

IBM WebSphere Application Server V8.5 Administration and Configuration Guide for Liberty Profile

Anil Esen

Toshiyuki Iue

Neil Patterson

Jennifer Ricciuti



WebSphere



International Technical Support Organization

**IBM WebSphere Application Server V8.5
Administration and Configuration Guide for Liberty
Profile**

October 2015

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

Second Edition (October 2015)

This edition applies to WebSphere Application Server V8.5.5.7 Liberty profile.

© Copyright International Business Machines Corporation 2013, 2015. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

IBM Redbooks promotions	vii
Notices	ix
Trademarks	x
Preface	xi
Authors	xi
Now you can become a published author, too!	xiii
Comments welcome	xiii
Stay connected to IBM Redbooks	xiv
Chapter 1. Overview of IBM WebSphere Application Server Liberty for administrators	1
1.1 Introduction to WAS Liberty	2
1.1.1 Java EE 7 and Liberty	3
1.2 Product editions	4
1.2.1 Liberty licensing	7
1.2.2 The Liberty Repository	7
1.3 Runtime architecture	8
1.4 Feature configuration	10
1.5 Directory structure	13
1.6 Configuration files	18
1.7 System management	20
1.7.1 Topologies	21
1.8 Security	22
1.9 Multi-server environments	24
1.9.1 Using multiple non-clustered Liberty servers	25
1.9.2 Using Liberty collectives and clustered servers	26
1.10 Serviceability and troubleshooting	28
1.11 Application development and deployment tools	30
Chapter 2. Liberty for the cloud	31
2.1 Liberty in the cloud	32
2.1.1 Why Liberty is an ideal runtime for the cloud	32
2.1.2 Liberty licensing	32
2.2 Administration of Liberty on cloud platforms	33
2.2.1 Software as a service	34
2.2.2 Containers	34
2.2.3 Platform as a service	35
2.2.4 Infrastructure as a service	36
Chapter 3. Installing and updating Liberty	37
3.1 Configuring the Java Runtime	38
3.2 Installation using downloaded files and archives	39
3.2.1 Installation by extracting a Java archive file	39
3.2.2 Installation by extracting a ZIP archive file	40
3.3 Installation by IBM Installation Manager using the GUI	41
3.3.1 Install Liberty and IBM WebSphere SDK Java Technology Edition for Liberty ..	41
3.3.2 Install features and add-ons without connecting to the Internet	43

3.4	Installation on z/OS	44
3.4.1	Install WebSphere Application Server Liberty for z/OS.	44
3.4.2	Install Liberty Repository features and addons.	45
3.4.3	Install IBM WebSphere SDK Java Technology Edition for Liberty	46
3.5	Considerations for upgrading Liberty V8.5.0 to V8.5.5	46
3.6	Installing content from Liberty Repository	47
3.6.1	Installing assets by using the installUtility command	48
3.6.2	The IBM WebSphere Liberty Repository.	48
3.6.3	The Liberty Asset Repository Service.	48
3.6.4	Local directory-based repositories	50
3.7	Updating Liberty	51
3.7.1	In-place update	52
3.7.2	Side-by-side update	52
	Chapter 4. Working with Liberty profile servers.	55
4.1	Working with the bootstrap.properties file	56
4.2	Working with the server.xml file	56
4.2.1	Adding new configuration options	56
4.2.2	Using include syntax	57
4.2.3	Using variables in configuration files.	59
4.2.4	Encrypting passwords.	60
4.3	Using WebSphere developer tools to work with the configuration	61
4.4	Liberty command-line utilities	62
4.4.1	Packaging a Liberty server	62
4.4.2	Installing config snippets with the configUtility	63
4.4.3	Application client commands.	65
4.5	Use the configuration dropins folder to specify server configuration.	66
4.6	Configuring dynamic application updates	66
4.7	Starting and stopping the server using the command line	67
4.8	Classloaders and shared libraries.	68
	Chapter 5. Administering the WebSphere Liberty profile	71
5.1	Installing the sample environment.	72
5.1.1	The Liberty environment	72
5.1.2	Installing Jython	72
5.2	Flexible deployment	73
5.3	The Liberty Management API	74
5.3.1	Connecting with JMX	74
5.3.2	Connecting through the Admin Center	81
5.4	Liberty collectives	87
5.4.1	Comparing Liberty and WAS Classic	87
5.4.2	Configuring a Liberty collective controller	88
5.4.3	Registering host computers within a Liberty collective	92
5.4.4	Creating a collective member	94
5.4.5	Adding members to the Liberty profile collective.	97
5.4.6	Configuring collective controller replica sets.	104
5.4.7	Setting up a Liberty server cluster.	110
	Chapter 6. Accessing databases	115
6.1	JDBC resources	116
6.1.1	JDBC providers and data sources	116
6.1.2	WebSphere support for data sources	117
6.2	Steps to define access to a database	117
6.3	Configuring data sources in Liberty.	117

