WebSphere Application Server for z/OS

Moving Apps to WebSphere z/OS

Based on WP101093
ibm.com/support/techdocs

WebSphere for z/OS Support Team
IBM Washington Systems Center
dbagwell@us.ibm.com

IBM Washington Systems Center
This presentation is a companion to the WP101093 white paper that’s offered on the IBM Techdocs website at ibm.com/support/techdocs. That white paper provides a more detailed discussion of the topics presented here, and the paper is designed to be more of a “design review checklist” format. It’s good information, but it makes for somewhat dry reading because of the way it’s designed. This presentation is designed to provide a more conceptual view of things.

The WP101093 Techdoc article also provides an MP3 recording of this presentation.
This Powerpoint has Speaker Notes

If you’re working with the source Powerpoint, be sure to consult the speaker notes that are under each slide:

They’re not quite as extensive as White Paper

(Powerpoint speaker note functionality is somewhat limited in how much can be provided)
Introduction and Objectives

This presentation is about moving an application from distributed up to z/OS.

Myth: “Applications need to be rewritten to run on WebSphere z/OS”

Fact: A well-written application will move to z/OS and run without any modifications … provided the two environments are configured the same way, and the application is designed to run efficiently.

This presentation is meant to cover the most common causes of failure when moving an application to WebSphere z/OS

The purpose of this presentation (and the white paper) is to help you understand what’s involved when moving an application from WebSphere on the distributed platform up to WebSphere on the z/OS platform. We’re making the assumption the decision to do that has already been made; we’re not so much trying to make the case to do that as we are explaining what to be aware of to make the move as successful as possible.

One of the things we see out in the world is this persistent myth that J2EE applications must somehow be written “differently” for WebSphere on z/OS. Or, stated another way, that applications have to be “rewritten” or “migrated” or “ported” to WebSphere on z/OS. That’s not true.

Note: to be fair, some of that might be from way back in the days of WebSphere for z/OS V4, where the some application redesign was necessary. That was then, this is now.

With WebSphere V6.0 and above, the various platforms on which WebSphere is offered have been aligned so that now “WebSphere is WebSphere” (more on this in a bit). So today, a well-written application will move to WebSphere z/OS without any modifications. But not all applications are well written. Further, applications often require various customization to the runtime for the application to work properly. That means the distributed WebSphere platform and the target z/OS WebSphere need to be configured in the same way.

That’s what this presentation (and the white paper) are all about -- helping you understand what to look for and review so you can minimize the chances the application fails to meet expectations when moved to WebSphere on z/OS.
Why WebSphere on z/OS?
This presentation assumes the decision to move the application to WebSphere on z/OS has already been made. Here’s a brief summary of “Why z/OS?”

The Value of “Just Showing Up” on the Platform
- Proven hardware and I/O subsystem design
- Advanced operating system provides isolation and protection
- Optimization such as hyper-channels and local TCP
- Sysplex Distributor and DVIPA for intelligent load balancing and high availability design
- Workload Manager (WLM) and Intelligent Resource Director (IRD) -- coordination of shared resources
- Well-established set of systems management tools and procedures

WebSphere z/OS Exploitation of the Strengths of the Platform
- WLM/RMF integration for classification of workload and reporting
- SMF Type 120 for capacity planning and chargeback
- Resource Recovery Service (RRS) for transaction management and recovery
- Vertical scaling with multiple servant regions, controlled by WLM based on defined goals
- Cross memory communications for local connectors to data (CICS, IMS, DB2)
- Security Access Facility (SAF) for userids, groups, roles, keyrings, certificates

WebSphere z/OS provides a proven integration platform for application and data
That said, from application perspective WebSphere is WebSphere …

This presentation is not designed to make the case for WebSphere on z/OS. This presentation assumes that decision has already been made. But for the sake of completeness we’ll offer one chart on the topic, and encourage readers to get with their IBM System z specialist to go over any of these bullet points if they’d like to learn more.

We’re going to organize these bullets into two groups: those thing related to WebSphere simply being on z/OS; and those things related to WebSphere actively exploiting strengths of the z/OS platform. It’s important to understand that WebSphere for z/OS is doing a fine balancing act -- it is striving to maintain a common code base with the other platforms and striving to exploit the z/OS platform. It does this with a small portion of its code being “platform unique.” However -- and we’ll see this on the next chart -- that “platform unique” code is well hidden from the application. The application will definitely benefit from it, but that stuff will not directly impact the J2EE APIs, methods, services and functions. At that layer “WebSphere is WebSphere.”

We’re not going to walk through each of the bullets in the list above. It would take far more space than the Powerpoint speaker notes permit. Here’s the bottom line(s):
- The System z hardware architecture and the z/OS operating system architecture is proven, reliable and robust solution. If you didn’t already more or less believe that you wouldn’t be considering moving your application there.
- WebSphere for z/OS is designed to benefit from and exploit those strengths.
- WebSphere for z/OS implements the same J2EE services and APIs as WebSphere on the other platforms when the version and release is the same.
- Therefore, WebSphere on z/OS provides the ability to integrate application and data into a single place, with all the benefits of System z and z/OS operating with it.

