Global Knowledge®

Expert Reference Series of White Papers

Three Key Strategies for Preventing Problems with IBM WebSphere Business Process Manager V8.5

Three Key Strategies for Preventing Problems with IBM WebSphere Business Process Manager V8.5

IBM

Introduction

A troubleshooter's job doesn't start after a problem occurs, preparations should be made well in advance of a problem. Troubleshooters often see a problem (for example, the system crashes) and start conducting many long and complex analyses that can take days or even weeks. However, prudent system administrators or troubleshooters start planning long before a problem occurs. In other words, they prepare the environment so that troubleshooting can be done more quickly and effectively if and when problems occur. The following key strategies will help prevent problems from occurring with IBM WebSphere Business Process Manager V8.5.

Three Strategies for Problem Prevention

#1: Tune and Monitor the Environment Regularly

Monitoring tools and a plan are needed to effectively detect problems or anomalies when they emerge. Monitoring is a trade-off. You want to detect important events, and yet not adversely impact the normal operation of the system. Monitoring is an entire technical area in itself, different from problem determination. This paper covers only a few points on this topic.

Passive monitoring can be done at all levels: network, operating system, application server, and application. The main system log files of dependent systems such as databases and LDAP directories can also be monitored for errors and events. For example, you might detect application server restarts that indicate the server is failing. Some tools for passive monitoring include the Tivoli Performance Viewer and IBM Tivoli Composite Application Manager for Application Diagnostics.

Active monitoring goes beyond passive monitoring—you periodically test the operation of the entire system from end to end. One technique is to ping system components, such as one server or one database connection. Another technique is end-to-end pinging: you periodically send an entire "dummy" transaction through the system and verify that it completes. Some tools for active monitoring include IBM Tivoli Composite Application Manager for Transactions and web-based, load-generating programs like Rational Performance Tester.

Make sure that you:

- Monitor the system at all levels, from the network to applications and dependent systems
- Monitor the main system log files for errors and events such as detecting application server restarts
- Use monitors and alerts on the following key system metrics:
    - Memory usage
    - Default PMI statistics
    - Performance advisors

Examples of ongoing system "health" monitoring include:

- Significant errors in the logs that the various components emit.
- Metrics that each component produces should remain within acceptable norms. For example, operating system processor and memory statistics, IBM Business Process Manager Performance metrics, and transaction rates through the application.
- Spontaneous appearance of special artifacts that get generated only when a problem occurs, such as Java dumps or heap dumps.
- Periodically send a "ping" through various system components or the application, and verify that it continues to respond as expected.

Be prepared to actively generate more diagnostic tests when a problem occurs. In addition to dealing with diagnostic artifacts that are present when an incident occurs, your troubleshooting plan should consider any additional explicit actions to take as soon as an incident is detected. You want these actions to take place before the data disappears or the system is restarted.

Here are some examples of explicit actions to generate more diagnostics:

- Actively trigger various system dumps, if they are not generated automatically (such as Java dump, heap dump, system dump, or other dumps that various products and applications might provide). For example, when a system is believed to be "hung," it is common practice to collect three consecutive Java dumps for each potentially affected JVM process.
- Take a snapshot of key operating system metrics, such as process states, sizes, or processor usage.
- Enable and collect information from the IBM Business Process Manager Performance Monitoring Infrastructure instrumentation.
- Dynamically enable a specific trace, and collect that trace for a specified interval while the system is in the current "unhealthy" state.
- Actively test or "ping" various aspects of the system to see how their behavior changes compared to normal conditions. This activity is done to try to isolate the source of the problem in a multicomponent system.

