WebLogic JMS als zentrale Enterprise Messaging Plattform

Thomas Störmer, Dr. Alexander Königs
Schenker AG
Essen

Schlüsselworte

JMS, Oracle WebLogic, Middleware, Governance, Operations, Tuning, Performance, Konfiguration, Monitoring

Einleitung

Eine zentrale Messaging-Lösung zum Datenaustausch zwischen Unternehmensapplikationen hat besondere Herausforderungen, aufgrund der Diversität der Applikationen und deren unterschiedliche Einsatzbereiche. Die Notwendigkeit einer solchen Messaging-Lösung wird oft bedingt durch die Etablierung von Service-orientierten Integrationsarchitekturen (SOA). DB Schenker hat eine zentrale JMS Plattform basierend auf Oracle WebLogic etabliert, welche den direkten Datenaustausch zwischen Applikationen sowie des zentralen Enterprise Service Bus realisiert.

Der Erfahrungsbericht stellt wesentliche Herausforderungen aus sämtlichen Bereichen von Architektur, Governance, Capacity Management bis hin zu Operations vor und zeigt mögliche Lösungswege auf. Dabei werden ebenfalls technische Best Practices für WebLogic JMS vorgestellt, sowie Erfahrungen für notwendiges Load Testing und dem Tuning der Plattform.

Dieser Querschnitt der Themengebiete soll hilfreiche Erkenntnisse und Rückschlüsse für den Aufbau, die Etablierung und den Betrieb von Messaging-Plattformen insbesondere basierend auf Oracle WebLogic bieten.

Enterprise Messaging

Messaging ist zu einem beliebten Kommunikationsmittel zwischen Unternehmensapplikationen geworden. Java hat hierfür den JMS Standard definiert, welcher von vielen verschiedenen Providern implementiert wurde, um eine Messaging Lösung zu realisieren. Im Zuge der Service orientierten Architektur (SOA) mit dem Einsatz entsprechender Integrationsplattformen (ESB) hat auch eine „Zentralisierung“ der Messaging Plattformen im Unternehmen eingesetzt.

Dieser Ansatz hebt folgende Vorteile. Zum einen wird das Messaging Know-How in einem Team gebündelt und kann vielfach von Synergien profitieren, um die Plattform für die Nutzungsanforderungen zu tunen und optimales Hosting/Operations zu realisieren. Hinzu kommt ein zentrales Monitoring und Reporting, welches auch eine Übersicht des Datenverkehrs mit Volumen und zeitlichen Gegebenheiten der Unternehmensapplikationen liefert. Letztendlich wird vermieden, kritische Plattform-Themen stets neu und unterschiedlich gut in verschiedenen Produkten lösen zu müssen.

Ein größerer Nachteil den jedes zentrale System mit sich bringt, ist die extreme Kritikalität der Plattform aufgrund von massiven Störungsauswirkungen. Hinzu kommt die besondere Herausforderung der Bandbreite an zu erfüllenden Plattformanforderungen. Aufgrund der Diversität der Enterprise Applikationen können die Anforderungen teils konträr ausfallen. Eine typische

Anforderung ist die Größe der Messages, die von einigen KB bis hin zu GB reichen kann (abgesehen davon welche Größen wirklich sinnvoll sind). Es gibt Applikationen mit „Echtzeit“ (also zeitnaher, zeitkritischer) getriebener Kommunikation, aber auch Applikationen mit Batch orientierter Verarbeitung, die keiner strengen zeitlichen Einschränkung unterliegt. Es gibt große Enterprise Applikationen, welche klassisch auf Application Servern (e.g. Oracle WebLogic) betrieben werden, aber auch schlanke Java Standalone Applikationen oder Cloud basierten Lösungen. Aus dieser Diversität entstehen sehr verschiedene Anforderungen, die zu einem Anforderungs-Kompromiss resultieren oder ggf. einer strikten Ablehnung bedürfen. Eine weitere Herausforderung ist die Ownership der verbunden Applikationen, welche bei anderen Teams und Abteilungen im Unternehmen oder sogar Tochter- und Fremdunternehmen liegen kann. Das erfordert verschiedene Maßnahmen zur Sicherstellung eines zuverlässigen Betriebs und der Nutzung der Plattform.

Governance

Aufgrund der angeführten stark verteilten Ownership der Applikationen, welche die Messaging Plattform nutzen, sollte eine klare Governance definiert werden. Basierend auf der Service orientierten Architektur, können Applikationen für Schnittstellen die Rolle als Service Provider oder Consumer einnehmen. Es ist sinnvoll die Ownership von Queues dem jeweiligen Service Provider der Schnittstelle anzuhängen. Dieser trägt damit die Verantwortung Anforderungen bezüglich Kapazität und Nutzung an die Messaging Plattform zu kommunizieren, in Störungsfällen zu unterstützen und ggf. über Maßnahmen zu entscheiden (Löschen von Messages, pausieren der Verarbeitung…), sowie der Abschaltung von Schnittstellen zu informieren. Aufgrund der Rolle als Service Provider sind diese Anforderungen meist selbstverständlich und müssen sowieso im eigenen Applikationskontext erfüllt und betreut werden.