We must now make sure everyone understands a key point … from an application perspective, WebSphere is WebSphere.
The whole purpose of an “Application Server” is to provide an application hosting environment that provides common services and APIs:

- Functions, Services and APIs Provided by Version of Product
- Customized Configuration Settings
- Java SDK
- Platform-Specific Code
- Platform Operating System
- Platform Hardware

Application

Application more directly affected by this

The application does not “see” these layers

When considering a move from one platform to another, the key is making those top three layers as equal as possible

Lots of things can get in the way of that:
- Different WebSphere Versions
- Different Customized Settings

The whole purpose of an application server is to provide applications a place to operate where common services, functions and APIs are provided. This allows developers to focus on the business solution and not have to write and rewrite “plumbing.” WebSphere is an application server … a J2EE application server to be precise.

From the application perspective the environment in which it runs consists of a set of J2EE functions, methods and APIs. What exactly those consist of depends on the version and release of WebSphere, as new function is being added all the time. Underneath that there will be a set of configuration settings you, the administrator of WebSphere, will have made. Those consist of things like data connectors, resources with defined JNDI names, CLASSPATH updates, WebSphere variables. And finally there’s the Java SDK which provides the Java runtime infrastructure. These things more directly affect the application.

Under that is platform specific code -- code written to exploit some element of the platform on which the copy fo WebSphere is running. Under that is the platform operating system itself, and under that is the hardware. These things are not “seen” by the application. The application will benefit from this, but the application does not directly interact with this platform-unique code.

So the key is making sure the top three “layers” in this picture are as equal as possible between the distributed WebSphere and the WebSphere up on z/OS. If the two are identical (from WAS 6.0 and higher) then the chances of the application running right out of the chute are higher. If the two environments are different then the probability of some problem surfacing is higher.
Thought Exercise to Start the Discussion

Consider this environment. Would the app likely move and work well?

“Source”
WebSphere for Windows
Version 6.1
Highly Customized
All data needed is accessible
Finely tuned

Windows to Windows

= “Target”
WebSphere for Windows
Version 5.1
Default configuration
Data not accessible
Default tuning

Answer: probably not. There are considerable differences between the “source” and the “target” platforms. Many things may affect the success of the move of the app from one platform to another.

The concept of moving the application to WebSphere z/OS is the same -- you have to make sure it’s “apples to apples” when the application is moved up to z/OS. That’s what this presentation will cover.

To illustrate the point made on the previous chart, here’s a thought exercise. Imagine two copies of WebSphere Application Server, both Windows based. Now imagine the place where the application is currently running is a customized copy of WebSphere 6.1. Further, that copy of WebSphere has access to all the data the application needs, and the WebSphere has been tuned so the application runs well.

Now imagine you’re planning on moving that application to another copy of WebSphere … again, also on Windows. But this copy of WebSphere is downlevel -- Version 5.1. Further, the installation of that is all default … no customization settings have been made. Will the application move over well?

Answer -- no. Too much is different.

And that same concept applies to moving the application to z/OS. Provided the “source” and “target” are more or less equal, it eliminates a lot of the issues involved with moving the application. The focus then can be on application design optimization for high volume. That’s what we’ll cover at the end of this presentation.
Two Fundamental Issues

This topic can be organized into two fundamental areas of exploration:

**Differences between “source” and “target” platforms**
- Different levels of the WebSphere product code
- Configuration differences between the two

**Problems with the application itself**
- Coding practices that tie the code to a specific platform
- Inefficient code that shows its faults under high load

What we'll do is provide a structured walk-through of the most common issues

White Paper provides a “review checklist” format

As we consider the basic issue of preparing to move a WebSphere application from a distributed platform up to z/OS, we face two fundamental issues:

- Differences between the “source” and “target” platforms. These differences can be in the level of WebSphere present on each; or the configuration of WebSphere in both places.
- Issues with the application itself. A common problem are coding techniques -- not recommended, but often used -- that tie an application to a specific platform. The other issue is application designs that seem to work okay at low loads but show their problems when subjected to the higher traffic seen on WebSphere for z/OS.

So what we’re going to do in this presentation is give you a kind of structured walk-through of the issues we’ve seen. The white paper provides this information in more of a “planning review” checklist format.
Six Areas of Concentration

We’ll focus on six areas for uncovering potential issues that could arise when the application is moved to WebSphere for z/OS

• **WebSphere z/OS installation readiness**
  WebSphere for z/OS not really installed properly or ready to accept the application

• **Functional difference and specification difference analysis**
  The level of code is different, which may create problems with function not there, or things having been “deprectated”

• **WebSphere z/OS runtime definition and data resource readiness**
  Configuration customization done on the distributed platform not mirrored on the z/OS platform. The application doesn’t work because configuration definition it counts on isn’t there

• **WebSphere z/OS performance tuning readiness**
  WebSphere is installed on z/OS but the environment isn’t tuned to operate properly. Disappointment in overall environment when application is brought to z/OS.