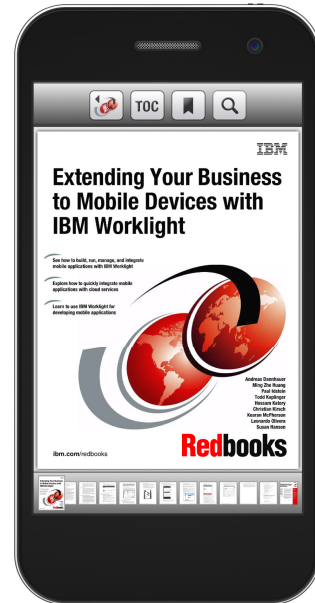
6.3.1	Configuring third-party data sources	119
6.3.2	Application-defined data sources in Liberty	120
6.3.3	Runtime data source configuration update in Liberty	121
6.4	Configuring connection pooling properties in Liberty	122
6.5	Accessing MongoDB databases	123
6.5.1	Configuring Liberty to access MongoDB APIs directly	123
6.5.2	Configuring Liberty to access MongoDB using runtime injection engine	124
6.5.3	Connecting to a distributed set of MongoDB instances	126
6.5.4	Configuring secure container-managed MongoDB connections.	126
6.6	Data access with CouchDB.	127
6.7	Logging data source activity	128
6.8	Using the timed operations feature to monitor database operations.	128
Chapter 7.	Messaging applications	131
7.1	Liberty messaging server configuration features	132
7.2	Liberty embedded JMS messaging provider.	132
7.2.1	Enabling JMS messaging for a single Liberty server	133
7.2.2	Enabling JMS messaging between two Liberty servers	136
7.3	Interoperating with the service integration bus messaging provider	138
7.3.1	Enabling service integration bus to connect to Liberty messaging	138
7.3.2	Enabling Liberty server to connect to a bus for point-to-point messaging	139
7.3.3	Enabling Liberty server to connect to a bus for publish and subscribe.	140
7.3.4	Enabling Liberty server to connect to a bus for message-driven beans	141
7.4	WebSphere MQ messaging provider	142
7.4.1	Enabling Liberty to connect WebSphere MQ	142
7.4.2	Deploying message-driven beans to connect to WebSphere MQ	143
7.5	Liberty application client container	145
7.5.1	Defining the server	145
7.5.2	Creating and configuring the client container	145
7.5.3	Deploying the JMS client application to the client container.	146
7.5.4	Starting the server and running the client	146
Chapter 8.	Monitoring the Liberty server environment	147
8.1	Introduction to performance monitoring	148
8.2	Monitoring Liberty using the monitor feature.	148
8.3	Monitoring Liberty using JConsole	149
8.3.1	Monitoring the Liberty run time remotely using a REST connector.	151
8.4	Monitoring Liberty by using the IBM Monitoring and Diagnostics Tools for Java - Health Center.	152
8.5	Monitoring Liberty using other tools	155
8.6	Tuning Liberty	159
Chapter 9.	Problem determination tools	161
9.1	Text log and trace	162
9.1.1	Configuring the server for logging.	162
9.1.2	Enabling tracing	162
9.1.3	Using the WebSphere developer tools to configure logging and trace	163
9.2	Binary log and trace	164
9.2.1	Log data repository	164
9.2.2	Trace data repository	164
9.2.3	Log and trace performance.	164
9.2.4	Configuring binary logging	165
9.2.5	Using the WebSphere developer tools to configure binary logging and trace	167
9.2.6	Reading logs with the binaryLog command	168

9.3	Creating a dump of a Liberty server	170
9.4	Event logging	171
9.5	Request timing	172
Chapter 10. Intelligent Management		173
10.1	Introduction to Intelligent Management	174
10.2	Dynamic routing	174
10.2.1	Configuring dynamic routing	175
10.3	Auto scaling	179
10.3.1	Auto scaling features	180
10.3.2	Scaling policies	181
10.3.3	Configuring auto scaling for JVM elasticity	184
10.3.4	Configuring auto scaling for Liberty elasticity	187
10.4	Maintenance mode	187
10.4.1	Configuring maintenance by using the command line	188
10.4.2	Configuring maintenance by using the Admin Center	193
10.5	Health management	195
10.5.1	Health policies	195
10.5.2	Health management controller	196
10.5.3	Configuring health management features for Liberty	196
Related publications		203
	IBM Redbooks	203
	Online resources	203
	Help from IBM	205

Find and read thousands of IBM Redbooks publications

- ▶ Search, bookmark, save and organize favorites
- ▶ Get up-to-the-minute Redbooks news and announcements
- ▶ Link to the latest Redbooks blogs and videos

Get the latest version of the Redbooks Mobile App



Promote your business in an IBM Redbooks publication

Place a Sponsorship Promotion in an IBM® Redbooks® publication, featuring your business or solution with a link to your web site.

Qualified IBM Business Partners may place a full page promotion in the most popular Redbooks publications. Imagine the power of being seen by users who download millions of Redbooks publications each year!



ibm.com/Redbooks
About Redbooks → Business Partner Programs

THIS PAGE INTENTIONALLY LEFT BLANK

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.


COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Bluemix™	MVS™	Redbooks (logo)  ®
DataPower®	Passport Advantage®	Tivoli®
DB2®	PureSystems®	WebSphere®
IBM®	Rational®	z/OS®
Informix®	Redbooks®	

The following terms are trademarks of other companies:

SoftLayer, and SoftLayer device are trademarks or registered trademarks of SoftLayer, Inc., an IBM Company.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

IBM® WebSphere® Application Server V8.5 includes a Liberty profile, which is a highly composable, dynamic application server profile. It is designed for two specific use cases: Developers with a smaller production runtime, and production environments. For developers, it focuses on the tasks that a developer does most frequently, and makes it possible for the developer to complete those tasks as quickly and as simply as possible. For production environments, it provides a dynamic, small footprint runtime to be able to maximize system resources.

This IBM Redbooks® publication targets administrators of Liberty environments. It provides the information needed to create, configure, and manage Liberty servers. It includes information about managing multiple servers in an installation, including the use of the new administrative capabilities introduced in WebSphere Application Server V8.5.5.7.

The following publications are companion publications for this book:

- ▶ *WebSphere Application Server: New Features in V8.5.5*, REDP-4870
- ▶ *WebSphere Application Server V8.5.5 Technical Overview*, REDP-4855
- ▶ *IBM WebSphere Application Server V8.5 Concepts, Planning, and Design Guide*, SG24-8022
- ▶ *WebSphere Application Server Liberty Profile Guide for Developers*, SG24-8076.

Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Raleigh Center.



Anil Esen is the team leader of the Web Middleware Enablement team in IBM Turkey. He has been working for IBM for more than four years, mostly Infrastructure Services. He has over six years of experience including WebSphere Application Server covering 5.1 through 8.5, including all editions installed on distributed platforms, IBM Content Manager, Enterprise Service Bus, Process Server, IBM DataPower® and Java Platform, Enterprise Edition technologies. He has lead many upgrade and migration projects, both between different vendors and IBM products. He designs, implements, and administers properly engineered middleware solutions to IBM clients. He has a Bachelor's degree in Computer Engineering and Master of Business Administration. He is certified in WebSphere Application Server. He gives lectures about Java Platform, Enterprise Edition concept and development, IBM Rational® Application Developer, and WebSphere Application Server in several universities in Turkey.



Toshiyuki Iue is an Advisory IT Specialist in Application Integration and Middleware group at IBM Japan Systems Engineering. He has 17 years of IT experience in IBM. For the last six years, he has worked as an expert of WebSphere Application Server for z/OS. He supports mainframe customers in Japan in terms of middleware and application, taking advantage of his previous experience as a Software Development Engineer.



Neil Patterson is a Senior IT Architect working with IBM Global Business Services in the Netherlands. He has over 30 years experience within IT and has been working with WebSphere since version 3.0 for financial institutions around the world. His areas of expertise include Service-Oriented Architecture, Master Data Management, and Model Driven Development.



Jennifer Ricciuti is a Course Developer and Instructor in IBM Training. She has 18 years of experience in developing and delivering education courses on various WebSphere products, including WebSphere Application Server, IBM Business Process Manager, WebSphere Portal Server, IBM Web Content Manager, and IBM WebSphere eXtreme Scale. Her areas of expertise include course design and development. She holds a Bachelor's degree in Computer Science from Point Park University. She works and resides in Pittsburgh, Pennsylvania.

This project was led by:

Margaret Ticknor, a Redbooks Project Leader in the Raleigh Center. She primarily leads projects about WebSphere products and IBM PureApplication System. Before joining the ITSO, Margaret worked as an IT specialist in Endicott, NY. Margaret attended the Computer Science program at State University of New York at Binghamton.

Thanks to the following people for their support of this project:

- ▶ Deana Coble, IBM Redbooks Technical Writer
- ▶ Karen Lawrence, IBM Redbooks Technical Writer
- ▶ Ann Lund, IBM Redbooks Residency Administrator

Thanks to the following people for their contributions to this project:

- ▶ David Adcox, IBM US, Software Engineer, WebSphere Liberty IM Development
- ▶ Kihup Boo, IBM Canada
- ▶ Bradley Bynum, IBM US, Software Engineer, WebSphere Liberty (System Management and UI)
- ▶ Yee-Kang Chang, IBM Canada, WebSphere Install Architect
- ▶ Steven D Clay, IBM US, Developer, WebSphere System Management
- ▶ Greg Dritschle, IBM US, WebSphere SCA Development
- ▶ Steve Fontes, IBM US, WebSphere Development
- ▶ Pamela Helyar, IBM US, WebSphere Integration Developer

- ▶ Jeff Mierzejewsk, IBM US, WebSphere Install and Configuration (z/OS)
- ▶ Alex Mullholland, IBM US, STSM WebSphere Runtimes
- ▶ Alasdair Nottingham, IBM US, WebSphere and Liberty Runtime Architect
- ▶ Kevin Schneider, IBM US, Software Engineer WebSphere Performance
- ▶ Christopher Vignola, IBM US, STSM, WebSphere Systems Management Architect
- ▶ Elson Yuen, IBM Canada, WebSphere Server Tools and Bluemix Tools Architect

Thanks to the authors of the previous edition of IBM WebSphere Application Server V8.5 Administration and Configuration Guide for Liberty Profile, published in August 2013:

Sebastian Kapciak, Gabriel Knepper Mendes, Catalin Mierlea, Sergio Pinto, Anoop Ramachandra, Carla Sadtler.

