MQ Transaction Monitoring for an Aviation Supply Chain System

At a Glance

INDUSTRY
Aviation Supply Chain Solutions

LOCATION
United States

USE CASE
The client in aviation supply chain solution wanted to track and audit business transactions. This visibility lets the business not only to measure how well their services achieve, but also improve them end-to-end.

CHALLENGES

Client wanted to have visibility, access and control over critical business transactions to quickly isolate and resolve potential performance from multiple vantage points and environments.

SOLUTION

With the increased focus on compliance, Royal Cyber helped them in many ways to reduce operational risk, in many transactional areas.

About The Client

The client is a neutral purchasing portal for the aviation industry based in Irving, Texas. Founded in July 2000 and commenced operations a couple of months later on October 1, 2000. Their service focuses on the repair, replenishment, sourcing, inventory pooling and other critical operations in the aviation supply chain. It automates the exchange of documents and information for commercial transactions.

Business Challenges

Client wanted a solution to track and audit business transactions. They have vendors who communicate over MQ cluster channels. Hence, they wanted to track the messages that are communicated over MQ channels with those external partners. Also, they wanted to make an audit record for internal applications for producing/consuming messages to/from queues.

Our Approach/Solution

WebSphere MQ has an exit interface to interact with each MQ API call. API exit can be invoked both before API execution and after API execution. In calling API exit programs, all parameters passed by MQ API caller are passed to API exit as they are. All parameters returned by MQ are also passed to API exit after API execution. In addition to API exit, exit interface in sending and receiving message data via MQ channels is also available. Setting up and configuring IBM MA0W SupportPac (API trace utility) provides a mechanism to trace all MQ API calls including parameters and process results in an easy way.

Key Takeaways

✓ **Diagnose and mitigate issue in no time**

✓ **Saving $ by minimizing overtime payments to employees**

✓ **Saving employee's time**

✓ **Audit and log all business transactions**

The customer gained the following features by implementing the solution:

Functional highlights in getting trace data

- All parameter values passed by MQ applications can be traced completely. (API exit)
- All parameter values set by MQ can be traced completely including completion code and reason code. (API exit)
- Time before API call, time after API call and process time of each API calls are shown for each API call. Time unit is in microsecond on UNIX systems. On Windows, time unit is in millisecond and 64-bit performance counter based on CPU clock is shown in addition, so that process time can be identified in detail. (API exit)
- All MQ messages transferred via MQ channels can be traced completely. (channel exit)
- Parameter values are parsed and shown in the format of easy view.
- Total call count for each API is shown as a summary. Success count, warning count and fail count are also shown respectively.
- Process time of each API call is shown in detail output and summary output. Average time, minimum time and maximum time are also shown at the last of trace data. (API exit)
- Multiple byte character set (MBCS) handling is considered for text output.

Supplemental functions in getting trace data

- Trace data can be put to multiple files when data volume becomes large. In default setting, new trace files are created in every 100MB trace data.
- File compression program can be invoked at the creation time of new trace file to reduce disk space usage.
- It is useful for long run test putting large data volume.
- Operating environment, process name, and related process/thread information are shown as a part of context information.
- Target queue name and/or process name to be traced can be specified to eliminate unnecessary data for minimizing output data and also for minimizing performance degradation.
- Alert files can be generated