• **Cross-platform portability coding practice readiness**
  The application is written in such a way that it’s tied to a specific platform. Best example: use of drive letter file specifications: C:\temp\myfile

• **Application design efficiency readiness**
  The application has design inefficiencies that weren’t uncovered on distributed because the application was never subjected to heavy load. But up on z/OS and under load the inefficiencies show themselves.

We’ll organize this presentation into six categories of things to focus on. Our objective is to reduce the probability of failure by doing a systematic review of the environment and application.
Installation and Configuration
Customization Issues

Making the source and target as equal as possible so the application has what it needs to operate properly.
WebSphere z/OS Installation Readiness
There’s more to it than doing just the SMP/E install …

- SMP/E Install Work
  - Standard SMP/E
  - Post-install System Programmer work
    - Relatively simple checklist of common things
    - Don’t want to discover this at last minute
  - Plan and create the customized jobs
    - Biggest challenge for people new to WebSphere
      - ibm.com/support/techdocs
      - PRS1331
      - WP100653
      - others
  - Run the customized jobs and start servers
    - Easy … unless something comes back not RC=0.
  - Install sample app and validate environment
    - Basic validation
      - Where you want to be at a minimum

Then there’s the configuration customization
Data connectors, JDBC providers, data source definitions, etc. More in a bit.

More than once we’ve seen cases where it was believed WebSphere was installed and ready to accept an application, but in reality all that had been done was the SMP/E work. For system programmers unfamiliar with WebSphere, they may well believe that laying down the SMP/E libraries and doing the checklist of sysprog work is all that’s needed. But in truth there’s more to it. Once that work is done, the runtime needs to be planned and built.

The third box in the chart above is what typically proves to be the stumbling block for those unfamiliar with WebSphere for z/OS. That step involves planning out names and ports, then creating the customized jobs that actually construct the cell. It’s not hard once you get the hang of it, but it’s definitely difficult for those who are new to it. Techdoc resources exist to help with this. But the key is this: you definitely do not want to try to do this at the last minute, under pressure to meet some deadline.

Ideally you want to make sure your WebSphere z/OS runtime is built and validated long before the application is to be moved up there.

Even with the runtime “skeleton” built, there may still be -- and likely will be -- customized configuration work that needs to be done. We’ll cover that in a few charts.
Version, Release and Maintenance Levels of WebSphere

It is best if the two platforms are at the same level. That helps insure functional compatibility.

Better

Version, Release and Maintenance the same
Level Ground

- Function and specification levels will be the same
- Application should not encounter functional disparity issues
- Other issues may arise … more to come

Source platform at lower level from target platform
Going Forward

- Function available at lower level may be deprecated at higher level
- Function available at lower level may be removed at higher level
- Application may run depending on what it does (but it may not)
- Should at minimum do review of functional differences between levels

Worse

Source platform at higher level from target platform
Going Backward

- Function available at higher level may not be present on lower level
- Application may not work if it uses new function to present on target
- Should review code and compare against function provided by WAS

WP101093 white paper on ibm.com/support/techdocs has URL pointer for functional support and what function has been deprecated and removed

In a perfect world you’d make sure the version and release and maintenance of WebSphere is identical on both platforms -- source and target. If you do that, and WebSphere is Version 6.0 or higher, then you can be assured that the function of WebSphere on both is aligned. Any J2EE API or method used by the application will exist on both the source and target WebSphere platforms.

But what happens if they're different? There are two possibilities here:

- The copy of WebSphere where the application runs now is at a lower level than the target WebSphere z/OS platform. This implies the move is “uphill” to a higher level copy of WebSphere. The things you might run into there are things the application uses that have been deprecated or removed in the newer version of WebSphere. The application will probably still run, but depending on how far apart the two levels of WebSphere are, it may not.

- The copy of WebSphere where the application runs now is at a higher level than the target WebSphere z/OS platform. This is the worst case … it implies moving backwards. Here you may run into function used by the application that simply wasn’t part of WebSphere at the earlier level. The code may run okay, but it may also throw many errors.

The white paper on Techdocs offers URL pointers to locations in the InfoCenter that provide a far more comprehensive review of what function has been deprecated and what function has been removed. You should at minimum review your code before moving it to WebSphere if the two levels of WebSphere are different.
**Runtime Configuration Customization**

These are things done *after* initial construction, typically through the Admin Console (or WSADMIN). An example to paint the picture for you:

1. If the application will access data through, say, a JDBC data source, that data source needs to be defined. Also, it may be that the application’s binding to the JNDI name is in the EAR file’s deployment descriptors, and it may be best to make sure the same JNDI name used on distributed is in play on z/OS as well.
2. JDBC providers (as an example) require WebSphere variables to resolve the location of the JDBC provider driver code. For DB2, those variables are present but empty by default.
3. The JDBC driver code needs to be there. If on distributed you used a particular driver code, that driver code may need to be made available on z/OS as well.
4. The application architecture may be such that it relies on EJBs packaged separately from the application. If that’s the case, then those utility EJBs need to be in the runtime for the application to work.
5. Some applications expect Java class and other native libraries to be available to it. This involves two things, really: update of the **CLASSPATH** and **LIBPATH** to include where those files are, and making sure the files are actually uploaded to the z/OS system.

