Choosing an optimal application deployment topology for WebSphere MQ depends on many factors. They include technical requirements, such as needed functions, performance requirements, and application availability requirements, and non-technical constraints, such as deadlines, budgets, and available skills. This guide to WebSphere MQ deployment topologies will help you weigh your specific requirements and constraints, and make a good topology choice based on them.

Introduction

To benefit from this article, you should have:

- Working knowledge of IBM® WebSphere® MQ APIs
- Working knowledge of Java™ programming
- Basic knowledge of IBM WebSphere MQ administration
- Basic knowledge of Linux®

Simple topologies

Application on one server, WebSphere MQ on another

The simplest deployment topology is one server hosting the application, which connects to a remote WebSphere MQ server via client libraries. From an operational support standpoint, the application team and WebSphere MQ administration team interact when WebSphere MQ resources need to be managed, and during WebSphere MQ server software or OS patching. This interaction is necessary so that the application owners can shut down their application and prevent possible errors.

You can eliminate the need for this coordination during patching if the application is written to detect server errors and handle them gracefully. But creating reliable reconnect logic can incur a heavy burden on developers and testers. Also, a remote WebSphere MQ server slows performance because each API call must travel through the network.

Application and WebSphere MQ on same server

An alternative solution is to install WebSphere MQ on the same server as the application. This topology avoids network round-trips and network errors, and speeds performance since
WebSphere MQ is local. It also enables easy setup of XA transactions if the application enlists other resource managers, such as a database. As long as there are sufficient resources available (processors, and volatile and persistent memory), the application and WebSphere MQ on the same server can provide a high-performance solution.

Highly scalable scenarios

If application data volume increases and the vertical scaling (more processors and memory) limit is reached, horizontal scaling (more servers) will be needed. In this case, more application servers will be created, and the question of WebSphere MQ deployment topology will arise. Several options are described below, each one with advantages and disadvantages that the architect and other stakeholders will have to weigh:

Several application servers sharing one remote WebSphere MQ server

This topology has a single point of failure -- the WebSphere MQ server. Even with advanced techniques, such as creating a WebSphere MQ server as multi-instance, data corruption will incapacitate all client applications. While this configuration is an easy one from an application perspective, each application instance on its server competes with applications running on other servers for messages from the same queue.

Also, operating system and WebSphere MQ upgrades will impact all applications unless WebSphere MQ deployment is done as multi-instance queue managers. Even with multi-instance queue managers, this deployment topology should be avoided because of the single point of failure and the network traffic load.

Several application servers, each with a remote WebSphere MQ server

This deployment topology eliminates the single point of failure, but it has the same disadvantages described in its simple form of single application connecting to a single remote WebSphere MQ server. Another disadvantage is that you must configure each application differently in terms of settings to connect to WebSphere MQ. Client applications running on the same enterprise message bus as your application will need different configuration entries that tell them which WebSphere MQ server to send messages to for the application to consume.

Several application servers, each hosting its own WebSphere MQ server

This scenario has the same benefits as the simple application and WebSphere MQ on one server scenario, only scaled out. The issue still remains, as described in the above scenario, where clients on the same message bus require multiple configuration entries.

Several application servers, each hosting its own WebSphere MQ server in a cluster

Given sufficient resources, this is the scenario recommended for applications required to process medium to large data volumes. Client applications on the same message bus will simply put messages destined for the application and let WebSphere MQ handle the workload balancing
algorithm for distributing messages. WebSphere MQ also has built-in logic to include only active and available servers in the message distribution list.

In addition, WebSphere MQ clustering removes the need for complex configuration management, such as remote queue, sender, and receiver channel definitions. Couple this deployment topology with the power of WebSphere MQ multi-instance queue managers, and it will perform flawlessly while being highly available and providing high throughput.

The next section shows you how to configure applications exchanging messages on the same message bus in a WebSphere MQ cluster environment.

