WebSphere Liberty z/OS
Operations, Monitoring, Problem Determination
(Some sections still under construction)
This unit is a collection of information cover three topics: operations, monitoring, and problem determination. Those three topics go hand-in-hand. Each has its own set of discussion points and good practices.

**Note:** we intend to release this unit prior to it being fully constructed. We'll add content as we develop it, and as we collect experiences from our own testing and the work of others.
Operations
A review of good practices related to operating your Liberty z/OS servers
We have not yet constructed the content for this section of the unit. Please check back later.
Monitoring
A review of monitoring your Liberty z/OS servers
Monitoring is a topic that has many different elements to it. When a Liberty z/OS server is operating, there are many things going on, so the question becomes: "What am I going to monitor?"

This chart is organizing the monitoring activities into four groups:

1. **z/OS Address Space** -- A Liberty z/OS started task runs as an address space, and because of that you can monitor using existing z/OS tools such as SMF and RMF. This will give you insight into z/OS activity such as CPU and virtual storage (which is important from a z/OS management perspective), but it does not really give much insight into the details of the Java environment or the application running inside there.

2. **Java Virtual Machine** -- the JVM is the heart of the Java runtime environment, and there's a number of metrics that indicate the relative health of the JVM: the behavior of the heap, and the frequency and duration of garbage collection, Java threads, locks, classes loaded, etc. There are quite a few tools you can use in this space.

3. **Application Activity** -- this is a next layer in: monitoring the work requests that enter the application, and what happens for each of those requests. Here we're going to look at the SMF 120.11 v2 record for Liberty z/OS, which provides request activity recording for HTTP requests that are processed by Liberty z/OS.

4. **Availability / Performance** -- this is a broader monitoring to see how well the server and application is servicing client requests. This is where response time and throughput comes into the discussion, and that's often compared against a defined service level agreement that defines the contracted expectations.

Our focus in this section is around points 1 through 3. The availability / performance topic is fairly expansive and for now we'll rule that out of scope for this unit.
The SMF 30 record is the one most commonly used to collect and analyze information about the address space. You can determine CPU usage and some I/O statistics. It can be used against an address space that runs Liberty z/OS, but it does not provide much information about the specifics of activity in the address space. Useful for monitoring at the address space level, but not granular enough to drill into what the JVM or application is doing.

You can use RMF (and the SMF 72 record RMF uses) to get information on specific work being performed in the Liberty z/OS server by enabling the WLM support and classifying work. When the work is classified, it can be assigned to a WLM report class, and from there you can generate RMF reports that tell the story of that work. (Service classification is also possible. That is more of a priority and performance discussion, rather than a monitoring activity discussion.)

This function is based on the concept of identifying work based on the URI pattern, and then assigning transaction class names to that identified work. This is done with definitions in the server.xml configuration file.

Note: WAS traditional made use of a separate "classification XML file" to do the identification of URIs and assignment of transaction class names, but with Liberty z/OS that definition was moved into the server configuration itself. With Liberty’s "include" processing you can have those classification definitions in a separate file if you'd like and merge them in with <include> statements.

You may define how granular you want these definitions to be. You can use a very expansive wildcard definition to assign all work to a transaction class name. If you do that you can monitor using RMF and all work is represented as a single logical group. Or you can define multiple URI patterns and capture work requests more granularly, with each being assigned a different TC name.

The WLM support for Liberty z/OS is enabled with the zosWLM-1.0 feature. That's a z/OS authorized service, and thus the Angel Process is required. The SERVER profile to enable WLM access is the ZOSWLM profile as shown in the chart. With that in place (reminder: check the messages.log file to make sure the z/OS WLM feature "is available"), you can classify the work, create WLM transaction classes, then use RMF to produce reports on the work requests.
One of the most basic JVM monitoring techniques is turning on "verbose garbage collection" (verbose:gc) and analyzing the output. This can be done on a JVM-by-JVM basis. Enabling verbose GC is done with a simple JVM property, and for Liberty z/OS that would be the jvm.options file, which resides in the same directory as the server.xml file.