The point of this chart is to illustrate that an application may need many other things defined for it to operate as expected. When that application was deployed on the distributed platform, those definitions were put in place in WebSphere on that platform. To move the application to another platform -- any platform, not just z/OS -- requires that similar definitions be made.

Any customized runtime settings required by an application must be present.
Some thought needs to be given to which user registry WebSphere z/OS will use. The default for the target configuration may be SAF. LDAP is possible.

Either will work for WebSphere z/OS

- If you choose to go with LDAP, WebSphere z/OS will need to be configured to use LDAP. *Likely won’t be “by default.”*
- If you choose to go with SAF, then you may need to load the user data from LDAP into SAF to support the application and its users

It’s a matter of making sure you have this thought out properly

If you intend to run WebSphere with security on -- which is what virtually everyone does in production mode -- then WebSphere is going to need to be made aware of where the “User Registry” is located. The User Registry is where user IDs, group IDs and user-to-group associations are maintained.

On the distributed platform that user registry will most likely be something such as LDAP, or perhaps Microsoft Active Registry. WebSphere on z/OS may use an external registry such as LDAP -- and indeed the very same registry as used before -- or it may use SAF (the IBM product implementation for SAF is RACF).

We bring this to your attention because the “default” for WebSphere for z/OS -- “default” meaning the setting that would occur if someone just clicked through the configuration and didn’t make too many changes -- would be SAF. The WebSphere on z/OS runtime can be configured after the fact to use the external registry. Or you can use SAF.

What you can’t do is ignore this issue … you have to think it through to make sure things work okay.

If you want to continue to use the external registry, then it implies configuring WebSphere z/OS to make use of it. It’s a cell-wide property that defines the user registry, so changing this should only be done if you’re certain nobody is already using SAF as the registry for the cell. If you go with the external registry it can be the very same one used for distributed. That would make the application move to z/OS the most seamless.

If you want to use SAF, you may do that as well. It would mean creating in SAF the same userid, group and user-to-group associations that exist in the current external registry.

Again, our main point here is that you should at least give thought to this so you don’t end up having authentication failures because WebSphere for z/OS goes to SAF and the userids aren’t defined there.
Application Role Authorization

If an application has defined “roles,” then WebSphere will look to enforce the roles*. How WebSphere z/OS is configured will affect whether this works.

“Roles” are used to determine the authorization users have to do things within the application. Two parts to this: (1) the definition of the role in the application, and (2) the assignment of users to the roles.

On distributed this is done by creating application bindings that define the roles and the groups authorized to the roles:

![Diagram showing App, Bindings (role names and the groups authorized to the roles), and LDAP]

This can work the exact same way on WebSphere for z/OS!

No change to application; no change to LDAP

The issue is that by default** WebSphere z/OS will look to use SAF EJBROLEs. That means roles would need to be defined in SAF.

1. Configure WebSphere z/OS cell to not use SAF EJBROLEs. Configure user registry to be LDAP. Then application bindings will take effect and role-to-user mappings works as it did on distributed.
2. Define EJBROLEs in SAF and provide user mappings. No changes to app required.

* Not all applications make use of roles

** There is a configuration option at cell creation time to not use EJBROLEs. But most people skip right past that and end up with a cell that has SAF EJBROLEs defined. That’s why we say “by default.”

Another topic in the realm of security that we need to bring up is the issue of application role authorization. Not all applications make use of “roles,” but if the application being considered for the move does then you need to consider the implications.

On the distributed platform the application will make use of “application bindings” (information contained in deployment descriptors). The bindings contain the role names and, most commonly, the groups assigned to the roles. When a user attempts to use a function of the application that is protected by a role, WebSphere will check to see if that user is authorized -- which means checking to see if the user has been given access to the role. On distributed this is done most typically by consulting with LDAP (or whatever the external registry is) and seeing if the user is a member of the group that’s been defined as having access to the role.

Imagine the user SMITH tries to use a function, and that function is protected with role MANAGER. In the application bindings the role MANAGER is defined so group GREEN has access. WebSphere then checks with LDAP to see if SMITH (the user) is a member of the group GREEN. If so, then SMITH has access to role MANAGER, and may therefore use the function.

On z/OS the “default” way this is done is with SAF EJBROLE definitions. The key to understanding the issue here is to know that if the WebSphere for z/OS cell is configured to use SAF EJBROLE definitions, then the bindings in the application deployment descriptors will be ignored. WebSphere will go to SAF and check the EJBROLE definition.

