These slides are based in part on materials provided by IBM’s Academic Initiative.
Today

- HTTP Server
- Websphere Application Server (WAS)
- Websphere MQ
Short History of WebSphere

• WebSphere started out as HTTP server and became one of the first z/OS products running under the UNIX interfaces

• In 1997, a JVM plugin was added to run servlets

• Version 4 saw the move to J2EE branding

• Version 5 (2003) of the application server was synchronized with other platforms

• Today, WebSphere is not only a J2EE application server but middleware implementing the concepts of a Service Oriented Architecture, Enterprise Service Bus and Business Process Management

• Portable
Web applications on z/OS

In the past:

- Existing (pre-Web) applications tied to z/OS (CICS, DB2)
- New developments were made on other platforms

Now: integration of both types of applications on z/OS.
z/OS HTTP Server

HTTP Server can run in three modes:

- *Stand-alone server* – for simple Web sites
- *Scalable server* – for interactive Web sites with servlets and JSPs and dynamic traffic volume
- *Multiple servers* – combination of multiple stand-alone and scalable servers
Serving Static Web Pages

- Works mostly like any other HTTP Daemon – HTTP Server retrieves the file and sends it to the user with HTTP header
- Main difference is that HTTP Server usually first has to convert files from EBCDIC to ASCII
Serving Dynamic Web Pages

- HTTP Server supports CGI (each request requires a separate address space) and FastCGI (multiple requests managed in the same AS)

- HTTP Server provides a plug-in interface to execute servlets either inside of the HTTP Server AS or communicate with a J2EE Server (using WebSphere HTTP Server plug-in)
Example: CGI

1. http://www.myzseries.com/cgi-bin/test.cgi
2. httpd.conf
3. test.cgi
4. test2.cgi

HTTP Server Address Space
z/OS Address Spaces

Client Browser
Example: WAS plugin

- **HTTP Server**
  - URL: http://www.myzseries.com/my.jsp
  - Config: httpd.conf
  - WAS plugin
  - Servlet
  - Response

- **Client Browser**

- **CICS Server or IMS Server**
  - Config: was.conf

Example: WAS plugin with J2EE server

HTTP Server

J2EE Server

CICS Server or IMS Server

http://www.myzseries.com/my.jsp

was.conf

EJB Container

EJB

Servlet

Response

httpd.conf

Client Browser

URL
HTTP Server Features

- Performance and usage monitoring (using System Management Facilities (SMF))

- Tracing and Logging

- Server Side Includes (SSI), Cookies, HTTPS

- Simple Network Management Protocol (SNMP), Lightweight Data Access Protocol (LDAP)

- Persistent connections (HTTP 1.1), Virtual hosts, Proxy support

- Thread level security (independent security environment per client)

- Caching (HFS files, z/OS data sets, etc.)
Questions
WebSphere Application Server (WAS)

WAS has two main components:

- A plugin for the Web server (HTTP Server, Apache, ISS, ...)

- J2EE Application Server:
  - Java Virtual Machine (JVM)
  - Support for servlets, JSPs, EJBs, CORBA, ...
  - Support for DB2, CICS and IMS
  - Administrative tools: security, performance, scalability, recovery
Basic Model for WAS

- Web browser client
- Admin UI
- HTTP Server
- Application server
- Application database
WAS in Context
Accessing Server Resources

Diagram showing the flow of accessing server resources:
- Web browser connects to Embedded HTTP Server via HTTP(S).
- Web Services client uses SOAP/HTTP(S) to interact with the Web Services Engine.
- Java client uses RMI/IIOP or SOAP/JMS for communication.
- EJB Container and Servlet/JSPs are part of the Application Server, which also interacts with the Messaging Engine and contains Application Data.

Key components:
- Web browser
- Embedded HTTP Server
- Web Container
- EJB Container
- Servlets/JSPs
- EJBs
- Web Services Engine
- Messaging Engine
- Application Data
- JMS Client
- RMI/IIOP
- SOAP/JMS
J2EE Application Model (*)

- Functional
- Reliable
- Usable
- Efficient
- Maintainable
- Portable
Enterprise Application Packaging

- EJB Module .JAR file
- Web Module .WAR file
- Client Module .JAR file
- Installed RAR
- J2EE Application .EAR file
- Application DD
- EJB DD
- Servlet
- JSP
- HTML, GIF, etc.
- Web DD
- Client Class
- Client DD
Java on z/OS

This layer has the calls to the C runtime library.
Java in the z/OS Language Environment

Java program and JVM executing in a z/OS LE Environment

- Java Threads
- System Thread
- LE Heap Storage
- C/C++ main routine
- Subroutine
- Enclave specific data
- Data shared between enclaves