Configuring applications exchanging messages on the same message bus in a WebSphere MQ cluster environment

What you'll need is a Java SDK, an integrated development environment (or text editor), the WebSphere MQ libraries (server, for bindings mode), and a server running your choice of OS. These instructions use Red Hat Linux®, but a Microsoft® Windows® or UNIX® server will have similar setup procedures.

Create a Maven project with dependencies on the WebSphere MQ libraries:

```xml
<dependencies>
  <!-- MQ -->
  <dependency>
    <groupId>com.ibm</groupId>
    <artifactId>mq</artifactId>
    <version>7.1.0.2</version>
  </dependency>
  <dependency>
    <groupId>com.ibm</groupId>
    <artifactId>jmqi</artifactId>
    <version>7.1.0.2</version>
  </dependency>
</dependencies>
```

The output of the Maven project is a JAR file containing both the Sender and Receiver application:

Creating the sender application

```java
package com.ibm.test;

import java.io.IOException;
import com.ibm.mq.MQException;
import com.ibm.mq.MQMessage;
import com.ibm.mq.MQQueue;
import com.ibm.mq.MQQueueManager;
import com.ibm.mq.constants.MQConstants;

public class MQSender {
    public static void main(String[] args) {
        System.out.println("Starting Sender");
        MQQueueManager qm = null;
        MQQueue q = null;

        try {
            // connect to default queue manager specified as first program
            // argument from the command line
```
Creating the receiver application

```java
package com.ibm.test;

import java.io.IOException;
import com.ibm.mq.MQException;
import com.ibm.mq.MQGetMessageOptions;
import com.ibm.mq.MQMessage;
import com.ibm.mq.MQQueue;
import com.ibm.mq.MQQueueManager;
import com.ibm.mq.constants.MQConstants;
public class MQReceiver {
    public static void main(String[] args) {
        System.out.println("Starting Receiver");
        try {
            // connect to default queue manager specified
            // as first program argument from the command line
            MQQueueManager queueManager = new MQQueueManager(args[0]);
            MQQueue q = queueManager.accessQueue("RECEIVER.IN",
                    MQConstants.MQOO_OUTPUT +
            MQConstants.MQOO_BIND_NOT_FIXED);
            for (int i = 1; i <= 5; i++) {
                MQMessage message = new MQMessage();
                message.writeString("Test message " + i);
                q.put(message);
                System.out.println("Put test message " + i);
            }
            q.close();
            queueManager.disconnect();
        } catch (MQException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        } catch (Throwable t) {
            t.printStackTrace();
        }
    }
}
```

```java
// Connect to the receiver's cluster queue
// This queue doesn't need to be defined
// as a remote queue on the local queue manager
// Rather, it is defined in the cluster repository
// therefore this queue manager knows about it
q = queueManager.accessQueue("RECEIVER.IN",
MQConstants.MQOO_OUTPUT +
MQConstants.MQOO_BIND_NOT_FIXED);
for (int i = 1; i <= 5; i++) {
    MQMessage message = new MQMessage();
    message.writeString("Test message " + i);
    q.put(message);
    System.out.println("Put test message " + i);
}
q.close();
queueManager.disconnect();
```
```java
while (true) {
    MQMessage message = new MQMessage();
    try {
        q.get(message, mqGMO);
        System.out.println("Received message: " + message.readStringOfByteLength(message.getDataLength()));
    } catch (MQException ex) {
        // for any other error than "no message available on queue", just exit
        if (ex.reasonCode == MQConstants.MQRC_NO_MSG_AVAILABLE) {
            break;
        }
    }
    if (q != null && q.isOpen()) {
        q.close();
    }
    if (queueManager != null && queueManager.isConnected()) {
        queueManager.disconnect();
    }
} catch (MQException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} catch (Exception e) {
    e.printStackTrace();
} catch (Throwable t) {
    t.printStackTrace();
}
```

**Setting up MQ on the application servers**

For complete instructions on setting up the queue managers, see [Installing a WebSphere MQ server](#).