What this means is that the roles need to be defined in SAF. Or WebSphere z/OS needs to be configured to make use of the external registry rather than SAF EJBROLEs. If you do that latter, then the bindings in the application will again be used, and the behavior will be more like what was seen on distributed.

You can do either … we bring this to your attention so you don’t encounter authorization failures because you believed the behavior on distributed would be mirrored automatically on z/OS. It can be mirrored, but it needs to be configured to do that.
Platform and Runtime Performance Tuning

Making the target runtime perform well
Three Areas of Performance Tuning Focus

There are three broad areas of focus when considering how well an application will run on the z/OS platform:

1. Tuning of things related to the operating system and the WebSphere runtime structure
   - WLM, UNIX Systems Services, the type of connector used, etc.

2. Tuning of the JVM inside the application server where the application runs
   - Optimizing the garbage collection cycles

3. Tuning of the application itself so it operates efficiently
   - WSC experience indicates most “problems” show up in this space. More on this in the next section of the presentation.

We’ll focus on the first two in this section, then the third one in the final section of the presentation.

Performance is a large and sometimes complicated topic space. We most definitely do not want you to think this presentation will serve as the definitive performance analysis and tuning guide. There are lots of resources available to help with technical specifics in this space. That said, there are three broad areas of focus we want to draw your attention to:

1. Tuning efforts aimed at the operating system and the WebSphere for z/OS runtime -- these are things like WLM settings, UNIX Systems Services settings in BPXPRMxx and the use of Type 2 connectors rather than Type 4.
2. Tuning efforts aimed at the Java Virtual Machine (JVM) of the server -- and specifically the optimization of what’s known as “garbage collection” within that JVM.
3. Tuning efforts aimed at the application itself -- this is where the Washington Systems Center has found significant performance gains can be found, particularly when an application is scaled to very high levels of traffic and throughput.

We’re going to focus on the first two in this section, then turn our attention to the third one in the final section of the presentation.
General System Setup and Tuning

These are things related more to supporting the general operations of the WebSphere runtime rather than the application specifically:

- **Controller**
- **Servant**
- **AppServer**

**General System**

- Does the system have sufficient resources to run WebSphere in an acceptable manner?
- No IEFUSI exits limiting WebSphere
- BPXPRMxx updates such as MAXFILEPROC and MAXSOCKETS
- Configuration file system should be owned by the same system where servers in the node are running

**WLM**

- Controller regions: “High Importance” and “High Velocity”
- Servant regions high enough to start quickly but lower than controller regions
- Classify controller jobnames with OMVS rules so applyPTF.sh can run properly

http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS2853

**Fairly standard system programmer things. See white paper for pointers to more detailed resources for tuning these things.**

These things relate more to the general operations of the servers rather than the applications that run inside those servers. We’ll focus on two areas. In both cases, the white paper has pointers to where you can get more information on specific tuning activities.

- **General System tuning** -- one of the most basic things is whether WebSphere for z/OS has enough resources to run well. We’ve seen a lot of cases where people bring up WebSphere on a terribly overtaxed system, and wonder why it doesn’t perform well. Similarly, when first configuring WebSphere, we’ve seen many cases where IEFUSI exists are in place and limit WebSphere’s ability to get memory to operate.

  There’s a couple of BPXPRMxx tuning values that are recommended. Finally, the configuration HFS (or ZFS) for the nodes ought to be owned by the MVS system where the server runs. Server startup, in particular, can be perceived as quite slow if the HFS is owned by SYSB when the server is starting up on SYSA, for example.

- **WLM** -- WLM will control the amount of system resource available to WebSphere, and it’ll do that in relation to the goals set for other system users. If the WLM settings for WebSphere take a much lower priority than other things, WebSphere will not appear to perform well. The recommendation is that controller regions be given a “high importance” and “high velocity” goal. Servant regions less than controllers, but enough to start quickly. The final piece of this puzzle is OMVS rules for the controller region because it will launch applyPTF.sh to do maintenance updates. If that OMVS task gets a low priority, that maintenance update can take a long time.

These are all fairly standard System Programmer things. Again, see the white paper for pointers to other documents that provide more detailed descriptions of tuning activities.
Data Connector Usage

Generally speaking, “local” connectors are preferable because of their superior performance characteristics and fewer network delays

From the white paper:

<table>
<thead>
<tr>
<th>Data Connectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>- As a general rule we recommend the use of local connections whenever possible. They result in less network delays and consume fewer CPU resources.</td>
</tr>
<tr>
<td>- Other connector-related recommendations:</td>
</tr>
<tr>
<td>- DB2: use Type 2 rather than Type 4 connectors</td>
</tr>
<tr>
<td>- Use SQLJ rather than dynamic SQL, but that’s really more of an application design recommendation than a connector issue.</td>
</tr>
<tr>
<td>- CICS: use CICS Transaction Gateway, which uses EXCI</td>
</tr>
<tr>
<td>- IMS: in order of preference -- local, MSC, remote IMS Connect</td>
</tr>
<tr>
<td>- MQ: bindings mode over client mode</td>
</tr>
</tbody>
</table>

Note: These recommendations came from the Redbook SG24-6365, “WebSphere for z/OS Connectivity Architectural Choices” at ibm.com/redbooks.