Heap
Thread stack
Method area

Runtime data area

Java program executing in a JVM

Process
LE Enclave

JVM
WAS Controller Region and Servant Regions

Application Server Instance = CR

SR
Controllers and Servants

- The Address Space in which the JVM resides is called a “Servant”
- Multiple servants maybe started by WLM
- Work is queued by a “Controller” Region
WAS Terminology

- Servers – CR and multiple SRs
- Nodes – logical grouping of WebSphere-managed servers sharing a common configuration (can not span LPARs)
- Cells – a grouping of nodes into a single administrative domain (can span LPARs)
- Cluster – multiple copies of the same component (server)
A “standalone” server is a single server in a single node in a single cell:
Separate Environments (Cells) using Standalone Servers

You can create as many of these things as you like, and they can be 100% operationally and administratively isolated from one another:

Even though they're on the same MVS image, their environments are separate from one another:

- Separate Mount Point and HFS
- Separate JCL procedures
- Separate TCP ports
- Separate Admin Consoles
- Separate Userids / Passwords
- Start/Stop servers independently

This is one of the primary benefits of creating Standalone Servers

There's another ...
Multiple Base Application Server Nodes / Cells

Multiple Base Application Server Nodes

- MVS System or LPAR
- Cell
  - Daemon CR
  - Node
    - Server Instance A CR
      - A
  - Browser
  - Cell
    - Daemon CR
    - Node
      - Server Instance B CR
        - B
      - Browser
    - Cell
      - Daemon CR
      - Node
        - Server Instance n CR
          - A
      - Browser

Multiple Base App Nodes is perfectly acceptable with WebSphere for z/OS Version 5

Involves running through ISPF for each Base App. Node

Each would have its own:
- Server Root (and HFS)
- Cell
- Node
- TCP Ports
- Admin interface

Implies great separation and isolation...
The Deployment Manager (DM)

- DM is a special kind of application server instance
- The administrative application runs in the DM
- DM manages application servers (CR/SR) grouped into Nodes
- The DM itself has its own node
- All other nodes have a Node Agent (with a CR)
WAS Network Deployment Overview
Clusters

Servers are clustered through the administrative interface.

Any given server may be a member of only one cluster at a time.
You cannot have Server_C be a member of two different clusters, for example.

Hybrid of vertical and horizontal is permitted.

“Vertical” Cluster
Two or more servers in the same system or LPAR

“Horizontal” Cluster
Two or more servers across multiple nodes (or systems)
Rules

- “Standalone” Servers need no DM
- Nodes never span LPARs
- DM needed to administer network deployment; nodes can continue to operate even if DM crashes!
- One DM per cell that uses network deployment
**WAS and WLM**

WebSphere uses three distinct functions of WLM:

- **Routing** – which server is best able to complete the work?
- **Prioritizing** – manage resources based on service level objectives
- **Queuing** – delay work until it can be processed
What are Connectors?

Objectives:

- Make the communication protocol / mechanism between client application and subsystem transparent to the application developer
- Provide portability of the client application
- Provide structured API and development tools for developers using connectors (J2EE Connector Architecture, J2CA)

Connectors may support remote operation (i.e. via TCP/IP) or use some form of optimized local communication.
Connectors and Resource Adapters

C=connector
RA=resource adapter
# Local vs. Remote Connectors

<table>
<thead>
<tr>
<th></th>
<th>Local Connection</th>
<th>Remote Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>No network overhead</td>
<td>network overhead</td>
</tr>
<tr>
<td><strong>Availability</strong></td>
<td>One LPAR</td>
<td>Multiple LPARs and network involved</td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td>Duplicate all components on another LPAR</td>
<td>Components can be distributed across LPARs, allowing better workload balancing</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>Thread identity can be used; no risky network connections</td>
<td>No thread identities; network connections require more security measures</td>
</tr>
<tr>
<td><strong>Transactionality</strong></td>
<td>2-phase commit supported</td>
<td>2-phase commit not always possible</td>
</tr>
</tbody>
</table>
Mainframe Connectors (*)

WAS supports the following connectors:

- CICS Transaction Gateway
- IMS Connect
- DB2 JDBC
Using J2CA to access CICS

Requires CICS Transaction Gateway; Version 6.1 supports:

• Local connections (only CTG libraries need to be present)

• Remote connections (requires CTG daemon)

• 2-phase commit supported for both local and remote mode
JDBC Driver Types

Type 1
- Java Program
- JDBC ODBC
- ODBC Driver
- DBMS

Type 2
- Java Program
- Java Driver
- Native Driver
- DBMS

Type 3
- Java Program
- Java Driver
- Daemon
- DBMS

Type 4
- Java Program
- Java Driver
- DBMS
Why run WAS on z/OS? (*)