**Note:** the overhead to the server to write verbose:gc output is fairly low. It is considered a general good practice to enable verbose:gc for servers that are important to keep track of. The output can be sizeable, so some consideration about managing the size of that output needs to be made.

There is no difference in verbose:gc analysis for Java z/OS than Java on any other platform. It's the same format.

On z/OS the output for verbose:gc can go to JES spool or to a file location. By default it goes to STDERR, and the sample JCL proc for Liberty z/OS has STDERR directed to SYSOUT=*.

A handy function of JES output for verbose:gc is that you can have JES spin the spool closed after some number of lines. An example of that is shown on the chart.

Once the output is created, you can analyze using any tool capable of understanding the format of the verbose:gc output. The IBM Support Assistant is one such tool. It is a no-charge tool that can be downloaded from the link provided on the chart. The IBM Support Assistant will analyze the output, report on its findings, and provide recommendations for further tuning.
Note: the purpose of this chart is to provide a high-level framework of monitoring. In particular we wanted to establish the idea of an agent that collects the data. That’s key, and there are different options depending on what tool you select. Depending on what experience the audience has with monitoring tools in the past, their view of "monitoring agents" may be negative (some agents in the past were heavyweight and problematic). The JVM monitoring agents that are available today are much lighter weight and trouble-free than were agents from five or ten years ago.

To collect real-time data on the JVM, some kind of agent is needed in the JVM to collect the information and provide it to a system that asks. The options here range from the default monitoring agent that comes with the IBM SDK (the "IBM Health Center"), up to more sophisticated agents that collect even more information (such as the agent that works with the OMEGAMON for JVMs product, which we'll discuss in a bit).

The type of data that's collected is a function of the agent that's supplying the data. The supplied IBM Health Center API provides a reasonable set of statistics on heap, thread usage, garbage collection, and locking. Other agents supply even more information.

Finally, there is typically some kind of viewing client that takes the information and displays it in a readable format. That can be a plugin to Eclipse, or it can be a 3270 client in the case of OMEGAMON for JVMs.
The IBM Java SDK comes with a monitoring agent that can supply a great deal of information about the JVM and applications running in the JVM. This agent is configured with JVM options that define the host and port clients can use to access the agent and extract the information being collected.

**Note:** if you read the Knowledge Center article at the link on the chart, it speaks of installing the agent. What that is saying is that depending on the level of the SDK you have, you may be able to download and install a more current version of the agent code that will capture more information. But you don't have to do that: you can make use of the IBM Health Center agent that comes with your level of the SDK. If you want to view the information using the Health Center client, you'll need to configure the port the agent will open for clients to use.

The Health Center client is an Eclipse plugin that can install into any Eclipse platform, or you can get the IBM Support Assistant which has the Health Center client already installed. The data you can view with the client is a function of the level of the SDK and agent, but what's shown on the chart is a good summary of the types of things it collects.

The key point here is this: there is a low-impact agent that's part of the IBM Java SDK for z/OS. This agent, and the client code that can access it, can be very useful in seeing what's going on in the JVMs of your Liberty z/OS servers.

We're next going to look at a licensed product that takes advantage of this IBM Health Center API for monitoring of all the JVMs on an LPAR.
IBM OMEGAMON for JVM on z/OS is a licensed software product designed to facilitate the monitoring of all JVMs that are running on your z/OS system. It’s a more comprehensive solution than a Liberty-specific tool. It is capable of monitoring Liberty z/OS JVMs, and it is capable of monitoring other JVMs on z/OS well.

OMEGAMON for JVM is capable of discovering JVMs that are running on your z/OS environment. This is of value because you may have JVMs operating you do not realize you have, either because they’re embedded in other products (such as CICS), or because other groups have started the JVMs without your knowledge. The discovery process is fairly clever -- the OMEGAMON for JVM Agent (a separate STC that runs on each LPAR you wish to monitor) has the ability to scan z/OS control blocks looking for instances of JVMs that have been started. By doing this, it can discern the nature of the JVM, the job name, the ASID, and other details of the JVM.