Upon completing the WebSphere MQ server installation, create three queue managers -- one sender and two receivers. For simplicity, it is recommended that you install all three queue managers on the same server.

From a terminal window, sudo to the mqm user and then run the following commands to create the queue managers. Then start each one in turn (you may have to cd to /opt/mqm/bin first if it is not on the PATH).

```bash
./crtmqm SENDER.QM
./strmqm SENDER.QM
./crtmqm RECEIVER.QM
./strmqm RECEIVER.QM
./crtmqm RECEIVER2.QM
./strmqm RECEIVER2.QM
```

After you get confirmation that each queue manager has been created successfully, run the following commands to create their associated listeners, which you'll background:

```bash
nohup ./runmqlsr -t TCP -p 1414 -m SENDER.QM &
nohup ./runmqlsr -t TCP -p 1415 -m RECEIVER.QM &
nohup ./runmqlsr -t TCP -p 1416 -m RECEIVER2.QM &
```
Setting up the WebSphere MQ cluster and cluster queues

Create a new cluster encompassing SENDER.QM, RECEIVER.QM, and RECEIVER2.QM, which is easy to do via the WebSphere MQ Explorer interface. For the test cluster, chose the following options:

- TEST_CLUSTER will be our cluster name
- SENDER.QM and RECEIVER.QM will be primary cluster repositories
- RECEIVER2.QM will be the secondary cluster repository

For generic instructions on how to create the cluster, see Setting up a new cluster. Here is an image of the newly created queue managers and cluster:

On each of the receiver queue managers, create the receiver input queue, either by using WebSphere MQ Explorer or by executing from the terminal:

```
runmqsc RECEIVER.QM
```

Then run the MQSC command:

```
define qlocal(RECEIVER.IN) cluster(TEST_CLUSTER) defpsist(YES)
```

Perform the same step for the RECEIVER2.QM queue manager. In the end, MQ Explorer will show:
Preparing the WebSphere MQ environment to run the applications

Before running the sample applications, make sure you run the following command from /opt/mqm/bin:

```
./setmqenv -m SENDER.QM -l
```

Run the sender program:

```
java -Djava.library.path=/opt/mqm/java/lib64 -cp
$CLASSPATH:<PATH_TO_YOUR_JAR>.jar com.ibm.test.MQSender SENDER.QM

./setmqenv -m RECEIVER.QM -l
```

Run the first receiver program:

```
java -Djava.library.path=/opt/mqm/java/lib64 -cp
$CLASSPATH:<PATH_TO_YOUR_JAR>.jar com.ibm.test.MQReceiver RECEIVER.QM

./setmqenv -m RECEIVER2.QM -l
```

Run the second receiver program:

```
java -Djava.library.path=/opt/mqm/java/lib64 -cp
$CLASSPATH:<PATH_TO_YOUR_JAR>.jar com.ibm.test.MQReceiver RECEIVER2.QM
```

The setmqenv script sets up the WebSphere MQ environment (classpath, load library path, and so on) for a specified queue manager.

Special considerations for applications running in a WebSphere MQ cluster

In anything but the most basic applications, you'll end up having multiple messages sent either as part of a group, or destined to more than one queue, or both. Therefore you need to address the situation where related messages end up on multiple local instances of cluster queues, defined on various servers. This behavior may be undesirable, and you can fix it by using the special MQOO_BIND_ON_OPEN flag after connecting to the queue manager, when opening the queue for output.
Run the sample sender application and observe that three of the five messages end up on one queue manager local instance of the cluster queue, and two end up on the other one. This distribution is due to the following line in the sender application:

```java
q = queueManager.accessQueue("RECEIVER.IN", MQConstants.MQOO_OUTPUT + MQConstants.MQOO_BIND_NOT_FIXED);
```

Experiment by changing one line in the sample sender application as follows:

```java
q = queueManager.accessQueue("RECEIVER.CLUSTER.Q", MQC.MQOO_OUTPUT + MQC.MQOO_BIND_ON_OPEN);
```

Rerun the sender, and all five messages will end up only on one queue manager local instance of the cluster queue. A subsequent run will show how the cluster workload balancing algorithm toggles between the two remote queue managers.