#2: Keep Good System Documentation

When investigating a problem, you need information about the system, and especially about how the system works when there is no problem. This information should be collected in advance. You should formalize this information by:

- Creating and maintaining a system architecture topology or flow diagram
- Establishing a series of baselines for the normal behavior of the system
- Creating and maintaining change logs

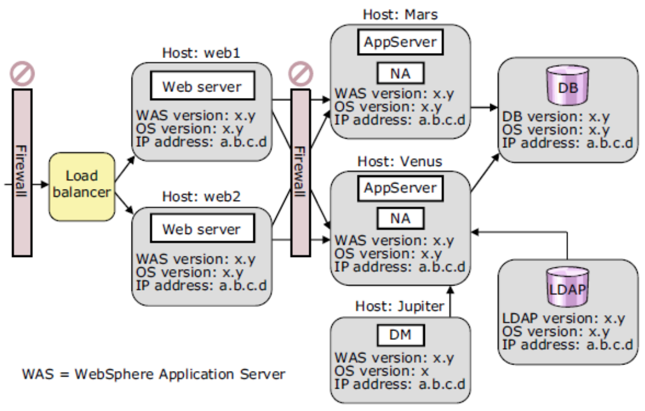
First, create a system architecture or topology diagram that shows all system components and the main flows between them. The system architecture diagram can help move the progress of the troubleshooting process by providing information for the following activities:

- Communicating with all parties involved in the problem determination effort.
- Identifying discrepancies between the expected environment and the current reality.
- Identifying points where monitoring or health checking can be done.
- Identifying points where diagnostics data can be collected.

When creating an architecture diagram make sure that you:

- Show all major components of the overall system including:
  - Machines and software components that operate on these machines
  - How they communicate
  - Main flows for requests that are processed through the system

- Are specific and detailed
  - Indicate software versions, server names, and IP addresses, if possible



Second, establish a system baseline and gather extensive information about the state of the system at a time when the system is operating normally.

Some questions to answer when creating system baselines:

- What does a "normal" system look like?
- What are the typical values for the following common system metrics?

  - Processor usage
  - Memory size
  - PMI statistics

- What is a typical load?
- What is a typical response time for various operations?
- What should you normally see in the logs during operation of the system?

Here are some examples of information that you might collect:

- Copies of the various log files and trace files over a representative time period in the normal operation of the system, such as a full day.
- Copies of a few Java dumps, heap dumps, system dumps, or other types of artifacts that are normally generated after a problem occurs.
- Information about normal transaction rates in the system, response times, and other time-dependent information.
- Various operating system statistics on a correctly running system, such as processor usage, memory usage, and network traffic.
- Copies of any other artifacts, information, or normal expected results from any of the earlier suggested special diagnostic collection actions, for each anticipated type of problem.

In many actual systems, it is not unusual to see various benign “errors” during normal operation. Learn to recognize these benign errors, or better yet, eliminate as many of them from the implementation of the system as possible.

Lastly, keeping a rigorous log of all changes that are applied to the system over time can help you determine system differences. When a problem occurs, you can look back through the log for any recent changes that possibly contributed to the problem. You can also map these changes to the various baselines that were collected in the past to ascertain how to interpret differences in these baselines.

Your change log should at least track all software upgrades and fixes that are applied in every software component in the system, including both infrastructure products and application code. It should track every configuration change in any component. It should also track any known changes in the pattern of usage of the system, such as expected increases in load, or a different mix of operations that users invoke.

In a complex IT environment where many teams contribute to different parts of the environment, the task of maintaining an accurate, up-to-date, and global change log can be surprisingly difficult.

You can use tools and techniques to help this task, from collecting regular snapshots of the configuration with simple data collection scripts, which are used to collect diagnostic data, to sophisticated system management utilities.

It is important to know and understand the concept of change control, and keeping a change log is generally broader than the troubleshooting arena. Change control is also considered one of the key good practices for managing complex systems to prevent problems, as opposed to troubleshooting them.

#3: Create and Maintain the Following Plans:

Diagnostic Data Collection Plan

When a problem occurs, there is often confusion along with great pressure to restore the system to normal operation quickly, which can cause mistakes that lead to unnecessary delays. The following steps are critical: make sure that you have an action plan, and make sure that everyone is aware of this action plan. As with monitoring, gathering data is a trade-off. You want to capture as much data as possible, but you do not want to adversely affect the normal operation of the system. You also want to make sure that you can reliably capture the diagnostic data that you need ask yourself:

- Can you find where the data is stored?
- Do you have procedures established for collecting the data?
- Do you have enough hard disk space to store the data?