We mentioned the IBM Health Center API on the earlier chart and said it came with the IBM Java SDK for z/OS. That means by default, any z/OS Java JVM that is operating is supplying at least that level of data metrics. (Assuming, of course, the Java SDK in use by the JVM is reasonably current. Some Java installation from 10 years ago would not provide this API; the more current ones do.) From that API comes information on GC statistics, thread details, and locking details.

You can augment the information you can monitor by installing in a JVM the OMEGAMON for JVM agent, which provides more information, such as CPU, GP, zIIP offload, and zIIP spillover to GP.

The information is collected by the OMEGAMON for JVM Agent task on the LPAR, which maintains current and historical data. Then you can use a data viewing client -- either Eclipse-based or 3270-based -- to view the information and drill down on finer details.

This is a very powerful tool. It is a separately licensed product and it carries a license fee. If you believe it provides value to your business, you can explore this option more fully. The link is:

The Liberty AdminCenter function -- enabled with the AdminCenter-1.0 feature and the appropriate security for the AdminCenter in place (summary: accessible user registry and ID granted access to the "Administrator" role) -- has a tool that provides for graphical monitoring of the JVM. This function is available to all platforms supported by Liberty, not just z/OS.

**Note:** if you have a Liberty collective configured, the AdminCenter in the controller can view and show the JVM monitoring information for members in the collective, whether on the same LPAR, another LPAR, or another platform.

The data you can review is based on the agent that's in effect. By default it'll have access to the "IBM Health Center" API that's part of the IBM Java SDK, and with that you see information on heap, loaded classes, active JVM threads, and CPU usage. If you enable the Liberty monitor-1.0 feature, then you get more information related to the applications that are running: active sessions, active threads, average response time, average wait time, etc.

This is part of Liberty (any platform) and serves as a useful monitoring tool. It does not maintain historical data, however: it shows the data for the past minute or so, but it does not maintain a history of that data. So it's a real time monitor with a brief view of history.
On the previous chart we mentioned the monitor-1.0 feature, and we wrote about how it could be used with the monitoring function provided by the adminCenter-1.0 feature.

What the monitor-1.0 feature provides is a set of management beans (mBeans) that provide information about the JVM and applications within the JVM. The adminCenter-1.0 feature simply invokes these mBeans to get the information it displays on the browser screen.

Those mBeans are accessible to any management client that can access the mBeans, including a custom JMX client that you create. That means you could maintain historical data if your client were to periodically collect statistics and store them away.

The chart has a URL that takes you to the Knowledge Center page for those mBeans. That will provide access to the details of the mBean implementation and the data fields that are provided.
Moving away from the monitoring of the JVM, we now turn to the SMF 120.11 version 2 record format that came available with Liberty z/OS 16.0.0.2. Right now this record is for HTTP traffic only, so application requests that come in on other mechanisms -- say, JMS message queueing -- would not cut this record. The purpose of this record is to record activity for each request. (It is like the WAS traditional SMF 120.9 record in that it’s a request record, but the Liberty z/OS 120.11 v2 records is HTTP only and has different data information captured.)

The SMF function is enabled with the zosRequestLogging-1.9 feature. This is not a z/OS authorized service, so the Angel process is not required for this. However, writing SMF records is controlled with the BPX.SMF FACILITY class profile, so for a server to write the record it will need to have READ to that profile.

The output is written to SMF buffers, then out to the SMF data sets. This is no different than SMF recording for any other system on z/OS.

The SMF 120.11 v2 record is new enough that some SMF analysis tools may not yet have awareness of the format. Some do, but some may not. Please check with your vendor to see if the tool you use does. The Techdoc listed on the chart provides information about the record format.
Problem Determination
A review of performing effective problem determination of Liberty z/OS servers
We are still in the process of designing and constructing the content for this section.

We have not yet constructed the content for this section of the unit. Please check back later.