleshooting points.



After completing this unit, you will be able to:

•Describe Network Deployment Cell architecture

•List and describe the function of IBM products involved in implementing stand-alone and distributed architectures

•Identify the components within the Application server and describe the services they provide

•Identify the components of an ND Cell and describe the function of each

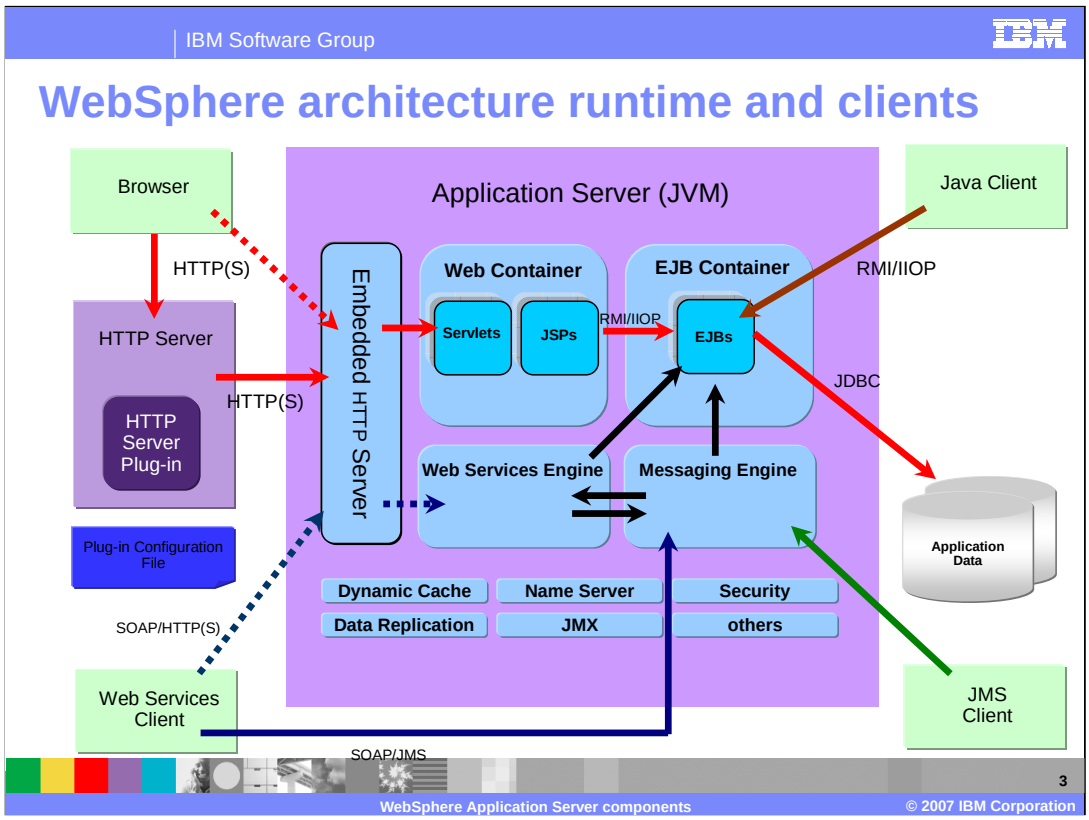
•Describe stand-alone server architecture

•Describe the different application server clients

•Describe the flow of an application request

•Describe the flow of administration requests

•Identify common troubleshooting points in the end-to-end flow of client request



This diagram illustrates the basic architecture of WebSphere Application Server, including several of the larger components.

The main element is the application server, a Java process that encapsulates many services, including the containers, where business logic executes. If you are familiar with J2EE, you will recognize the Web Container and the EJB container. The Web Container executes Servlets and JavaServer Pages (JSPs), 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 JSPs can act independently, they most commonly make calls to Enterprise Java Beans (EJBs) to execute business logic or access data. EJBs, 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 Servlet/JSP.

The JMS messaging engine is built into the application server. This is a pure-Java messaging engine. JMS destinations, known as queues and topics provide asynchronous messaging services to the code running inside the containers. JMS will be covered in more depth later in this course.

The Web Services engine enables application components to be exposed as web services, which can be accessed using Simple Object Access Protocol (SOAP).

Several other services run within the application server, including the Dynamic Cache, Data Replication, Security, and others. These will be covered later in the course.

There are also some important components outside of the application server process.

WebSphere Application Server also provides a plug-in for HTTP servers that determines what HTTP traffic is intended to be handled by WebSphere, and routes the requests to the appropriate server. The plug-in is also a critical player in workload management of HTTP requests, as it can distribute the load to multiple application servers, as well as steer traffic away from unavailable servers. It too reads its configuration from a special XML file.

Web container

- Each application server runtime has one logical Web container, which can be modified, but not created or removed
- Each Web container provides the following:
  - Web container transport chains
  - Servlet processing
  - JSP processing
  - Session management
  - HTML and other static content processing
  - Web services engine
  - Dynamic caching
  - Threading support (thread pool)
- The Web container processes
  - Servlets
  - JSP files

Each application server has one logical Web container that processes the servlets and jsp files. The Web container has several components that can be configured through the administration console.

Requests are directed to the Web container using the Web container inbound

transport chain. The chain consists of a TCP inbound channel that provides the connection to the network, an HTTP inbound channel that serves HTTP 1.0 and 1.1 requests, and a Web container channel over which requests for

servlets and JSPs are sent to the Web container for processing.

When handling servlets, the Web container creates a request object and a response object, then invokes the servlet service method. The Web container invokes the servlet's destroy method when appropriate and unloads the servlet, after which the JVM performs garbage collection.

Requests for HTML and other static content that are directed to the Web container are served by the Web container inbound chain. However, in most cases, using an external Web server and Web server plug-in as a front-end to a Web container is more appropriate for a production environment.

Web services are provided as a set of APIs in cooperation with the J2EE

Application specification. Web services engines are provided to support Simple Object Access Protocol (SOAP).

Enterprise JavaBeans (EJB) container

- The Enterprise JavaBeans (EJB) container provides all the runtime services that are needed to deploy and manage enterprise beans.
- It is a server process that handles requests for session beans, entity beans, and message-driven beans (MDBs)
- The EJB container provides an interface between the enterprise beans and the server.
- Together, the container and the server provide the enterprise bean runtime environment.
- The container provides many low-level services including
  - Threading support (thread pool)
  - Transaction support
- The container manages data storage and retrieval for the contained EJBs
- A single container can host more than one EJB Java archive (JAR) file.

The EJB container is responsible for all the runtime services necessary to deploy and manage all of the enterprise beans. The EJB container provides an interface so that the deployed EJBs can communicate with the server because EJBs cannot directly communicate beyond the EJB container. The container also provides thread pooling and transaction support for the EJBs. Together, the EJB container and the server provide the EJB runtime environment.