These are general recommendations. Other considerations may override these.
(Such as using JDBC Type 4 -- all Java -- to utilize zAAP engines)

We’ll take a quick detour into an element of the runtime configuration. WebSphere uses “connectors” to access backend data systems such as DB2, CICS and IMS. There are two kinds of connectors -- “remote” connectors intended to put the request on a network in a distributed environment; and “local” connectors that make use of native code to access the data systems cross-memory.

As you’d expect, the better performer turns out to be local connectors. When WebSphere for z/OS and the data system are on the same LPAR, we recommend the use of local connectors whenever possible. The chart shows what is in the white paper in terms of which is recommended for which data system.

These are general recommendations. We are not suggesting these are hard-and-fast rules that must be abided by. There are other considerations. For instance, JDBC Type 4 connectors, which are all-Java, will exploit zAAP engines more fully than will Type 2. They won’t perform quite as well. But you may determine you want as much offloaded to the zAAP as possible. In that case you may choose to run with Type 4. Just an example.
Java Virtual Machine Tuning

Good results can be achieved with a properly tuned JVM. Much depends on the application and the heap size settings of the server.

You can achieve fairly significant performance gains by properly tuning your JVM so the garbage collection cycles are optimized.

This chart gives a rather high-level view of what garbage collection involves. The “JVM Heap” is memory allocated for the purpose of running the Java programs inside that copy of the JVM. (Remember, each application server servant region has its own JVM.) As your application runs, it loads in more and more Java classes, which uses up more and more memory. The JVM heap starts at the defined minimum and grows up to potentially the maximum amount defined to the JVM.

Periodically the JVM will go through and “mark for deletion” objects in memory it sees as not accessed recently. Once marked, the JVM will go through and delete from memory those objects that have been marked. Then follows a compaction where the amount of allocated memory is decreased as the objects in memory are more efficiently arranged in the allocated heap.

This process of mark-sweep-compact is what garbage collection is all about. And the frequency and duration of GC is determined by the minimum and maximum heap size settings, as well as the activity within the JVM. It is very much dependent on the application and its behavior.

The objective of tuning is to get the GC to a point where the frequency is no more than about once every 10 seconds; and the duration of that GC is no longer than about 1 to 2 seconds. Overall, the time spent on GC should be no more than about 2%.

Setting a really high maximum means that GC might not happen all that frequently, but when it does it takes a long time because of all the memory to be processed. A small heap will mean fast GC, but lots of them … the extreme being that the JVM goes into GC pretty much all the time.

There’s no magic formula here … it’s all based on analyzing the behavior of the GC cycles within the JVM and tweaking the settings to achieve the objectives.
IBM SDK 5 and GC Policy Settings

IBM has extended the GC behavior of its SDK 5 with four settings that can be used to further fine-tune your JVM:

From the white paper:

- **Optimize for Throughput** – flat heap collector focused on maximum throughput
  
  "I want my application to run to completion as quickly as possible."
  
  `-Xgcpolicy:optthrput (default)`

- **Optimize for Pause Time** – flat heap collector w/ concurrent mark & sweep to minimize GC pause time
  
  "My application requires good response time to unpredictable events."
  
  `-Xgcpolicy:optavgpause`

- Generational Concurrent – divides heap into “nursery” & “tenured” segments - fast collection for short lived objects. Max. throughput w/ minimal pause time
  
  "My application has a high allocation and death rate."
  
  `-Xgcpolicy:gencon`

- Subpool – flat heap technique to increase performance on MP systems, (> 8) available on IBM pSeries™ and zSeries™
  
  "My application is running on big iron & high allocation rates on many threads."
  
  `-Xgcpolicy:subpool`

Again ... much depends on the application. Main point here is that you have tools available to fine tune the JVM to match the needs of your application.

In addition to setting the MIN and MAX values, with IBM Java SDK 5, which is what comes packaged inside of WebSphere V6.1, you’ll find that four “policy” settings that can further refine the GC behavior. Again, this depends on what the application behavior is and what your objectives are. You can read more about this in the InfoCenter, which we point to on the next chart.
Java Diagnostics Guide 5.0 (for WebSphere for z/OS V6.1)

The InfoCenter has a very good reference for information on the IBM SDK for Java 5, which is what’s packaged inside of WebSphere for z/OS V6.1:

http://publib.boulder.ibm.com/infocenter/javasdk/v5r0/index.jsp

If you’re interested in the topic of garbage collection and JVM tuning, this is worth investing time in and reading carefully.