Eine weitere Maßnahme ist die Gestaltung eines Prozesses zur „Beantragung von Queues“. Dieser soll sicherstellen, dass die Ownership, Art der Datenkommunikation, Beteiligte Kommunikationspartner und erwartete Kapazitätsanforderungen geklärt sind. Dieser Prozess wird ebenfalls zur nächsten Maßnahme benötigt, um standardisierte Queuenamen zu definieren. Es ist empfehlenswert den Queuenamen mit wichtigen Informationen wie Art der Daten oder Servicename (z.B. CustomerAccountUpdate), Queueowner, Kommunikationsrichtung Inbound/Outbound (z.B. aus Sicht des Owners) und involvierte Applikation zu versehen. Diese Informationen sollten zusätzlich in einem „Repository“ gespeichert werden, damit sie für verschiedene Stakeholder zugänglich sind (z.B. Support).

Die nächsten Maßnahmen sollen die geeignete Nutzung der Plattform sicherstellen. Hierfür werden klare Regeln und Verbote definiert, wie zum Beispiel einer Message-Größen Limitierung auf 10 MB. Für die Unterstützung von unerfahren Projekten, welche meist kleine Java Standalone Applikationen anbinden möchten, empfiehlt sich die Bereitstellung eines „JMS Cookbooks“ mit Best Practices, sowie einer Checkliste von Punkten zur Berücksichtigung bei der Implementierung.

Architektur

Die Messaging Plattform basierend auf Oracle WebLogic JMS sollte als Cluster bestehend aus mehreren WebLogic Servern betrieben werden. Für Queues sollten ausschließlich Uniform Distributed Queues genutzt werden, damit jeder Server im Cluster einheitlich genutzt werden kann. Es bietet sich an für den JNDI Lookup einen Loadbalancer (oder DNS Loadbalancing) zu implementieren. Somit werden die Clients nur die URL für den JNDI Lookup wissen müssen und erhalten die Cluster-Informationen später durch die genutzte ConnectionFactory. Es muss keine umfangreiche Konfiguration auf der Clientseite erfolgen, denn alle Objekte wird der WebLogic Server über JNDI bereitstellen. Als Persistenz für die JMS Server sollte eine redundante Datenbank oder Veritas FileCluster verwendet werden.

Monitoring & Operations

Für den Betrieb der Messaging Plattform ist ein geeignetes Monitoring & Alerting zwingend erforderlich. Auf WebLogic Server Ebene sollten JMS Connections hinsichtlich aktuell offener Verbindungen und Anzahl neu geöffneter Verbindungen kontrolliert werden. Hierüber kann man z.B. Clients identifizieren die JMS Connections nicht sauber schließen.

Für einen allgemeinen Überblick des Zustandes, sollten auf JMS Server Ebene aktuelle Anzahl Messages und Bytes sowie „Pending" und „Received" beobachtet werden. Damit lassen sich problematische Situationen schnell erkennen. Beispielsweise kann die Anzahl aktueller Messages und Bytes auf fehlende JMS Consumer oder verzögerte Verarbeitung der Applikationen hindeuten. Die Anzahl der „Pending Messages" kann auf eine Applikation mit Transaktionsproblemen oder nicht verfügbaren Ressourcen (Datenbank, WebService) hinweisen (in diesem Fall wurden die Messages zurückgerollt und für einen späteren Versuch „delayed" → „RedeliveryDelay").

Für eine unabhängige Bewertung der Verfügbarkeit der Messaging Plattform und der Performance der JMS Operationen (Response Times) sollte ein Health-Check Client implementiert werden, welcher das Senden und Empfangen auf jeden JMS Server in einer dedizierten Testqueue prüft. Damit kann die Verfügbarkeit und Funktionalität der JMS Server sichergestellt werden und darüber hinaus liefern die Messzeiten der einzelnen JMS Operationen wichtige Hinweise bezüglich Performance in Korrelation zu JMS Serverlast. Es wird direkt sichtbar, wenn zu Message-Peaks die Antwortzeiten der Plattform bedrohlich steigen.

Auf Systemebene sollte die CPU-Auslastung kontrolliert werden, welche maßgeblich in Abhängigkeit zum Message-Durchsatz steigt, sowie beispielsweise Disk I/O im Fall von FilePersistence für den Byte-Durchsatz.