- Consolidation of workloads
- Continuous availability
- Performance
- z/OS Security
- Integration with legacy applications
Questions
Styles of Communication

Conversational

Program A
Program B

Call and return

Program A

Messaging

Program A
Program B
Program C

Call and return
Messaging

The principal objective of messaging is to exchange information:

• in an asynchronous way
• offering assured delivery
• offering failure independence of applications
• offering a triggering mechanism on the receipt of messages
• providing message persistence, transactional integrity and security
WebSphere MQ

- Implements messaging and queueing
- Provides a common set of APIs independent of platform or network complexity
- WebSphere MQ is available on over 40 platforms
- Enables a loosely-coupled form of application integration
- Enables flexible placement of business logic
  ⇒ suited for a Service Oriented Architecture
Examples

‘Send and Forget’

Program A

Put Invoice-Q

Invoice-Q

Get Invoice-Q

Program B

Request / Response

Program A

Target Queue

Program B

Reply-to-Queue
Examples
Publish/Subscribe

Pub/Sub Broker

Data

Subscription

(re-) Publication

A
B

A
B

A
B

A
B

A
B

A
B

A
B

A
B

C
D
E
F

Data
Time Independence

Everything Constantly Available

Everything Conditionally Available

Busy

Not Available

Queue

Logical Connection

Message/Queuing Environment
Synchronous Communication Model
Asynchronous Communication Model
The Message Queue Interface (MQI)

- Major calls
  - MQCONN
  - MQCONNX
  - MQOPEN
  - MQCLOSE
  - MQPUT
  - MQPUT1
  - MQGET

- Minor calls
  - MQBEGIN
  - MQCMIT
  - MQBACK
  - MQINQ
  - MQSET
Transparency of Distributed Queueing

System 1
- Program A
- MQPUT Q2
- MQPUT Q1
- MQGET Q1
- Queue Manager 1
- XQ
- Q1
- MCA

System 2
- Program C
- MQGET Q2
- MQI
- Queue Manager 2
- Q2
- DLQ

Network

MQI
What is a Message?

- Header
- User Data

A Series of Message Attributes
Understood and augmented by the Queue Manager
- Message Id
- Correlation Id
- Unique Routing information
- Reply routing information
- Message priority
- Message codepage/encoding
- Message format
  ....etc.

- Any sequence of bytes
- Private to the sending and receiving programs
- Not meaningful to the Queue Manager

- Message Types
  - Persistent ... recoverable
  - Non Persistent

- Up to 100MB message length
Types of Messages

- Datagram (send and forget)
- Request (send, wait for reply)
- Reply (response to request)
- Report (status, failure, etc.)
What is a Queue?

- Place to hold messages
- Different Queue Types: Local, Alias, Remote, Dead-letter
- Can be predefined or dynamically defined
- Access can be FIFO, Priority, Direct, Destructive or non-Destructive, ...
- Parallel access is managed by the queue manager
The Queue Manager

MQ API

Kernel

Local Queuing

Dist. Queueing

Put

Get
MQ Channels

Network

Program 1
get
MQI
put

Local Q2
Remote Q1

QMA
XmitQ
QMB

Channel QMB.QMA
Receiver

Channel QMA.QMB
Sender

Program 2
get
MQI
put

Local Q1
Remote Q2

QMB
XmitQ
QMA

Channel QMB.QMB
Receiver

Channel QMB.QMA
Sender
MQ Clustering for Improved Availability
MQ Clustering using Coupling Facility
Shared Queues and Server Failure

• Failure isolated to failed entity

• Automatic peer recovery for failing queue managers:
  – In-flight MQPUTs and MQGETs are rolled back
  – No marooned messages!
Security

• MQ provides access control to its objects (queues, commands, messages)

• MQ provides channel security using SSL

• On z/OS, MQ uses SAF to map MQ security to native platform security (such as RACF)

⇒ RACF can map an SSL certificate to a userid
Transactional Support

- MQBEGIN, MQCMIT and MQBACK control transactions

- Messages and other transactional resources can be:
  - Managed by a Transaction Manager (WAS, CICS, IMS, ...)
  - Managed by WebSphere MQ (is also a TM)
Example

MQPUT CAR RENTAL
MQPUT FLIGHT
MQPUT HOTEL
MQGET Reply-to-queue
Car rental
Flight
Hotel
Car
Flight
Hotel
MQPUT
MQPUT
MQPUT
MQPUT
MQPUT
MQPUT
MQPUT
MQGET Reply-to-queue
MQ and the “Big Picture”
Questions