WebSphere Application Server services

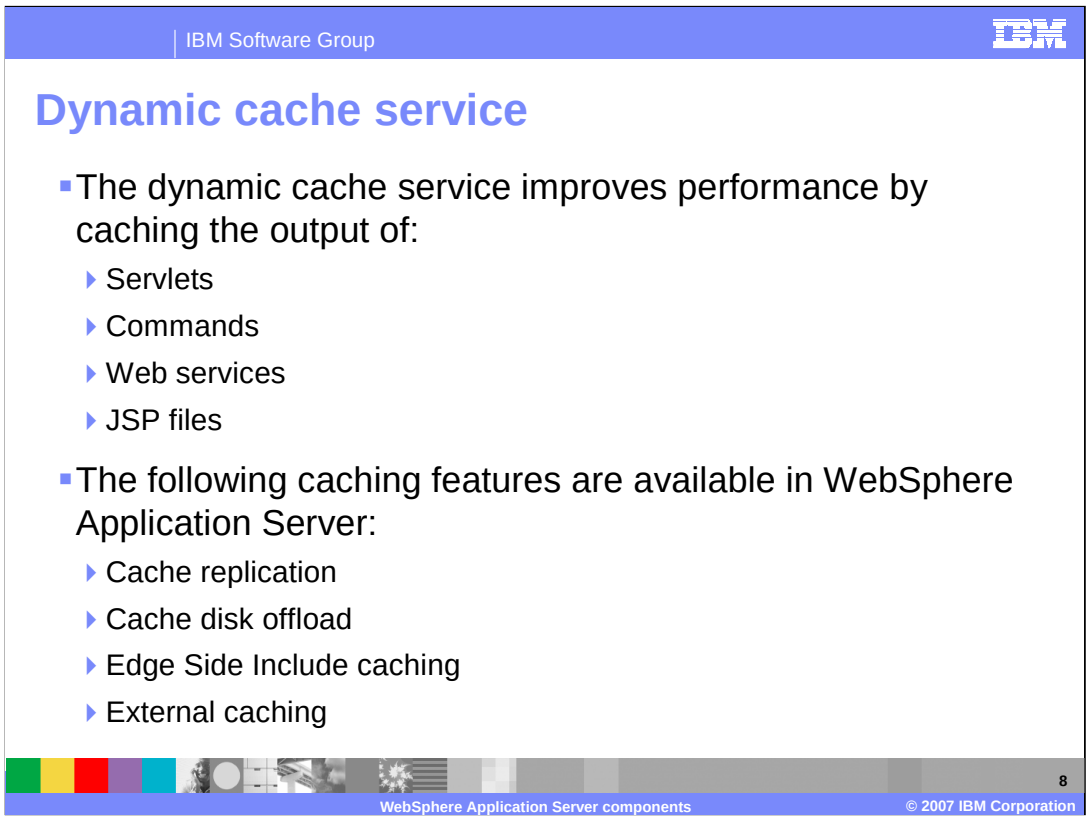
- J2EE Connector Architecture service (JCA)
- Transaction service
- Dynamic cache service (Dynacache)
- Data Replication Service (DRS)
- Message listener service
- Object Request Broker (ORB) service
- Administrative service (JMX)
- Diagnostic trace and Debugging service
- Name service (JNDI)
- Performance Monitoring Interface service
- Security service (JAAS and J2 security)
- Service Integration Bus (SIBus) service

WebSphere Application Server is responsible for hosting and managing a large number of both external and internal services. This is a sampling of some of the services that WebSphere provides. We will go into more detail on some of the most common services later in this presentation.

Transaction service

- WebSphere applications use transactions to coordinate multiple updates to resources as one unit of work.
- Transactions are started and ended by applications or the container. Hence transactions can be configured as either:
  - Container-managed
  - Bean-managed
- WebSphere Application Server is a transaction manager that supports the coordination of resource managers through the XAResource interface and participates in distributed global transactions.
- You can also configure WebSphere applications to interact with:
  - Databases
  - Java Message Service (JMS) queues
  - JCA connectors

Transactions are a way for WebSphere Application Server to bundle multiple updates into one unit of work. They type of transactions an application uses depends on the tye of application. A session bean can either use container-managed transactions where the bean delegates management of transactions to the container, or bean-managed transactions where the bean manages transactions itself. Entity beans use container-managed transactions while web components or servlets use bean-managed transactions.



The dynamic cache works within an application server by intercepting calls to objects that can be cached. For example, through a servlet's service() method or a command's execute() method.

The dynamic cache either stores the object's output to or serves the object's content from the within 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 significant gains in server response time, throughput, and scalability.

Cache replication takes place using the WebSphere data replication service. Data is generated one time and then copied or replicated to other servers in the cluster, saving execution time and resources.

By default, when the number of cache entries reaches the configured limit for a given WebSphere server, eviction of cache entries occurs, allowing new entries to enter the cache service. The dynamic cache includes a disk offload feature that copies the evicted cache entries to disk for potential future access.

The Web server plug-in contains a built-in Edge Side Include (ESI) processor.

The ESI processor caches whole pages, as well as fragments, providing a higher cache hit ratio. The cache implemented by the ESI processor is an in-memory cache, not a disk cache. Therefore, the cache entries are not saved when the Web server is restarted.

The dynamic cache controls caches outside of the application server, such as that provided by the Edge components, an IBM HTTP Server's FRCA cache, and a WebSphere HTTP Server plug-in ESI Fragment Processor. When external cache groups are defined, the dynamic cache matches external cache entries with those groups and pushes out cache entries and invalidations to those groups. This external caching allows WebSphere to manage dynamic content beyond the application server. The content can then be served from the external cache, instead of the application server, improving performance.

Object Request Broker service

- An Object Request Broker (ORB) manages the interaction between EJB clients and EJBs, using Internet Inter-ORB Protocol (IIOP).
- The ORB service enables clients to make requests and receive responses from EJBs in a network-distributed environment.
- The ORB service provides a framework for clients to locate objects in the network and call operations on those objects.
- The client-side ORB is responsible for creating an IIOP request that contains the operation and any required parameters, and for sending the request across the network.
- The server-side ORB receives the IIOP request, locates the target object, invokes the requested operation, and returns the results to the client.

An Object Request Broker (ORB) manages the interaction between clients and servers, using Internet Inter-ORB Protocol (IIOP). The ORB service enables clients to make requests and receive responses from servers in a network-distributed environment. The ORB service provides a framework for clients to locate objects in the network and call operations on those objects as though the remote objects were located in the same running process as the client. The ORB service provides location transparency. The client calls an operation on a local object, known as a *stub*. Then the stub forwards the request to the desired remote object, where the operation is run, and the results are returned to the client. The client-side ORB is responsible for creating an IIOP request that contains the operation and any required parameters, and for sending the request in the network. The server-side ORB receives the IIOP request, locates the target object, invokes the requested operation, and returns the results to the client. The client-side ORB demarshals the returned results and passes the result to the stub, which returns the result to the client application, as though the operation had been run locally.

WebSphere Application Server uses an ORB to manage communication between client applications and server applications as well as communication among product components.

Administrative service

- The administrative service runs within each server JVM. In Base and Express, the administrative service runs in the application server.
- In Network Deployment, each of the following hosts an administrative service:
  - Deployment manager
  - Node agents
  - Application servers
- Provides the necessary functions to manipulate configuration data for the server and its components.
- The configuration is stored in a repository in the server's file system.
  - The repository consists of sub-directories which contain XML files of configuration information
- The administrative service is invoked via JMX calls that go through Web Services (SOAP/HTTP or RMI/IIOP)

The administrative service runs within each server JVM. In Base and Express, the administrative service runs in the application server. In Network Deployment, the Deployment manager, Node agent, and Application server each host an administrative service. The administrative service provides the necessary functions to manipulate configuration data for the server and its components. The configuration is stored in a repository in the server's file system. The administrative service has a security control and filtering functionality that provides different levels of administration to certain users or groups using the following administrative roles: Administrator, Monitor, Configurator, and Operator.