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:
<http://www.facebook.com/IBMRedbooks>
- ▶ Follow us on Twitter:
<https://twitter.com/ibmredbooks>
- ▶ Look for us on LinkedIn:
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<http://www.redbooks.ibm.com/rss.html>



Overview of IBM WebSphere Application Server Liberty for administrators

IBM WebSphere Application Server (WAS) Liberty is a lightweight, highly composable, fast to start, dynamic application server runtime environment. This chapter provides an introduction to WAS Liberty and its architecture.

In this chapter, the following topics are discussed:

- ▶ Introduction to WAS Liberty
- ▶ Product editions
- ▶ Runtime architecture
- ▶ Feature configuration
- ▶ Directory structure
- ▶ Configuration files
- ▶ System management
- ▶ Security
- ▶ Multi-server environments
- ▶ Serviceability and troubleshooting
- ▶ Application development and deployment tools

1.1 Introduction to WAS Liberty

WAS Liberty (herein called *Liberty*) is a lightweight, modular profile of IBM WebSphere Application Server. Liberty provides a dynamic, flexible runtime for Java applications, by providing the complete Java EE 7 platform and a subset of the full WebSphere Application Server API. Applications that are developed on Liberty generally run without any changes to WAS Classic.

Liberty is ideal for use in both development and production environments. Within the development environment, Liberty supports the same platforms as the full application server, plus the Mac OS X operating system. Liberty is a good option for developers who are building web applications that do not require the full Java EE environment of traditional enterprise application server profiles. Each runtime instance can be customized to match the needs of the application. In production environments, enterprise qualities of service, such as security and monitoring, are enabled as required.

Liberty has a simplified installation and uses an easy-to-configure XML configuration file format. Liberty allows fine-grained configuration of each server instance so that only those services needed by the hosted application are loaded into the server process. This approach keeps the memory footprint low and the server start time very fast. For example, if an application requires only a servlet engine, Liberty can be configured to start the kernel, an HTTP transport for connection, and the web container. If the application needs additional features, such as database connectivity, the Liberty configuration can be dynamically modified to include the Java Database Connectivity (JDBC) feature without the need of a server restart. While most of the Liberty components are shared with WAS Classic, the Liberty kernel is new and is based on Open Service Gateway initiative (OSGi) services that provide highly dynamic behavior. This method allows features, application, and configuration to be added to (and removed from) a running server, with no restarts required.

Because a Liberty server is lightweight, it can be packaged easily with applications in a compressed file. This package can be stored, distributed to colleagues, and used to deploy the application to a different location or to another system. It can even be embedded in your own product distribution.

Liberty includes the following key features:

- ▶ A dynamic and flexible runtime to load only what the application needs
- ▶ A quick start time (under 5 seconds with simple web applications)
- ▶ A simplified configuration that uses a single configuration file or modular configuration
- ▶ Support for deploying applications developed in Liberty to run in WAS Classic
- ▶ Full Java EE 7 platform and OSGi application support
- ▶ Flexible DevOps for fast, continuous deployments
- ▶ A secure server environment. User registry options include single or federated LDAP registries, role-based authorization. Other secure options for SSL, single sign-on, custom login modules, OAuth support, and more
- ▶ Integrated configuration of MongoDB and CouchDB, NoSQL database systems
- ▶ Ability to deploy an application and configured server as a package
- ▶ Ability to cluster servers for high availability and scalability
- ▶ Ability to extend Liberty with custom features by using OSGi bundles, including web application bundles
- ▶ Centralized operational management of groups of Liberty servers and of Liberty clusters

- ▶ Support for HTTP session and dynamic web content caching
- ▶ Support for binary logging
- ▶ Managed, centralized deployment for many nodes of a packaged application and server using the Job Manager
- ▶ Enhanced administration capabilities by using the Liberty Administration Center (Admin Center)
- ▶ A high-performance threading model that provides a significant performance boost for highly concurrent workloads
- ▶ Liberty performance is improved across start, footprint, and runtime with IBM JDK 8 for multiple workloads
- ▶ Availability of WebSphere Application Server Developer Tools as Eclipse plug-ins for broad tool support
- ▶ Security enhancements for Simple and Protected GSSAPI Negotiation Mechanism (SPNEGO) to enable single sign-on (SSO) mechanism for Liberty in Kerberos environments, OpenID and OpenID Connect protocols, and Open Trusted Technology Provider Standard (OTTP-S) Accreditation
- ▶ Support for IBM z/OS® platform native features such as System Authorization Facility (SAF), Resource Recovery Services (RRS), and z/OS Workload Manager (WLM)

For more information about Liberty, go to the IBM Knowledge Center at the following web address:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/cwlp_about.html

1.1.1 Java EE 7 and Liberty

Liberty now supports the full Java Platform, Enterprise Edition (Java EE) 7. Liberty also has partial support for the Java EE 6 platform, including the Java EE 6 Web Profile, web services, Java Message Service (JMS), and message-driven beans (MDBs). The Java EE 6 features should be used when developing applications on Liberty that will be deployed to WAS Classic until WAS Classic also has support for Java EE 7.