The definitive place for information on the SDK 5 can be found at the URL shown on this chart. The information is provided in a tree format. If you expand it and seek “How to do heap sizing” you can get a good understanding of the process you go through to do JVM heap tuning.
Application Readiness

Making sure the application has no platform-specific coding and is written in an efficient manner
Two Topics

This section consists of two topics:

Application Portability

Things that would hinder the movement of an application from one platform to another:

- Character set considerations -- make use of the default character set for the platform rather than specific code page references
- Don’t use drive letters in file references (such as `C:\temp\myfile`)
- Pay attention to case -- doesn’t matter in Windows but does in a UNIX system

That’s all we say on this topic in this presentation

Application Efficiency

Inefficient code may show its flaws when moved to z/OS and subjected to much heavier traffic than before.

These are general considerations to keep in mind when reviewing an application you plan to bring to z/OS

This section has two sub-sections: one is the avoidance of coding practices that artificially tie an application to a given platform. The most common issues here are specific references to character code pages rather than relying on the default character set for the platform; the use of Windows drive letters to point to files (and of course up on z/OS USS there’s no such thing as drive letters, as there isn’t on any UNIX or Linux platform); and finally, mismatches due to letter case. Windows is more tolerant of that; z/OS USS (or any UNIX or Linux) is not. The bottom line here is to avoid these sorts of things so the application is capable of moving easily from one platform to another.

The second sub-section involves application design efficiency. The thing here is to design the code so its as efficient as it can be, so that when it comes to z/OS and is subjected to heavy load it’ll work as expected. We’ll now cover a few items in this topic space.
No Such Thing as “Code Written for WebSphere z/OS”

Remember, “WebSphere is WebSphere” ...

... and good code is good code, regardless of the intended target platform

But, what we often see in the WSC benchmark center when an app is brought in for load testing on WebSphere for z/OS:

- Applications written without a good understanding of the performance expectation
- No rigorous performance testing had been done
- No careful profiling of the application had been done to fine-tune the code

So when the code hits z/OS and is put under heavy load, the code often performs poorly. It’s not the platform but the code that falters.

A key concept is that there really isn’t any such thing as “code written specifically for WebSphere on z/OS.” As we mentioned earlier, “WebSphere is WebSphere,” which means the same industry standard APIs, services and methods are available on all the platforms of the same level. Good, efficient code is good, efficient code … no matter where it runs.

That said, what the Washington Systems Center has seen in some of its benchmarking activities is code brought to the z/OS platform that is not very efficient. Once made more efficient the code ran at very high levels of throughput (in the thousands of trans per second range).

The issue here is that quite often code developed first for the distributed platform is written with an eye on functionality first, without a clear view of what the performance and scaling expectations for the code are. Typically the code has not been subjected to any good performance testing. And if application profiling has been done, often it’s not been very extensive. So when this code comes to z/OS and deployed, it runs perfectly well at low levels of activity, but then cracks and breaks at higher levels. That’s the inefficiencies showing through.
Four Fundamental Things

They may seem obvious, but they bear repeating to remind you of them:

Write smart (efficient) server-side componentry

Server-side code that’s expected to run thousands of transactions a minute or second can’t be doing lots of unnecessary things.

Cache interim things

The creation of things implies overhead. The first time it’s created can’t be avoided. But if you know something will be re-used, then cache it.

Carefully engineer portions of code you know will be heavily used

Pay particular attention to areas of the code you know will be heavily used. Invest in the producing the tightest code you can in those areas.

Take advantage of application profiling tools

This helps you understand where applications are particular resource-intensive. Focus should be given to those areas to reduce the overhead where possible.

Check out these “Top Java EE Best Practices”


These things should be done for any code, not just code that will run on WebSphere for z/OS.

We’ll offer four basic, fundamental things to keep in mind:

- The key focus is to write smart code that’s intended to run up on the server. Any code that’s intended to run thousands of transactions a minute or second has to be tight, efficient, and not doing anything unnecessary. Even really fast clock speed CPUs can be overwhelmed doing vast amounts of unnecessary work. This is a pretty broad statement … and it involves lots of coding “best practices” to achieve this.

- Anything your code creates implies initial overhead to perform the object creation. That’s unavoidable. But what is avoidable is incurring that overhead time and time again. So where possible, cache interim things and re-use them from cache.

- Pay particular attention to portions of the code that will be heavily used. Common utility functions used by many users, over and over again should be the very tightest code possible.

- Take advantage of profiling tools to drill into the application itself and find out where resource is being used. Use that information to focus your attention on key areas of the application.

The URL offered in the chart provides a nice listing of some key Java best practices. Follow those, and keep in mind the four points offered here, and you’ll be well down the road to having application code that will run well under heavy loads.