Name service

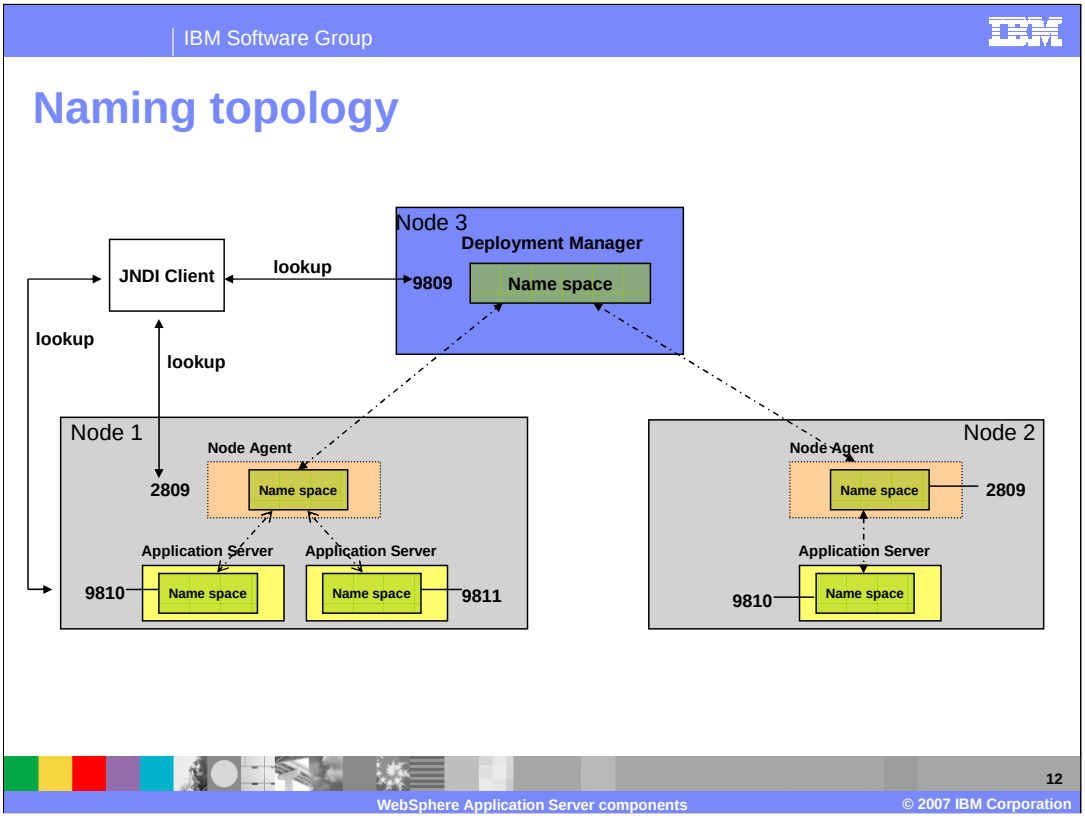
- Each application server hosts a name service that provides a Java Naming and Directory Interface (JNDI) name space.
- Registers all EJB and J2EE resources that are hosted by the application server including:
  - JDBC providers
  - JMS destinations
  - JCA (J2C) components
  - URL providers
  - JavaMail providers
- The deployment manager and all node agents host a name service.
- Configured bindings can map resources to remote locations.

The Name Service is used to register resources hosted by the application server. The JNDI implementation in WebSphere Application Server is built on top of a Common Object Request Broker Architecture (CORBA) naming service (CosNaming).

JNDI provides the client-side access to naming and presents the programming model that application developers use. CosNaming provides the server-side implementation and is where the name space is stored. JNDI essentially provides a client-side wrapper of the name space stored in CosNaming and interacts with the CosNaming server on behalf of the client.

The naming architecture is used by clients of WebSphere applications to obtain references to objects related to those applications. These objects are bound into a mostly hierarchical structure, referred to as a *name space*. The name space structure consists of a set of name bindings, each containing a name relative to a

specific context and the object bound with that name. The name space can be accessed and manipulated through a name server.



For additional scalability, the name space for a cell is distributed among the various servers. The deployment manager, node agent, and application server processes each host a name server.

The default initial context for a server is its server root. System artifacts, such as EJB homes and resources, are bound to the server root of the server with which they are associated.

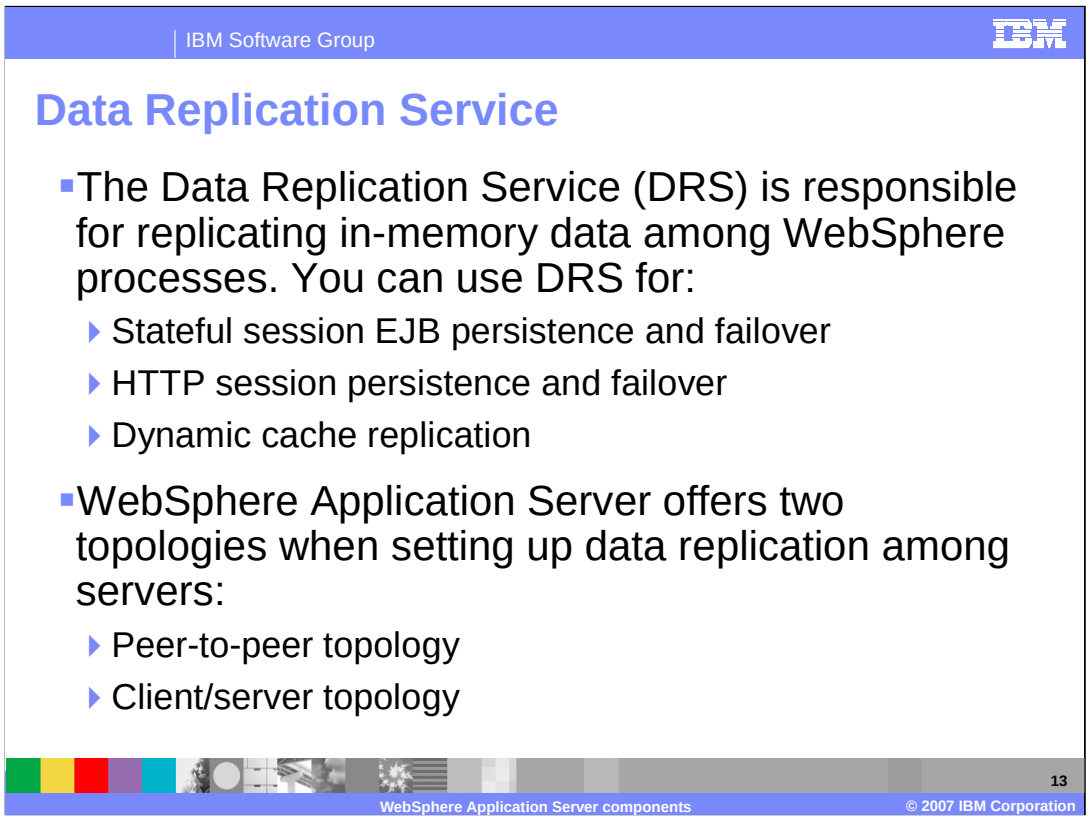
The name space is partitioned into *transient* areas and *persistent* areas. Server roots are transient. System-bound artifacts such as EJB homes and resources are bound under server roots. There is a cell-persistent root that is used for cell-scoped persistent bindings and a node-persistent root that is used to bind objects with a node scope.

A *name space* is a collection of all names bound to a particular name server. A name space can contain naming context bindings to contexts located in other servers. If this is the case, the name space is said to be a *federated name space*, because it is a collection of name spaces from multiple servers. The name spaces link together to cooperatively form a single logical name space. In a federated name space, the real location of each context is transparent to client applications. Clients have no knowledge that multiple name servers are handling resolution requests for a particular requested object.

In a Network Deployment distributed server configuration, the name space for the cell is federated among the deployment manager, node agents, and application servers of the cell. Each such server hosts a name server. All name servers provide the same logical view of the cell name space, with the various server roots and persistent partitions of the name space being interconnected by means of the single logical name space.