The Java EE 7 Web Profile provides specifications for web applications. Java EE 6 introduced the Web Profile to help developers of dynamic web applications, providing technologies, such as EJB Lite, Java Persistence API, and Java Transaction API. The Java EE 7 Web Profile updates the specifications of the original Web Profile and adds support for HTML5, new technologies, such as WebSocket and JavaScript Object Notation (JSON), and provides updates to existing technologies.

The Java EE 7 full platform includes the Web Profile specifications and specifications for remote EJB, web service, batch, and other applications. It also adds support for application security, deployment, and management. All Java EE 7 specifications (or JSRs) are in the full platform. Specifications for web applications are in the Web Profile, a subset of the full platform.

Java EE 7 has over 20 new or changed specifications. These new features have been added to Liberty across V8.5.5.4, V8.5.5.5, and V8.5.5.6 deliveries and they do not replace the Java EE 6 features. In fact, you do not have to take advantage of Java EE 7. You can continue to use Java EE 6 features; you do not have to migrate your existing applications. If you use any Java EE 7 features, you also need to use Java SE 7 or Java SE 8.

A key feature in Liberty is the ability to mix and match features from Java EE 6 and Java EE 7 in your Liberty environment. This technique allows you to combine features from Java EE 6 and Java EE 7 to build a complete Java stack based on your application needs. However, you cannot use the same feature in Java EE 6 and Java EE 7 on the same Liberty server instance. This statement means you cannot use two different versions of the same API on the Liberty server. For example, you cannot use servlet-3.0 from Java EE 6 and servlet-3.1 from Java EE 7 on the same Liberty server. Also, some combinations of Java EE 6 and Java EE 7 are not compatible and can cause an error when the server starts. To see a complete list of the supported Java EE 6 and Java EE 7 feature combinations, see the following IBM Knowledge Center web page:

https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/rwlp_prog_model_supported_combos.html

1.2 Product editions

Because different application scenarios require different levels of application server capabilities, WebSphere Application Server is available in multiple packaging editions. Each packaging edition includes the application server component and an appropriate combination of complementary products, for example, IBM HTTP Server, IBM Assembly and Deploy Tools for WebSphere Administration, Edge components, and other products. Although these options share a common foundation, each provides unique benefits to meet the needs of applications and the infrastructure that supports them. As your business grows, the WebSphere Application Server family provides a migration path to more complex configurations.

The following editions are available:

- ▶ WebSphere Application Server Express
- ▶ WebSphere Application Server Base
- ▶ WebSphere Application Server Network Deployment
- ▶ WebSphere Application Server for z/OS
- ▶ WebSphere Application Server for Developers
- ▶ WebSphere Application Server Hypervisor Edition
- ▶ WebSphere Application Server Liberty Core
- ▶ WebSphere Application Server Community Edition

WebSphere Application Server (base edition), WebSphere Application Server Network Deployment, and WebSphere Application Server for Developers are also available in a Tools Edition. The Tools Editions are bundles of a WebSphere Application Server runtime and development tools.

For more information about the Tools Editions, go to the following web address:

<http://www.ibm.com/software/webservers/appserv/was/tools>

WebSphere Application Server provides two runtime profiles: WAS Classic and Liberty. The runtime that has always been available with the WebSphere Application Server is referred to as *WAS Classic*, also known as the *full profile*. The application serving runtime, provided by WAS Classic, is composed of a wide spectrum of components that are always available in the application server.

Starting with WebSphere Application Server V8.5, Liberty is included with each package. Liberty is highly composable where you have many features installed but you can configure only the features that you need. You can design your server configuration to match the needs of your applications.

Liberty is also available as a stand-alone offering, called *WebSphere Application Server Liberty Core*. Each package, except for Liberty Core and Community Edition, includes both the WAS Classic application server and a Liberty application server. The Community Edition is open source based and contains neither WAS Classic or Liberty. The features available for each runtime, for example programming model support, vary among the different packaging options.

WebSphere Application Server (base edition), WebSphere Application Server Network Deployment, and WebSphere Application Server for z/OS include WebSphere eXtreme Scale in the package and entitlements to its use. Both Liberty and WAS Classic can take advantage of the caching abilities of WebSphere eXtreme Scale.

Figure 1-1 shows a high-level view of the WebSphere Application Server packaging editions.