The simplest diagnostic collection plan is in the form of plain, written documentation that lists all the detailed manual steps that must be taken.

To be more effective, try to automate as much of this plan as possible. Provide one or more command scripts that can be invoked to do a complex set of actions, or use more sophisticated system management tools. The various collector tools and scripts now offered as part of IBM Support Assistant can provide a good framework for you to start automating many diagnostic collection tasks.

Also, do not forget the human element:

- When there is a production problem, who is in charge of implementing the diagnostic data collection plan and taking recovery actions?
- What groups must be informed or coordinated with?

The following are some tools to consider when you plan for and implement diagnostic data collection.

- Check the default WebSphere logs: The default WebSphere server logs include SystemOut, SystemErr, native_stdout, and native_stderr.
- WebSphere first-failure data capture (FFDC) facility: FFDC provides the instrumentation for exception handlers (catch blocks) to record exceptions that a component produces. FFDC data can be found in the SystemOut.log file and in the FFDC folder in the server profile logs directory.
- HTTP access logs: Increased logging of requests at the HTTP server to show not just a single log entry for each request, but a separate log entry for the start and end of each request.
- JVM verboseGC log: This log is often useful, and usually uses relatively low processing power on a well-tuned system.
- Core memory dumps, JVM Java dumps, heap dumps, and system dumps: Java dumps are typically cheap to produce, and can be enabled for automatic generation. Heap dumps and system dumps can involve significant processor usage, so consider carefully before setting them up to be triggered automatically.
- Enable verbose garbage collection: Verbose garbage collection has a minimal impact on performance; the benefit usually outweighs the cost.
- Minimal monitoring of network health: For example, a moderate level of monitoring performance counters that the WebSphere Application Server Performance Monitoring Infrastructure (PMI) provides.
- Minimal monitoring of operating system resources: For example, minimal WebSphere Application Server tracing to capture one or a few entries only for each transaction (web requests or EJB requests).
- Attempt pings and application checks.
- Application-level tracing and logging, if any.
- Collect specific PMI statistics. When PMI is enabled, the monitoring of individual components can be enabled or disabled dynamically.
- Use IBM Java Health Center to monitor an active JVM: IBM Java Health Center does not use much processing power, and it runs alongside an IBM Java application with a small effect on application performance. It can be downloaded from the following website: https://www.ibm.com/developerworks/java/jdk/tools/healthcenter/

Relief or Recovery Plan

After a problem occurs, consider how much time is available to look into the problem before you must provide relief to affected users. It is a good practice to create a relief or recovery plan to help you restore function to users. The relief or recovery plan lays out general steps that you take to restore functions, and actions to take for specific problems.

For the relief or recovery plan:

- Try to predict the most common types of problems, which are based on knowledge of the system topology and flows, such as:
    - Loss of an application server
    - Loss of database connectivity
    - Loss of the LDAP server

- Identify different regions of the system that can be isolated or restarted independently, including:
    - Applications (Java EE, BLA, OSGi)
    - Application servers (JVM)
    - Application server clusters
    - Servers
- Document the following processes:
    - Who decides what to do and who does it?
    - Criteria for making such decisions.
- Practice the most common relief actions in advance and:
    - Ensure that you know how to do them.
    - Ensure that they have no unexpected side effects.

Maintenance Plans (Scheduled and Emergency)

Applying regular maintenance (interim fixes, fix packs) reduces the probability and impact of problems. In addition to regular scheduled maintenance, you also must perform emergency changes or maintenance to the system, in response to a newly diagnosed problem. The emergency maintenance plan outlines how to do so safely and effectively.