You can use the configuration graphical interface and script interfaces to configure bindings in various root contexts within the name space. These bindings are read-only and are bound by the system at server startup.

WebSphere Application Server also contains support for CORBA object URLs (corbaloc and corbaname) as JNDI provider URLs and lookup names.



Replication domains, consisting of server or cluster members that have a need to share internal data, use the Data Replication Service to share their data. Multiple domains can be used, each for a specific task among a set of servers or clusters. While HTTP session replication and EJB state replication can (and should) share a domain, you need a separate domain for dynamic cache replication. You can define a domain so that each domain member has a single replicator that sends data to another domain member. You can also define a domain so that each member has multiple replicators that send data to multiple domain members.

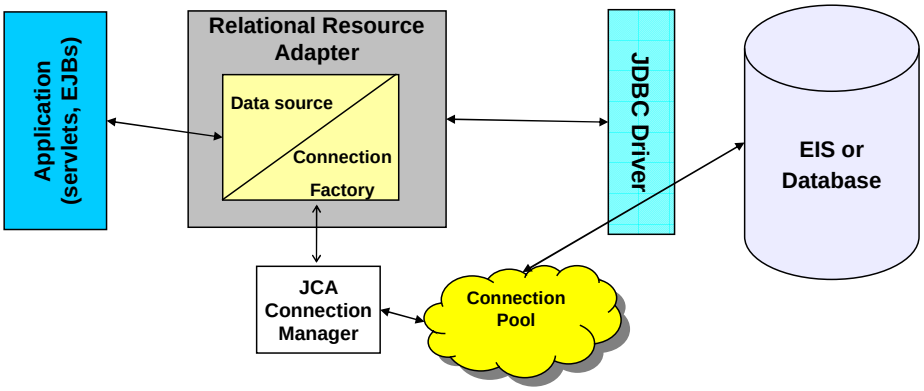
WebSphere Application Server offers two topologies when setting up data replication among servers, Peer to peer and Client server.

In Peer-to-peer, each application server stores sessions in its own memory and retrieves sessions from other application servers. In other words, each application server acts as a *client* by retrieving sessions from other application servers. Each application server also acts as a *server* by providing sessions to other application servers. This mode, working in conjunction with the workload manager, provides hot failover capabilities.

In Client Server, client application servers send session information to the replication servers and retrieve sessions from the servers. They respond to user requests and store only the sessions of the users with whom they interact. Application servers act as either a replication client or a server. Those that act as replication servers store sessions in their own memory and provide session information to clients. They are dedicated replication servers that store sessions but do not respond to user requests.

J2EE Connector Architecture service (JCA)

- The JCA Connection Manager administers
  - Connections obtained through resource adapters defined by the JCA
  - Data sources defined by the JDBC 2.0 Extensions



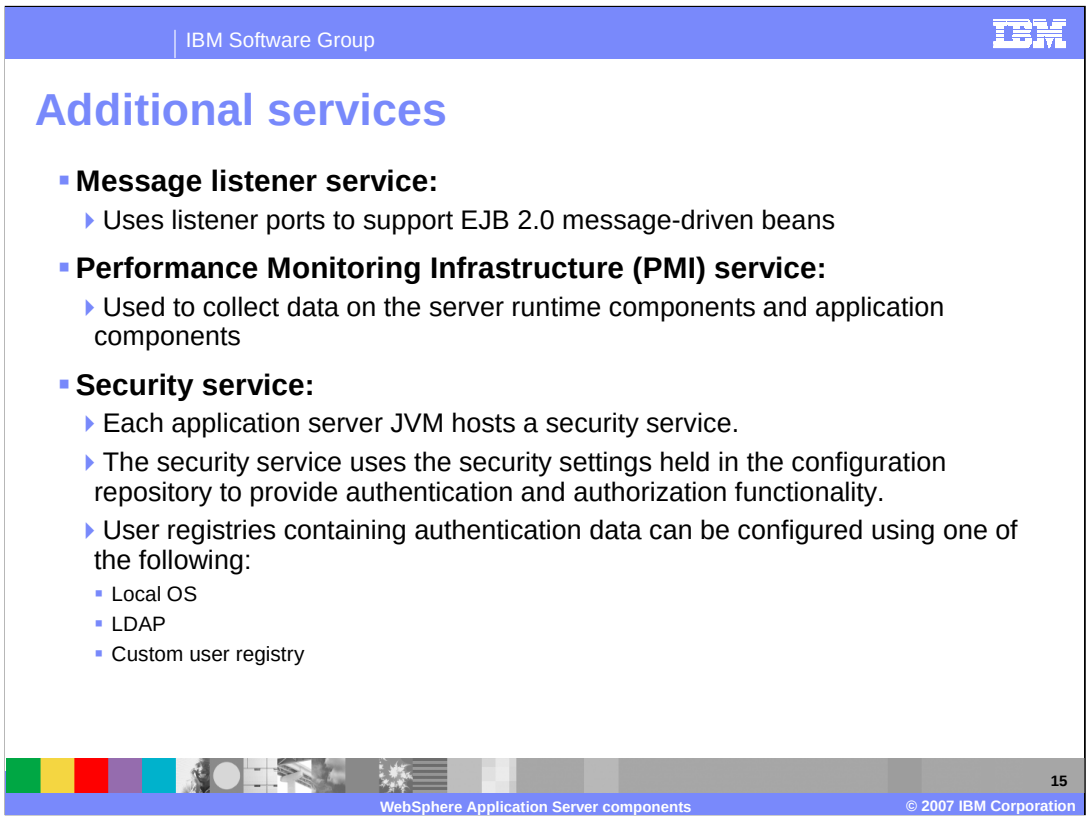
Connection management for access to enterprise information systems (EIS) in WebSphere Application Server is based on the J2EE Connector Architecture (JCA) specification, also sometimes referred to as J2C. The connection between the enterprise application and the EIS is done through the use of EIS-provided resource adapters, which are plugged into the application server. The architecture specifies the connection management, transaction management, and security contracts that exist between the application server and the EIS.

Within the application server, the Connection Manager pools and manages connections. The Connection Manager administers connections that are obtained through both resource adapters defined by the JCA specification and data sources defined by the JDBC 2.0 Extensions.

The JCA Connection Manager provides the connection pooling, local transaction, and security supports. The relational resource adapter provides the JDBC wrappers and JCA CCI implementation that allow applications using bean-managed persistence, JDBC calls, and container-managed persistence beans to access the database JDBC Driver.

The JCA resource adapter is a system-level software driver supplied by EIS vendors or other third-party vendors. It provides the connectivity between J2EE components (an application server or an application client) and an EIS.

One resource adapter, the WebSphere Relational Resource Adapter, is predefined for handling data access to relational databases. This resource adapter provides data access through JDBC calls to access databases dynamically. It provides connection pooling, local transaction, and security support. The WebSphere persistence manager uses this adapter to access data for container-managed persistence beans.

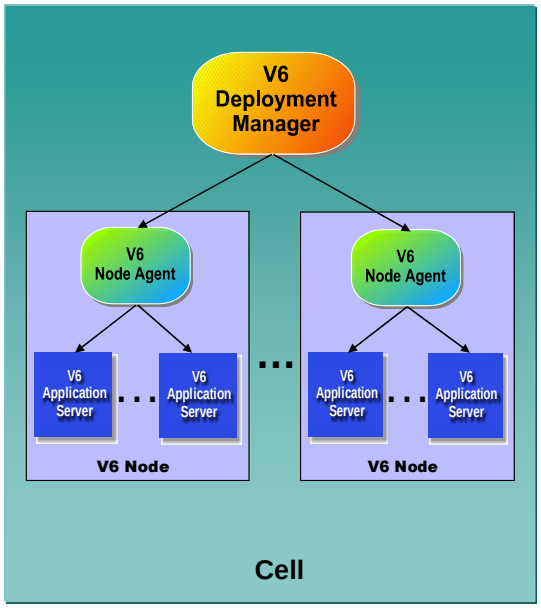


EJB 2.1 uses an ActivitionSpec to connect message-driven beans to destinations. However, you can deploy existing EJB 2.0 message-driven beans against a listener port as in WebSphere Application Server V5. For those message-driven beans, the message listener service provides a listener manager that controls and monitors one or more JMS listeners. Each listener monitors a JMS destination on behalf of a deployed message-driven bean.