Zur Erfassung von Monitoringwerten kann die JMX Schnittstelle genutzt werden. Aufgrund der schlechten Latenzzeiten sollte jedoch eher der Einsatz eines WLDF Moduls mit Harvester oder die Abfrage über „dms/spy" favorisiert werden. Optimal ist die Auswertung der Kennzahlen über neuartige Lösungen wie Elasticsearch in Kombination mit Kibana.

WebLogic JMS Konfiguration

Die Konfiguration neuer Queues sollte automatisiert über WLST Skripte und Deploymentpakete erfolgen. Das macht insbesondere Sinn, wenn der Betrieb der Plattform bei einem Dienstleister oder anderen Abteilungen erfolgt. Bewährt hat sich die Entwicklung eines allgemeinen WLST Skriptes, welches basierend auf bereitgestellten XML-Konfiguration Queues, Connection Factories oder Foreign Server konfiguriert. Somit müssen nur die XML-Konfigurationen erweitert oder angepasst werden und können als Deployment Paket mit WLST Skript übergeben werden. Die Deploymentpakete können nach Ownership strukturiert werden.

ForeignServer und Connection Factories sollten über ein Subdeployment einem Cluster als Target zugewiesen werden. Uniform Distributed Queues dagegen sollten explizit allen JMS Servern im Cluster zugewiesen werden.

Da die WebLogic Client Lib aktuell noch nicht automatisch alle Queue-Instanzen auf allen JMS Servern im Cluster anspricht, haben insbesondere Single-Threaded Standalone-Clients Probleme alle Messages aller JMS Server zu empfangen. In diesem Fall ist es hilfreich das WebLogic JMS Forwarding Feature zu nutzen, welches automatisch Messages zu anderen Queue-Instanzen im Cluster weiterreicht, wenn diese Consumer zur Verarbeitung verbunden haben. Wichtig ist allerdings die „Forwarding Window Size" (weblogic.jms.DDWindowSize) zu limitieren, da es sonst zu Memory

Problemen kommen kann, wenn mehrere große Messages zum selben Zeitpunkt durch Forwarding weitergereicht werden. Die Größe sollte in Abhängigkeit der maximalen Message-Größe und dem verfügbaren Heap gewählt werden.

Queues sollten mit einer Quota versehen werden, um sicherzustellen, dass der verfügbare JMS Server Speicherplatz nicht durch einzelne Kommunikationskanäle ausgereizt wird und somit alle Schnittstellen blockiert werden würden. Zu beachten ist, dass beim Einsatz von Quotas die Limitierung auf JMS Server Ebene nicht mehr greift (auch wenn dies Sinn machen würde).

Ein häufiges Problem von Applikationen sind Verarbeitungsfehler von empfangen Messages. Das können temporäre Probleme wie nicht verfügbare Datenbanken oder WebServices sein, aber auch Fehler durch ungültige Datensätze, die nicht sauber aussortiert werden (Poison Messages). In diesen Fällen wird ein Rollback der JMS Message direkt eine neue Verarbeitung anstoßen, was zu einer Endlosschleife mit massiver Systembelastung führt. Schutz bietet hierfür WebLogic JMS mit dem „Redelivery Delay“. Dieses kann für Queues, oder übergreifend für Connection Factories gesetzt werden und ermöglicht die verzögerte Zustellung (z.B. von 5 Minuten) von Messages nach einem Rollback. Zusätzlich können auf Queueebene Prozesse zum finalisieren von Messages nach einem bestimmten „Redelivery Count“ konfiguriert werden.

Es ist von großem Vorteil, dass die JMS Komponenten wie Queues und Connection Factories zentral durch den JNDI Service bereitgestellt werden. Damit können Konfiguration zentral angepasst werden und Clients erhalten sie automatisch mit dem nächsten JNDI Lookup und benötigen keine lokalen Anpassungen.

Ein weiteres hilfreiches Feature ist die Message Compression, welche für Connection Factories konfiguriert werden kann. Damit ist es möglich Messages ab einer konfigurierten Größe automatisch vor der Übermittlung zum JMS Server zu komprimieren. Damit werden die Messages komprimiert im JMS Server Persistentstore abgelegt und komprimiert zum JMS Consumer geleitet. Das kann z.B. für XML Daten massiv Disk I/O sowie Netzwerktraffic einsparen, wird aber geringfügig die CPU belasten. Dieses Feature hat sich bewährt um für den JMS Server transparent größere Messages übermitteln zu können, als eigentlich laut Richtlinie erlaubt wären. Vorteilhaft ist, dass die Clients keine Logik implementieren müssen und rein durch Konfiguration das Feature nutzen können.