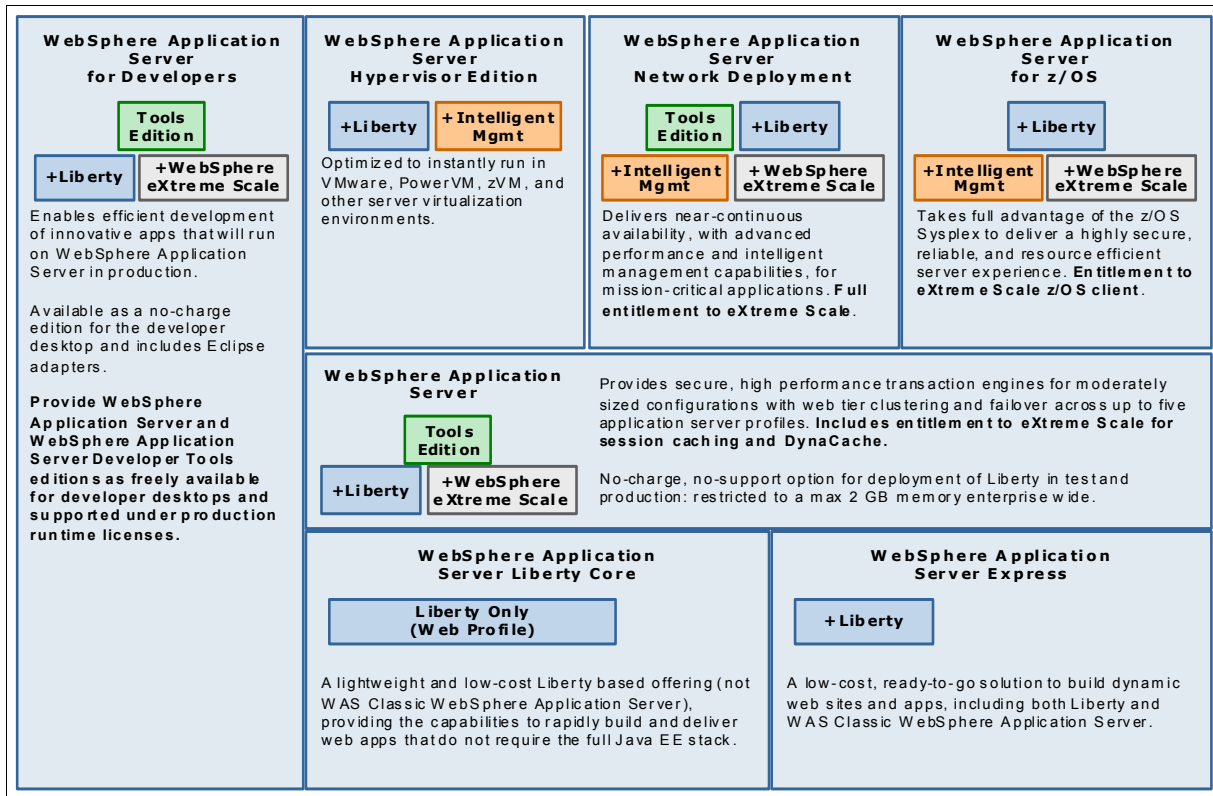


Figure 1-1 WebSphere Application Server V8.5 packaging editions

WebSphere Application Server offers a continuous delivery model to deliver new features and functions to WAS Liberty. The continuous delivery model provides new optionally installable features and functions, which can be added to an existing Liberty installation at the latest service level with no requirement for a version upgrade or migration. The continuous delivery model allows IBM to deliver features at regular intervals so you do not have to wait for these new technologies to be released at the next major release.

It is important to understand what you get when you purchase a specific edition and what installation options you have to select from. Figure 1-2 shows a high-level view of the WebSphere Application Server packaging options and what is included in each edition.