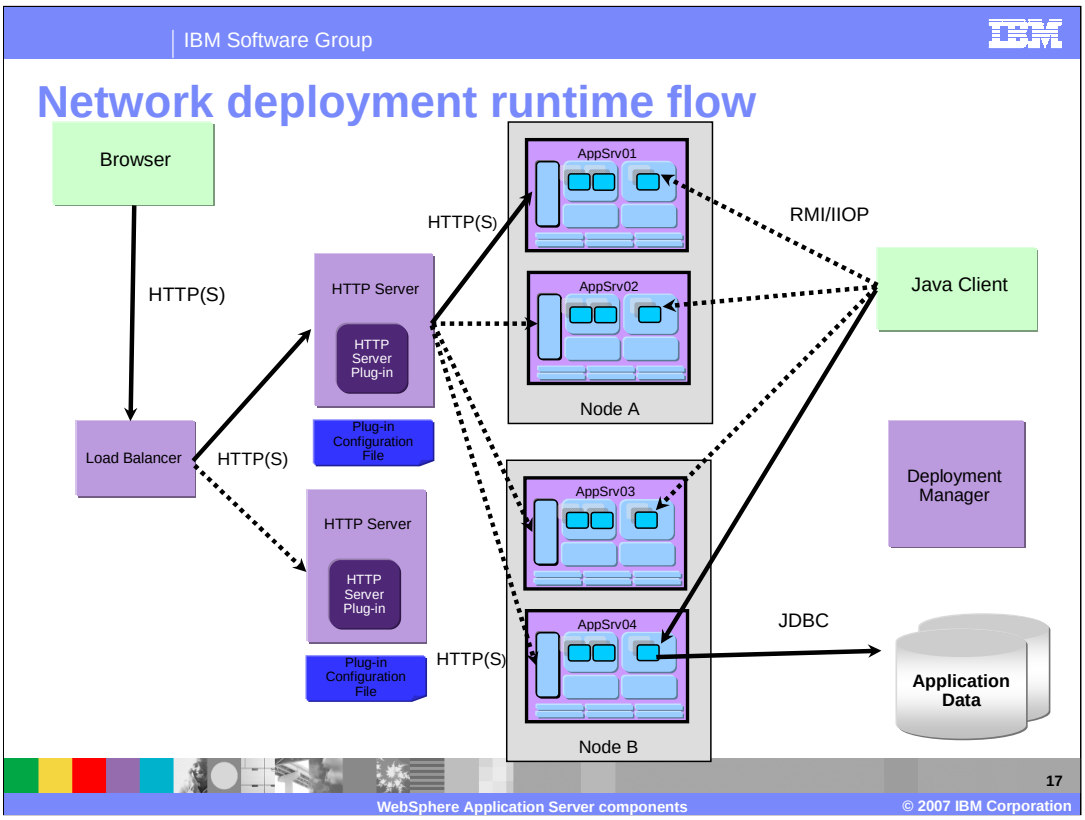
The performance monitoring infrasturec is use by WebSphere Application Server to collect data on runtime and applications. This infrastructure is compatible with and extends the JSR-077 specification and is used by performance monitoring tools to provide realtime information about the application server. PMI uses a client-server architecture. The server collects performance data from various WebSphere Application Server components and stores it in memory. This data consists of counters such as servlet response time and data connection pool usage. The data can then be retrieved using a Web client, Java client, or Java Management Extensions (JMX) client. WebSphere Application Server contains Tivoli Performance Viewer, which is integrated into the WebSphere administrative console and displays and monitors performance data. WebSphere Application Server also collects data by timing requests as they travel through the product components. PMI request metrics log the time spent in major components, such as Web containers, EJB containers, and databases. These data points are recorded in logs and can be written to Application Response Time agents that are used by Tivoli monitoring tools

Network deployment concepts

- A *node* is a logical grouping of application servers.
  - Each node is managed by a single *node agent* process.
  - Multiple nodes can exist on a single machine through the use of profiles.
- A *deployment manager* (DMgr) process manages the node agents.
  - Holds the configuration repository for the entire management domain, called a *cell.*
  - Within a cell, the administrative console runs inside the DMgr.

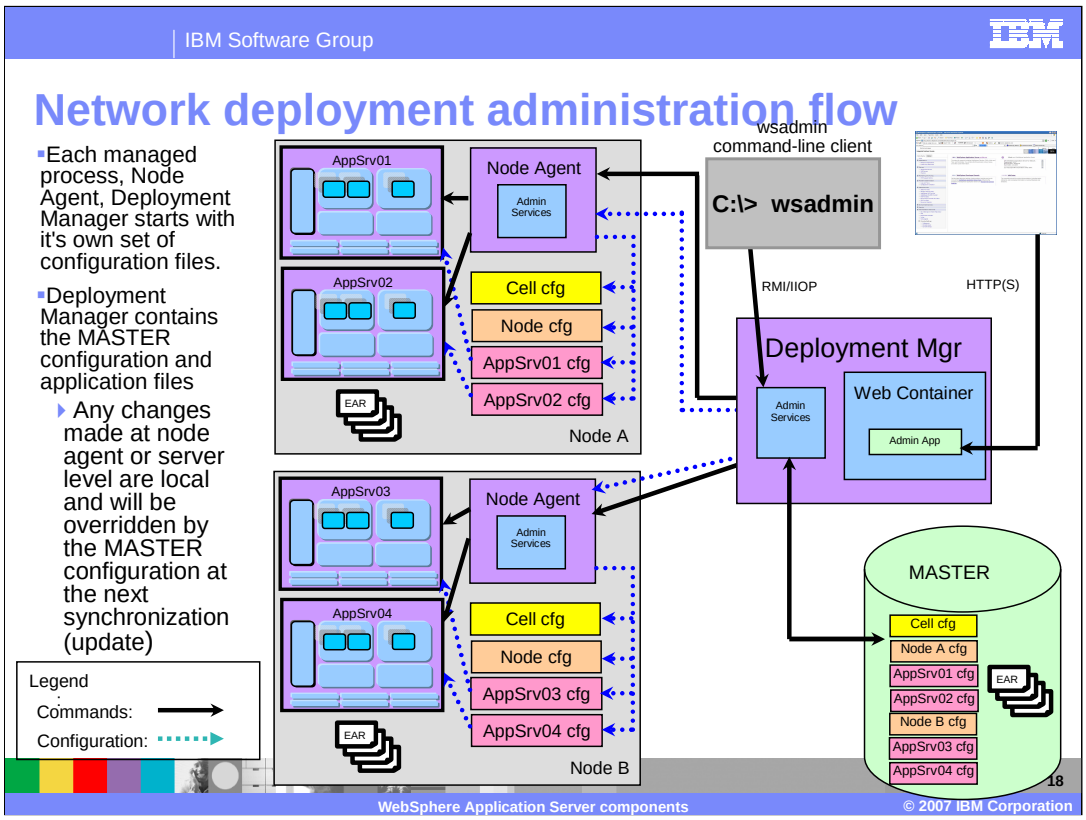


The Deployment Manager here is an application server that manages the administrative environment within a cell. You will see later in this unit that a node is represented by a profile. The node agent is a very important process that allows for communication of administrative information (commands and configuration files) to reach the applications servers.