WebLogic JMS bietet auch das Feature der „JMS Server Migration“, womit JMS Server im Krisenfall auf andere WebLogic Server migriert werden können. Es hat sich bewährt manuelle Migration zu nutzen, damit eine koordinierte und betreute Umschaltung erfolgen kann. Die automatische Migration kann zu Störfällen und nicht kontrollierbaren Verhalten führen, wenn der JMS Server Zustand nicht zuverlässig ermittelt werden kann.

WebLogic JMS unterstützt die Nutzung von XA Transaktionen, um zum Beispiel Datenbank und JMS Operationen in einer Transaktion abzuwickeln. Allerdings muss in diesen Fällen beachtet werden, dass es keine Interoperabilität mit anderen Application Servern gibt, wenn diese als XA Koordinator WebLogic JMS als XA Resource einbinden möchten.

Performance & Tuning

Ein mögliches Mittel zur Verbesserung der Performance hinsichtlich Geschwindigkeit der Message-Übermittlung sowie Speicherung wurde bereits mit dem Message Compression Feature erwähnt.

Für einen stabilen und zuverlässigen Betrieb sind die Durchführung ausführlicher Lasttests sowie der Simulation von Problemsituation unabdingbar. Voraussetzung sind geeignete Tools zur Lastgenerierung, sowie Monitoring zur Auswertung der durchgeführten Tests. Da mit gängigen Tools

wie JMeter nicht zufriedenstellende Ergebnisse erzielt werden konnten, wurde ein eigener konfigurierbarer Java Client zur Lastgenerierung entwickelt. Dieser liefert gezielte Informationen und ermöglicht notwendige Anpassungen im Verhalten. Für die Laststufen ist es sinnvoll einen Mix verschiedener Message-Größen anhand der produktiven Kennzahlen zu definieren und auf ein Set mehrerer Queues zu verteilen. Die stufenweise Erhöhung mit Hilfe des Message-Mixes ermöglicht die Auslotung der Durchsatzgrenzen für einen JMS Server sowie des gesamten Clusters. Der Vergleich liefert Aufschluss über das Skalierungsverhalten.

In diesen Tests wurde die Datenbank der JMS Server PersistentStores als limitierende Ressource identifiziert und analoge Tests mit einem Veritas Filecluster als PersistentStores haben erhebliche Performance/Kapazitätssteigerungen aufgezeigt. Damit konnte im Vergleich abhängig vom Byte-Volumen eine Steigerung des bis zu 6fachen erzielt werden, bei stabileren Antwortzeiten.

Als wesentliche Beobachtungsgrößen wurde die Dauer der „Send“ Operation der JMS Producer gewählt. Anhand der „Current Messages“ wurde verifiziert, dass es zu keinen Message Rückstau kommt und somit die verbunden Consumer die Messages zeitnah abholen können. Das Niveau der Pending Messages dient als Indikator, wenn das System in ein Backlog gerät – das Niveau steigt unverhältnismäßig an, wenn die JMS Response Times steigen. Ein parallel laufender Health Check Client auf allen JMS Servern zeigt die Entwicklung der Response Times für jede JMS Operation.

Neben diesen klassischen Lasttests sollten auch spezielle Szenarien geprüft werden. Eines wäre das Backlog-Szenario, wenn ein JMS Consumer für längere Zeit nicht vorhanden ist und sich ein größeres Message-Volumen in den Queues aufstaut. In diesem Fall sollten die Startzeiten der JMS Server geprüft werden (je höher die Message-Anzahl, desto länger die Startzeit). Dieses Szenario sollte auch mit einer manuellen JMS Server Migration getestet werden. Zusätzlich sollte das Forwarding in diesem Fall geprüft werden, insbesondere für „flackernde“ JMS Consumer, die aufgrund einer Störung beständig neue JMS Connections erzeugen und sich dabei ständig auf anderen JMS Servern verbinden und ein Forwarding auslösen. Derartige Clients müssen natürlich gestoppt werden, aber im Falle einer Störung muss die Messaging Plattform die Situation temporär bewältigen können.

Zusammenfassung

Die Bereitstellung einer zentralen Messaging Plattform ist eine Herausforderung. Anhand des kurzen Einblicks in die verschiedenen Themen, wurde ersichtlich welche Problemfelder dabei besonders berücksichtigt werden sollten und welche Maßnahmen oder Mittel genutzt werden können, um diese zu bewältigen.

Kontaktadresse:

Thomas Störmer
Schenker AG
Kruppstr. 4
D-45128 Essen

Telefon: +49 (0) 201-8781 8441
E-Mail: thomas.stoermer@dbschenker.com
Internet: www.dbschenker.com

Dr. Alexander Königs
Schenker AG
Kruppstr. 4
D-45128 Essen

Telefon: +49 (0) 201-8781 8141
E-Mail: alexander.koenigs@dbschenker.com
Internet: www.dbschenker.com