**Conclusion**

WebSphere MQ should be at the top of your list for asynchronous messaging. This article described several deployment topologies, each with its own advantages and disadvantages, then recommended one of them for high availability and high throughput environments. You may find one of the other topologies more suitable for your environment based on resource constraints, but please consider this topology as the recommended one.

**Acknowledgments**

The authors would like to thank IBM Senior Technical Staff Member and B2B Developer Michael Hudson for his help in reviewing and helping to improve this article.
Related topics

- **WebSphere MQ resources**
  - **WebSphere MQ knowledge center**
    A single portal to all WebSphere MQ documentation, with conceptual, task, and reference information on installing, configuring, and using WebSphere MQ.
  - **WebSphere MQ developer resources page**
    Technical resources to help you design, develop, and deploy messaging middleware with WebSphere MQ to integrate applications, Web services, and transactions on almost any platform.
  - **WebSphere MQ product page**
    Product descriptions, product news, training information, support information, trial download, and more.
  - **WebSphere MQ product family**
    A description of the ten or so different editions of WebSphere MQ.
  - **WebSphere MQ documentation library**
    WebSphere MQ information centers and product manuals.
  - **WebSphere MQ trial downloads**
    Choose from a free, 90-day, full featured trial of WebSphere MQ, or a free download of WebSphere MQ Advanced for developers.
  - **WebSphere MQ support page**
    A searchable database of support problems and their solutions, plus downloads, fixes, and problem tracking.
  - **WebSphere MQ support page**
    A searchable database of support problems and their solutions, plus downloads, fixes, and problem tracking.
  - **WebSphere MQ SupportPacs**
    Downloadable code, documentation, and performance reports for WebSphere MQ products. The majority of SupportPacs are available at no charge, while others can be purchased as fee-based services from IBM.
  - **WebSphere MQ forum**
    Get answers to your WebSphere MQ technical questions and share your knowledge with other users.
  - **WebSphere MQ public newsgroup**
    A non-IBM forum where you can get answers to your WebSphere MQ technical questions and share your WebSphere MQ knowledge with other users.

- **WebSphere resources**
  - **developerWorks WebSphere**
    Technical resources for developers who use WebSphere and related middleware products. Includes downloads, how-to information, support resources, and a free technical library of tutorials, best practices, IBM Redbooks, and product manuals.
  - **Most popular WebSphere trial downloads**
    No-charge trial downloads for key WebSphere products.
  - **WebSphere forums and community**
Product-specific forums where you can get answers to your technical questions and share your expertise with other WebSphere users.

- **WebSphere demos**
  Download and watch these self-running demos, and learn how WebSphere products can provide business advantage for your company.

- **WebSphere-related tutorials on developerWorks**
  Nearly 3000 tutorials on WebSphere and related technologies by top practitioners and consultants inside and outside IBM. Search for what you need.

- **WebSphere-related books from IBM Press**
  Convenient online ordering through Barnes & Noble.

**developerWorks resources**

- **Trial downloads for IBM software products**
  No-charge trial downloads for selected IBM® DB2®, Lotus®, Rational®, Tivoli®, and WebSphere® products.

- **Technical events for developers**
  Conferences, seminars, and technical events to deepen your technical expertise and help you succeed in your software projects.

- **developerWorks podcasts**
  Listen to interesting and offbeat interviews and discussions with software innovators.

- **IBM Education Assistant**
  A collection of multimedia educational modules that will help you better understand IBM software products and use them more effectively to meet your business requirements.