The main theme with Network Deployment is distributed applications. While the "flow" of an application remains the same, there are significant additions to the runtime of an application. Note the "Load Balancer" this allows for multiple HTTP servers, users point there browsers to the load balancer and their request will be work load managed to an HTTP Server. Once the request hits one of these HTTP Servers, the HTTP Server Plug-in will load balance the request between the application servers that it is configured to serve. Once the request enters the application server, the flow is identical to how it was in WebSphere Express and Base.

The Java clients requests to EJBs can also be work load managed so that the requests do not all hit one application server.

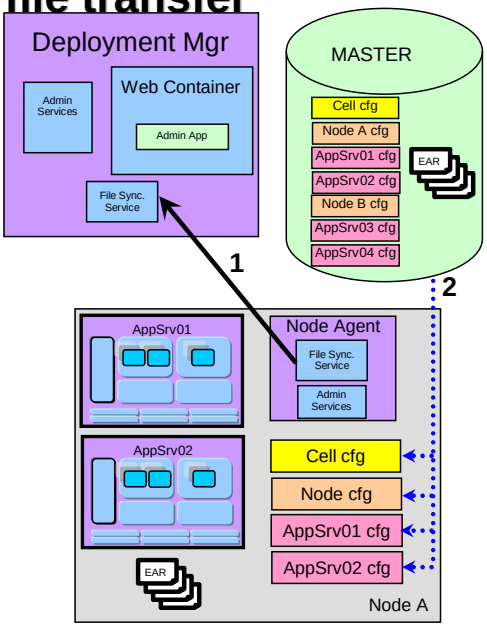


The administrative console and wsadmin are still the two ways that the environment is administered. However, take note that these tools now talk to the Deployment Manager and **NOT** to the application servers directly. The communication of these commands flows from the tools to the Deployment Manager to the node agents, to the application servers. This allows administration of multiple nodes (each possibly containing multiple application servers) from a single focal point (the Deployment Manager).

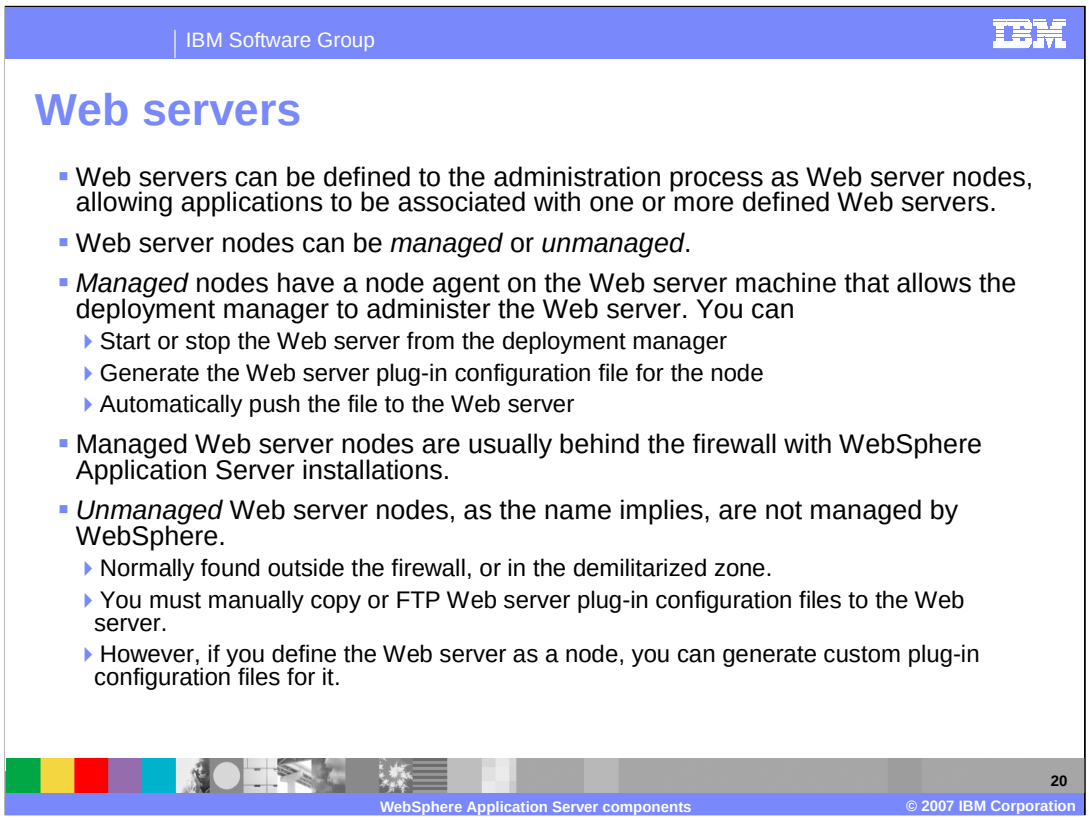
There is **ONE** main repository for the configuration files within a cell, and those are associated with the Deployment Manager. All updates to the configuration files should go through the Deployment Manager. You will see in a moment how this process works. You should be very careful in connecting to an application server directly with wsadmin or the administrative console as any changes that are made to the configuration files are only temporary, they will be overwritten with the configuration files from the MASTER files.

File synchronization and file transfer

- Deployment Manager contains the "master" configuration
- Node agents synchronize their files with the "master" copy
  - Automatically
    - At start up
    - Periodically
  - Manually
    - Administrative console
    - Command line
- During synchronization
  1. Node agent checks for changes to master configuration
  2. New or updated files are copied to the node



File synchronization service--the administrative service responsible for keeping up to date the configuration and application data files that are distributed across the cell. The service runs in the deployment manager and node agents, and ensures that changes made to the master repository will be propagated out to the nodes, as necessary. The file transfer system application is used for the synchronization process. File synchronization can be forced from an administration client, or can be scheduled to happen automatically. During the synchronization operation, the node agent checks with the deployment manager to see if any files that apply to the node have been updated in the master repository. New or updated files are sent to the node, while any deleted files are also deleted from the node. Synchronization is one-way. The changes are sent from the deployment manager to the node agent. No changes are sent from the node agent back to the deployment manager.



As a special case, if the unmanaged Web server is an *IBM HTTP Server*, you can administer the Web server from the WebSphere

administrative console. Then, you can automatically push the plug-in configuration file to the Web server with the deployment manager using

HTTP commands to the IBM HTTP Server administration process. This configuration does not require a node agent.

Web server plug-ins

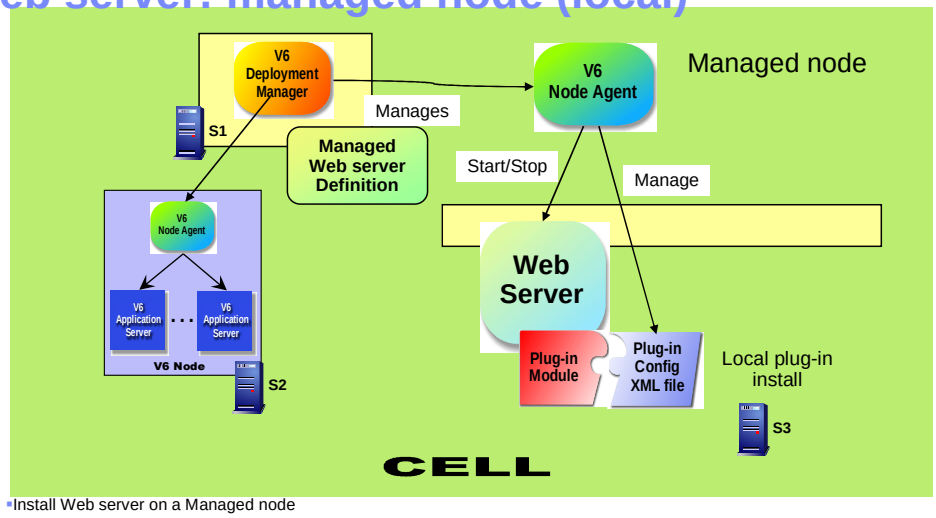
- Web containers can serve static content.
  - But more likely an external Web server receives client requests
- Web servers usually serve requests that do not require dynamic content such as HTML pages.
- When a request requires dynamic content, such as JSP or servlet processing, it must be forwarded to the application server.
- The Web server plug-in forwards a request based on its URI.
- The Web server plug-in is also responsible for load-balancing and failover among server cluster members using plugin-cfg.xml files.