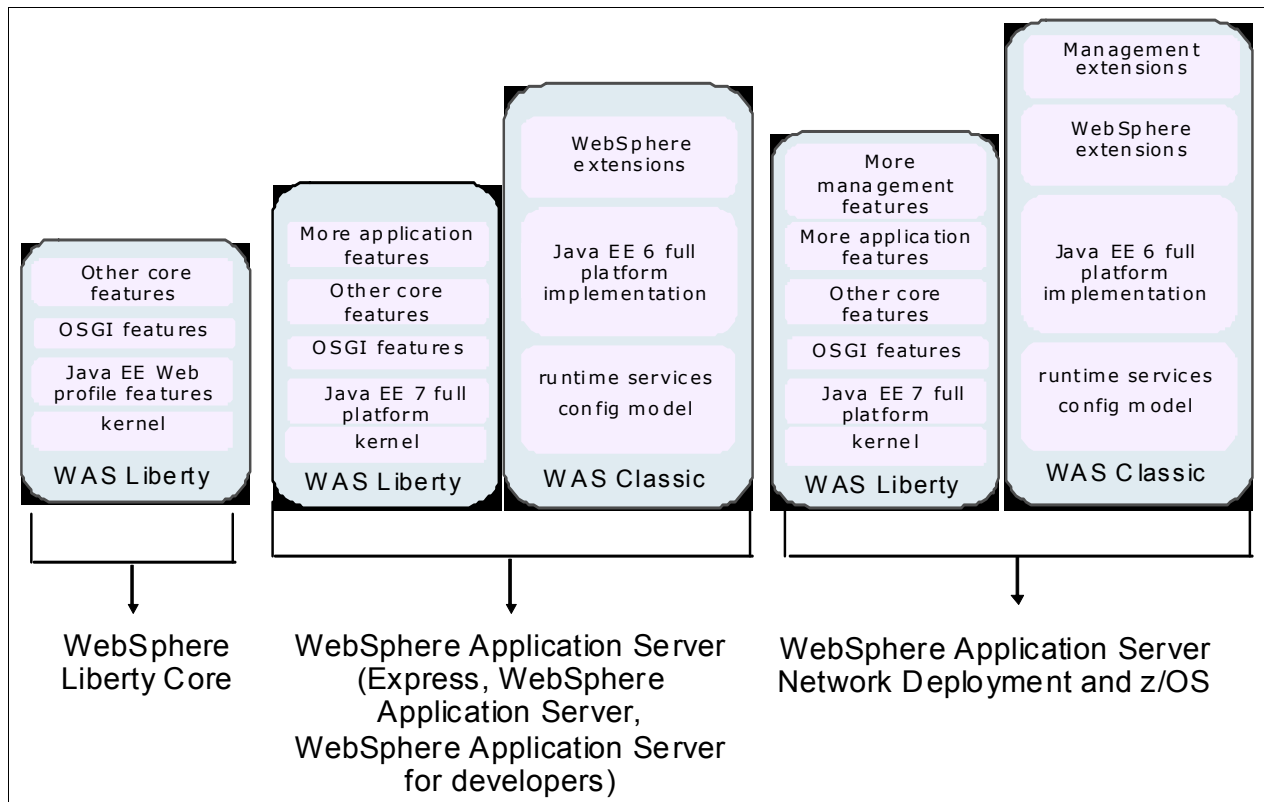


Figure 1-2 WebSphere edition offerings

There are a few key points to note on the various package editions:

- ▶ Liberty Core includes support for the Java EE 6 and Java EE 7 Web Profiles
- ▶ Liberty in WebSphere Application Server (base) and higher editions includes support for the Java EE 6 Web Profile and the Java EE 7 Full platform including the Web Profile
- ▶ WAS Classic in each of the package offerings includes support for the Java EE 6 Full platform
- ▶ Liberty in the Network Deployment edition builds on everything in Liberty in the Express and Base editions plus topology management, enterprise class clustering provided by the collective controller, and Intelligent Management features
- ▶ Liberty in the z/OS edition includes the Liberty features that come with the Network Deployment edition offering plus z/OS specific features
- ▶ When you purchase either the WebSphere Application Server Express, Base, Network Deployment, or z/OS editions, you have your choice of using WAS Liberty or WAS Classic

1.2.1 Liberty licensing

Liberty also provides a no-charge and no-support option for web-centric applications for use in small test and production environments, which includes both on-premises or in the cloud. This use is restricted to a total of 2 GB of Java virtual machine (JVM) heap size across all instances of application servers for the licensee. IBM also provides an in-place option to upgrade from a no-charge, no-support to other WebSphere Application Server package offerings.

Liberty is very flexible with license upgrades and there is a simple process to upgrade your Liberty license. For example, if you downloaded and are using the Liberty development runtime, which includes a development license. And you want to upgrade this license to a full production license. Perhaps you want clustering and auto scaling capabilities in your runtime which requires the Network Deployment license. To upgrade, simply go to Passport Advantage and download the license that you need. The license is a JAR file that you apply to your production-ready machine. In this simple process, you were able to upgrade easily from a development license to a fully supported production Network Deployment license.

1.2.2 The Liberty Repository

The rapid evolution and adoption of cloud, mobile, and social media technologies are driving the demand for delivering applications faster and more frequently. WebSphere Application Server is now delivering features for Liberty on a continual basis by using the Liberty Repository. The Liberty Repository provides an online mechanism to deliver Liberty and additional content, enabling a single point of access for various asset types. The Liberty Repository provides early access to supported new content, including new product capabilities, when they are delivered, rather than waiting for a new release.

You can use the Liberty Repository to easily extend or enhance your Liberty-based applications. The optional, production-ready features can be quickly and easily added to an existing Liberty installation. Simply choose the features that you want and then install the features to the applicable product service level. The features that you add inherit the same support of your existing installation.

In addition to features, the repository also includes artifacts, such as administration scripts, samples, configuration snippets, and artifacts that integrate open source projects more quickly and effectively. These assets are specifically designed to encompass end-to-end integration and provide important business value for the entire life-cycle of your Liberty application.

There are a few ways that you can access the online Liberty Repository:

- ▶ From the Downloads page on the WASdev.net web site
- ▶ From within the developer tools
- ▶ By using the Installation Manager and command-line utilities such as the `installUtility` command

In addition to accessing assets in the public, online Liberty Repository, you can create the following types of repositories to enable on-premises or offline access to Liberty Repository assets:

- ▶ Liberty Asset Repository Service (LARS): An open source service that you can use to create an on-premises repository that is remotely accessible behind the firewall of an enterprise.
- ▶ Local directory-based repository: Local directory-based repositories that you create when you download assets by using the `installUtility` download command.

For example, from behind your firewall you want to set up a local repository where you can install new features. You can set up a Liberty server to host a local repository. You can use the LARS client on GitHub on WASdev.net to populate your local repository on your Liberty server. You can now host a local repository and install new features through this local repository behind your firewall.

For more information about using the Liberty Repository and installing assets, see Chapter 3, “Installing and updating Liberty” on page 37.

1.3 Runtime architecture

The highly dynamic and composable nature of the Liberty runtime is achieved by using the OSGi framework as the foundation for the services that manage the component lifecycle. Liberty comprises the Liberty kernel and any number of optional features that run inside of a single JVM process. Most of the kernel runs as OSGi bundles within an OSGi framework. The kernel provides configuration, feature management, and logging services. The features that are present in the runtime are specific to the function that is needed for that server instance and its applications.