Again, this is not a z/OS statement. This is true for any code run on any platform. The difference is that z/OS is commonly used for heavy usage scenarios. That’s why the code has to be particularly efficient.
We’re going to draw your attention to one aspect of application architecture you may or many not be aware of. It has to do with the invocation of EJBs, and what happens when that’s done across servers on different classloaders. The picture at the top illustrates a fairly common structure: one EJB calls another, which in turn calls two others.

A common design approach in the distributed world is to use remote interfaces on the EJBs and to host the EJBs in separate servers. That works, but what happens is the method parameters that are passed during invocation need to be serialized. Depending on the amount of parameter data to be serialized, that can involve a fair amount of processing and network delay. Under lower levels of volume this may work well enough, but under increasing load the work required to serialize and transfer can become an inhibitor to high volume operations.

A more efficient solution -- if you can affect the design of the application to accomplish this -- is to implement local interfaces and to package the EJBs in the same EAR and host on the same server. This allows objects to be “passed by reference” which does not involve the overhead of serialization or the delays of transferring on the network. The packaging in the same EAR file means all the EJBs are on the same classloader, which enables this pass by reference.

Let’s say you can’t implement local interfaces. Then -- again, if possible -- try to package the EJBs in the same EAR file (so they’re all on the same classloader). Deploy that into the same server. And then set the server’s “ORB Service” (see white paper for how to do this) to “pass by reference.” This will circumvent expensive method parameter serialization.

Finally, if the application is deployed into a cluster, then set “prefer local” at deployment time. That tells WebSphere to avoid crossing over to the other cluster member when invoking EJBs. By “prefer local” you’re telling WebSphere to drive local EJBs first, avoiding potentially costly cross-server calls.
Concluding Points

- Verify that z/OS system had adequate resources to run WebSphere for z/OS and that basic system tuning has taken place
  Some of this can be done prior to the application being brought to the platform

- If possible, make certain WebSphere for z/OS is at the same version and release level
  Eliminates questions of J2EE function deprecation or removal

- Make certain all the customization settings present on distributed have been replicated on the WebSphere z/OS platform
  Connectors, variables, CLASSPATH or LIBPATH updates, utility JAR files, other utility EJBS, etc.

- Allow time in deployment schedule to do necessary verification testing prior to going into production
  Allows time to correct missing customization definitions

- Where possible, take advantage of close proximity to data
  Locate WebSphere and data in same LPAR; use local connectors

- If application will handle high volumes in production, prepare for that with proper JVM tuning and review of application architecture and design
  Have an idea about what performance you need; take advantage of profile tools; allow time to perform stress validation testing before launching into production

We finish with six concluding points:

- Make sure WebSphere is installed, installed correctly, actually runs and has adequate resources. Much of this can be done before the application is moved to z/OS. This is “up front” work.

- Wherever possible, make the levels of WebSphere the same. This will help insure functional alignment, which will avoid issues with deprecation and function removal.

- Replicate on the WebSphere z/OS runtime the customized settings you made on the distributed platform that allowed the application to run. If not an exact one-for-one mapping of customized changes, then make certain what the application needs is configured.

- Don’t make the first invocation of the application on z/OS be the final drop-dead date. Provide time for proper validation and functional testing.

- If you can locate the application in the same MVS system as the data, then all the better. Make use of local (“native”) connectors where possible.

- Have some sense for what kind of performance you’re looking to achieve and test for that ahead of time. If you don’t have the facilities to benchmark the application beforehand, at least review the application and seek to understand possible code inefficiencies that might limit scaling of the application. Tune your JVM according to the information provided in the InfoCenter. And monitor the resource usage and response time of the application as workload grows on the z/OS platform.
Reference Material Cited in the White Paper

Redbook SG24-6369 - WebSphere Application Server V6 Migration Guide

Techdoc PRS2494 - "Performance Engineering & Tuning for WebSphere Version 6.1 on z/OS"

PRS1331 at ibm.com/support/techdocs -- an Excel planning spreadsheet
WP100999 at ibm.com/support/techdocs -- a step-by-step guide using zPMT and spreadsheet
WP100653 at ibm.com/support/techdocs -- WebSphere z/OS V6 sample configuration reference

Deprecated and removed list

Specification and API documentation

Here are some of the documents referenced in the white paper.
Reference Material Cited in the White Paper

"WebSphere for z/OS V6 Connectivity Handbook"
http://www.redbooks.ibm.com/abstracts/sg247064.html

"WebSphere for z/OS Connectivity Architectural Choices"
http://www.redbooks.ibm.com/abstracts/sg246365.html

Techdoc: "Avoiding the Potholes on the WebSphere Application Server On Ramp"
http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS2853

SDK 1.4.2 and SDK 5
http://www.ibm.com/support/docview.wss?rs=3182&uid=swg27007948

SG24-6365, "WebSphere for z/OS Connectivity Architectural Choices" at ibm.com/redbooks.

Java Best Practices
http://www.ibm.com/developerworks/websphere/techjournal
/0701_botzum/0701_botzum.html#sec17

InfoCenter on Server Tuning

And more references.

End of Presentation