You use a Web server plug-in to foward requests from the Web Server to the Application Server. The plugin is included with the WebSphere Application Server packages. You copy an Extensible Markup Language (XML) configuration file, located on the

WebSphere Application Server, to the Web server plug-in directory. The plug-in uses the configuration file to determine whether a request should be handled by the Web server or an application server. When WebSphere Application Server

receives a request for an application server, it forwards the request to the appropriate Web container in the application server. The plug-in can use HTTP or HTTPs to transmit the request.

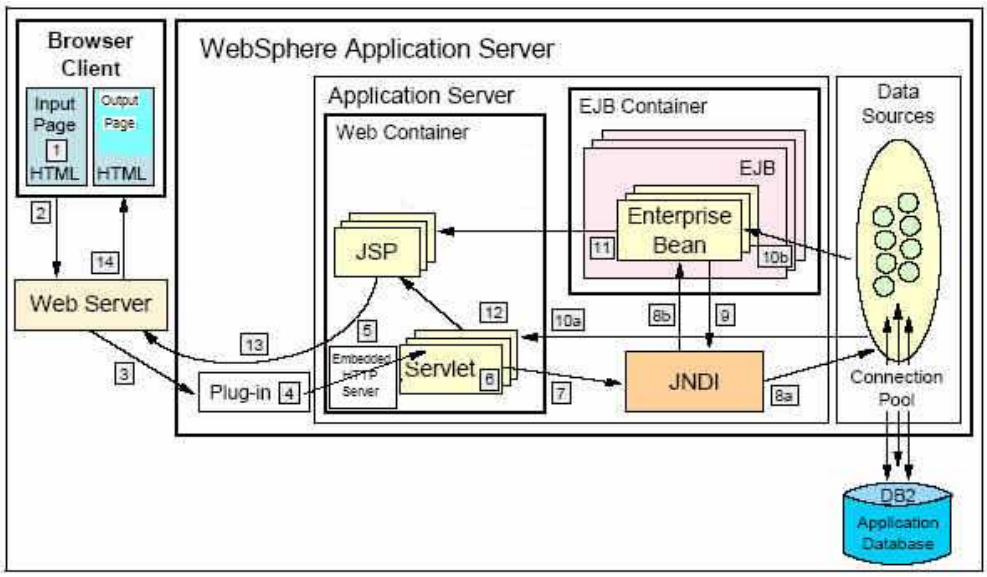
Web server: managed node (local)



- Install Web server on a Managed node
- Create a Web server definition within the DMgr
- Node Agent receives commands from DMgr to administer the Web server
- Plugin-cfg.xml file is propagated through the file synchronization service and lives under the config directory.

The web server is managed via the Deployment Manager through the Node Agent. This provides ability to start and stop the Web server and automatically push the plug-in configuration file to the web server from the DMgr. This can be used when the web server is on the same machine where WebSphere Application Server is installed. This is a common scenario for behind a firewall where a WebSphere Node can be installed.

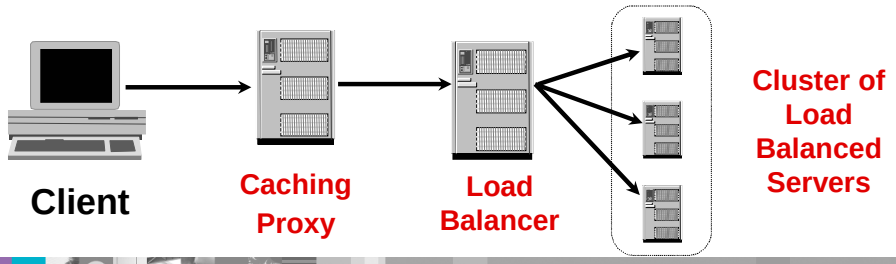
Typical client request application flow (1 of 2)



This graphic outlines the typical client request as it flows from the browser, through the application server, and then back to the client's browser. It is important to note that the flow can diverge in the Web Container depending on what the servlet needs in order to properly construct the response. If the request was for a simple JSP then the servlet can dispatch the request directly to the JSP and return the output page. Otherwise, the request will flow through the JNDI and to the appropriate EJB or datasources.

Edge components

- WebSphere Application Server Network Deployment package contains the following Edge Component functionality:
  - Load Balancer
  - Caching Proxy
- Edge Components install separately from WebSphere Application Server.
- Load Balancer is responsible for balancing the load across multiple servers that can be within either Local Area Network or Wide Area Network.
- The purpose of the Caching Proxy is to reduce network congestion within an Enterprise by offloading security and content delivery from Web servers and Application Servers.

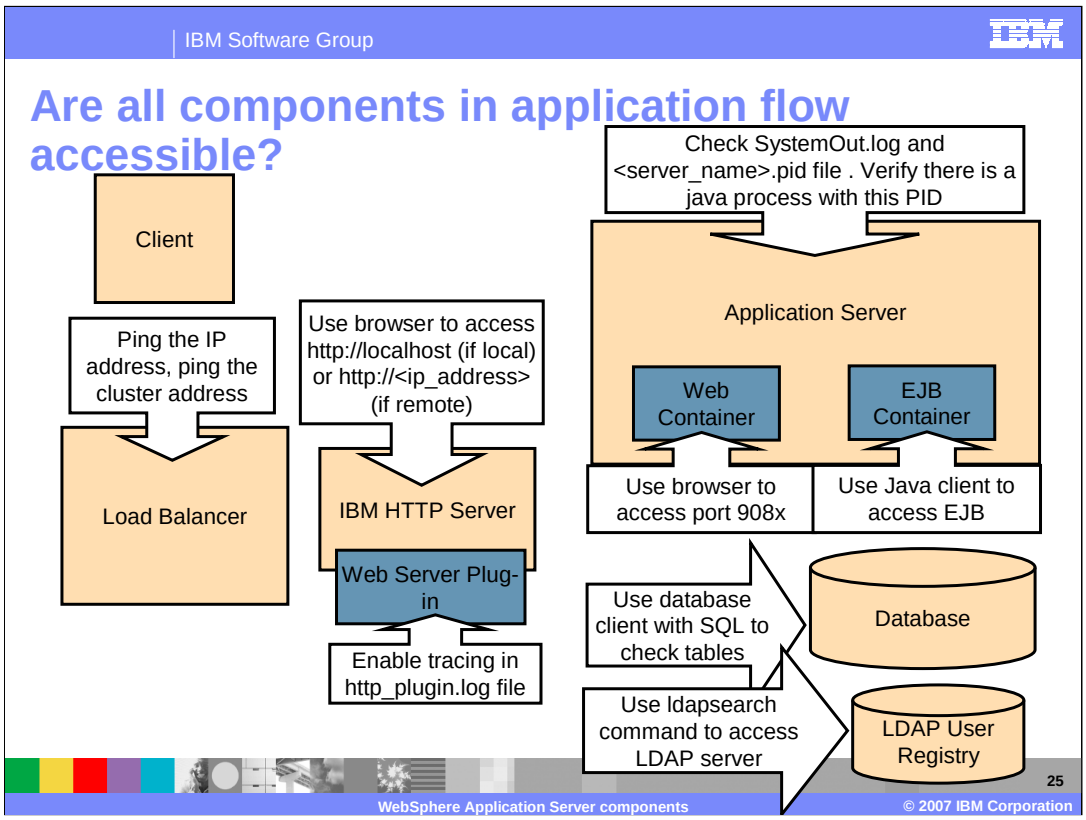


The ND version of WebSphere Application Server contains functionality for two key edge components, load balancers and caching proxies.