Maintenance should occur at all levels: on the operation systems level, on the application server, and on each of the products that are involved in the system. You can track current maintenance levels in the topology diagram. You should also have processes for verifying the maintenance levels regularly.

Lastly, consider upgrading to the latest available fix pack during an investigation. Individual fixes are meant to be temporary until a fix pack is available. There is considerable risk in using too many individual fixes because it is not possible to test all the possible interactions among individual fixes. Because of the complexity of the system and difficulty of reproducing problems and gathering diagnostic information, it is not always practical to determine exactly which fix (APAR) resolved a particular situation. Use Fix Central to download fixes: http://www.ibm.com/support/fixcentral

Additional Considerations for Problem Prevention

Apply Good Practices for Module Design and Connectivity Groups

It is a good practice to create a “connectivity group” to represent the possible request sources for the system. A connectivity group is a specific pattern of behavior that is found in an SCA module. The connectivity group does not contain useful component types such as long-running business processes and business state machines. These connectivity groups provide encapsulation and isolation of the specific endpoint’s integration requirements. WebSphere Enterprise Service Bus mediation modules are commonly used for this purpose as they represent convenient ways to implement infrastructure-related tasks.

The concept of connectivity groups also provides a convenient way to acquiesce the system in case there is a need for recovery. Since a connectivity group module is stateless, the module can be temporarily stopped, thus cutting off the inbound flow of new events. After the system is recovered and able to process new work, these modules can be restarted.

Thoroughly Plan Application Design, Exception Types, and Fault Processing

Good application design takes advantage of the error-handling and fault-processing capabilities from IBM Business Process Manager and WebSphere Enterprise Service Bus.

It is, therefore, necessary for solution architects to understand how IBM Business Process Manager and WebSphere Enterprise Service Bus represent declared and undeclared exceptions before they can create a comprehensive error-handling strategy.

The SCA programming model provides two types of exceptions:

- Service business exceptions, which are defined on the service interface
- Service runtime exceptions, which are undeclared (an unexpected condition in the runtime)
-

Create an Error-Handling Strategy

The architecture team must understand the error-handling and recovery tools and capabilities of the product. This team is responsible for creating the error-handling strategy for the project and must account for the following items:

- Appropriate usage of units of work (transactions and activity sessions)
- Declaration and usage of faults and Service Business Exceptions
- Consistent fault processing for all component types, especially BPEL and mediation flow components
- Usage of retry logic and "continue on error" Business Process Choreographer capabilities
- Appropriate settings for completed process instance deletion
- Correct usage of synchronous and asynchronous invocation patterns
- Appropriate usage of import and export types
- Appropriate usage of the retry capability in mediation flows

In addition to this list, the architecture team must create design patterns in which built-in recovery capabilities, such as the IBM Business Process Manager and failed event manager, are used appropriately.

Test, Test, and Test

The best tool for problem prevention in production is the execution of a comprehensive functional and system test plan. In general, tests for deployed solutions can be broken into two groups:

- Functional tests, which confirm the functionality of the implementation as compared to the business requirements
- System tests, which include cases to verify performance, high availability, and recovery service level agreements

Conclusion

A major challenge of problem determination is dealing with unanticipated problems. It is much like detective work: finding clues, making educated guesses, verifying suspicions, and other considerations. An ideal strategy for problem prevention is to monitor the system regularly. Use the strategies outlined in this paper to minimize downtime and detective work so you can maximize performance.

Learn More

Learn more about how you can improve productivity, enhance efficiency, and sharpen your competitive edge through training.

Administration of IBM Business Process Manager Standard V8.5

Developing Applications in IBM Business Process Manager Advanced V8.5 -I

IBM Business Process Manager V8.5 Problem Determination

Visit www.globalknowledge.com or call 1-800-COURSES (1-800-268-7737) to speak with a Global Knowledge training advisor.

About the Author

The contents within this paper are derived from the “IBM Business Process Manager v8.5 Problem Determination” courseware developed by the WebSphere Education group, part of International Business Machines Corporation.