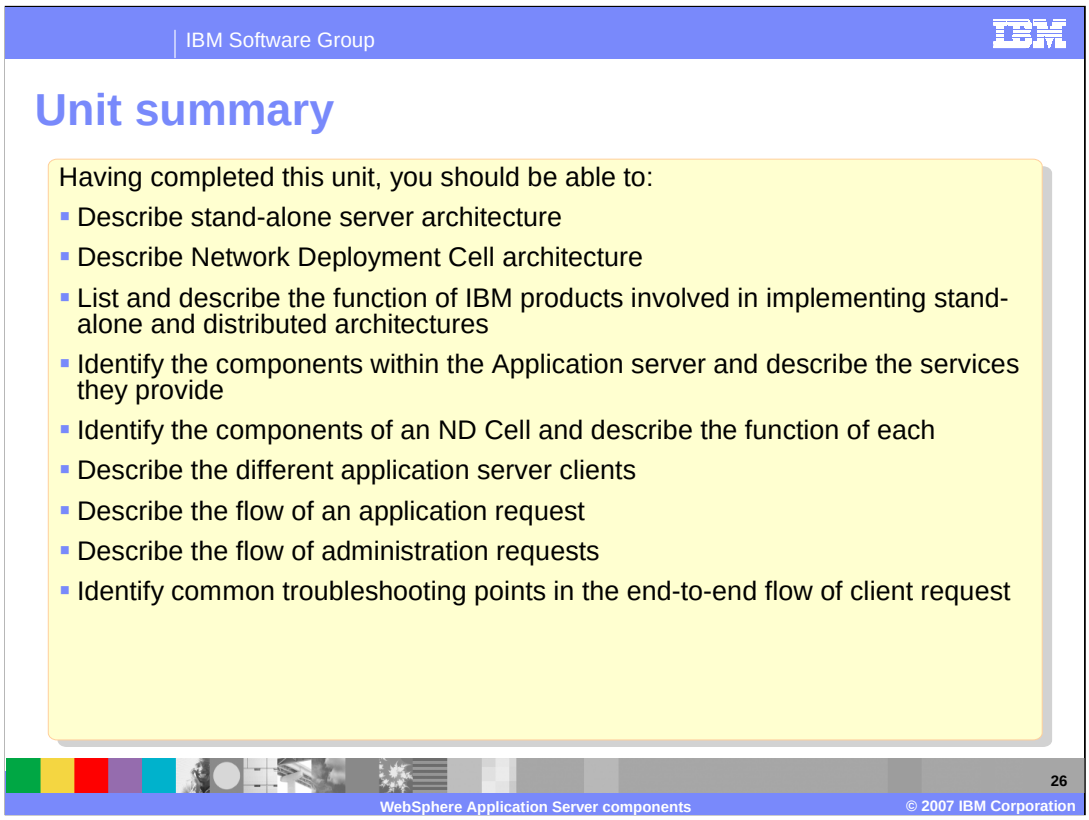
The Caching Proxy intercepts data requests from a client, retrieves the requested information from the application servers, and delivers that content back to the client. It stores cacheable content in a local cache before delivering it to the client. Subsequent requests for the same content are served from the local cache, which is much faster and reduces the network and application server load.

The Load Balancer provides horizontal scalability by dispatching HTTP requests among several, identically configured Web server or application

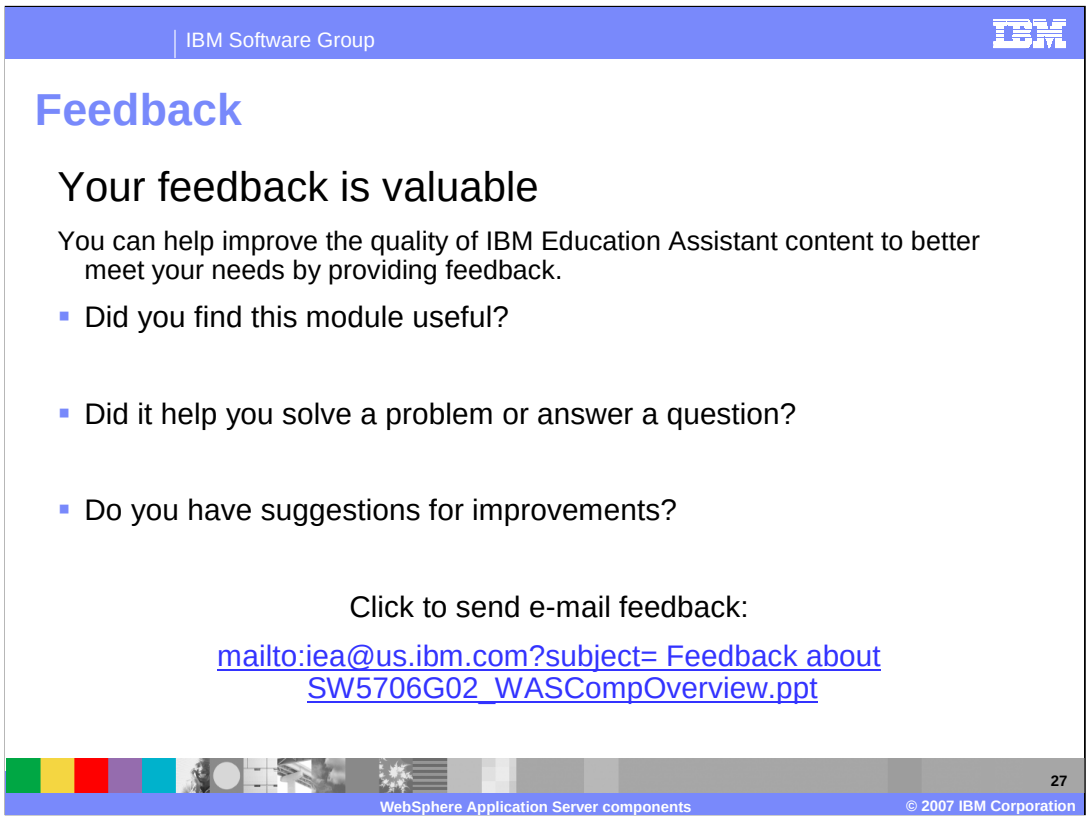
server nodes.



There are several components involved in hosting applications and so there are several different points where you should investigate an application flow. This slide outlines some of the ways in which you can verify that each of the components is on and operational. If any one component does not pass this check, and is involved in the application flow, then it is likely causing problems for users.



This unit covered the key components of WebSphere Application Server as well as how it interacts with other components involved in a user‘s request as it flows from browser to application server and then back to the browser. Understanding this will not only help you administer your WebSphere environment but be valuable to any problem determination you may need to perform.



You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere

EJB, Enterprise JavaBeans, J2EE, Java, Java Naming and Directory Interface, JavaMail, JDBC, JMX, JSP, JVM, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.