The Liberty server environment operates from a set of built-in configuration defaults. A Liberty server configuration consists of a `server.xml` file, an optional `bootstrap.properties` file, and any files that are included by these two main configuration files. The `server.xml` file is the primary configuration file for the server and contains information about the following items:

- ▶ The features to be included in the runtime environment
- ▶ The applications that are deployed into that runtime environment
- ▶ Their data sources and operational properties, such as an override to a configuration default or a trace specification

The `server.xml` file can point to (include) one or more remote XML files. This approach allows common configuration settings to be reused in multiple configuration files and to be shared across multiple servers. You can edit the `server.xml` file directly by using an XML editor, an Eclipse-based editor, or the Config tool in the Admin Center web UI.

Figure 1-3 on page 9 shows an overview of the Liberty architecture.

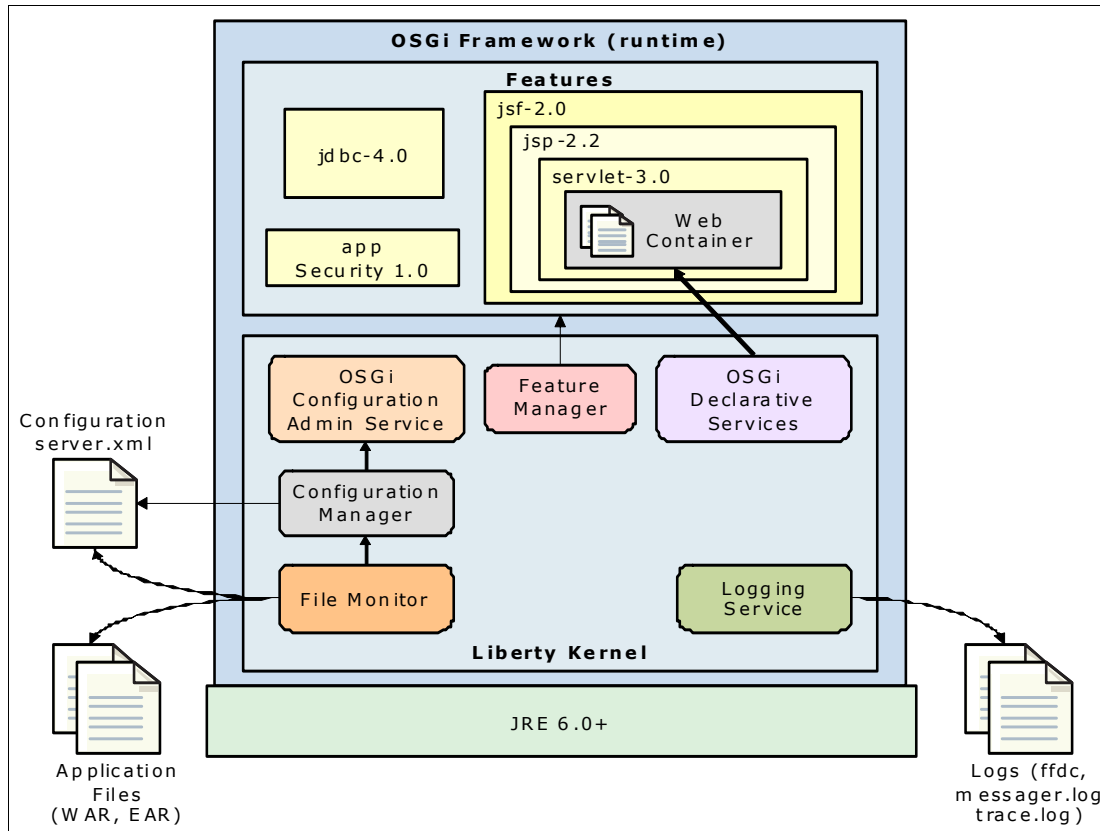


Figure 1-3 Liberty architecture

When the server is started, the launcher bootstraps the kernel and starts the OSGi framework. The server configuration is parsed, and features are loaded by the feature manager. The kernel makes extensive use of OSGi services to provide a highly dynamic runtime:

- ▶ System configuration is managed by the OSGi configuration admin service.
- ▶ The OSGi declarative services component is used to manage the lifecycle of system services.
- ▶ Application and configuration file changes are detected by the file monitor service. The file monitor service detects changes that are then reflected in real-time updates.

The use of the OSGi declarative services component enables functions to be decomposed into discrete services, which are activated only when needed. This technique helps the runtime to be *late and lazy*, keeping the footprint small and the start fast. Declarative services are added or removed from the OSGi service registry, and dependencies between services can be resolved without loading implementation classes. Service activation can be delayed until a service is used, when the service reference is resolved. Configuration for each service is injected as the service is activated and is reinjected if the configuration is later modified.

Most of the Liberty capabilities come from WAS Classic. This fact is important because when developing applications on Liberty, developers can be confident that their code runs in exactly the same way that it runs on the full WAS Classic. Also, because the tested and proven functionality from WAS Classic is used in Liberty, it is a solid foundation as a production environment. Figure 1-4 shows the stack of Liberty, which shares the same code with WAS Classic. The main exception to this is the JAX-WS web services stack: WAS Classic uses Apache Axis 2 while Liberty used Apache CXF. This design point is important to consider when developing web applications that need to run on both runtime environments as the applications need to adhere to the JAX-WS specification and not use any implementation packages directly.

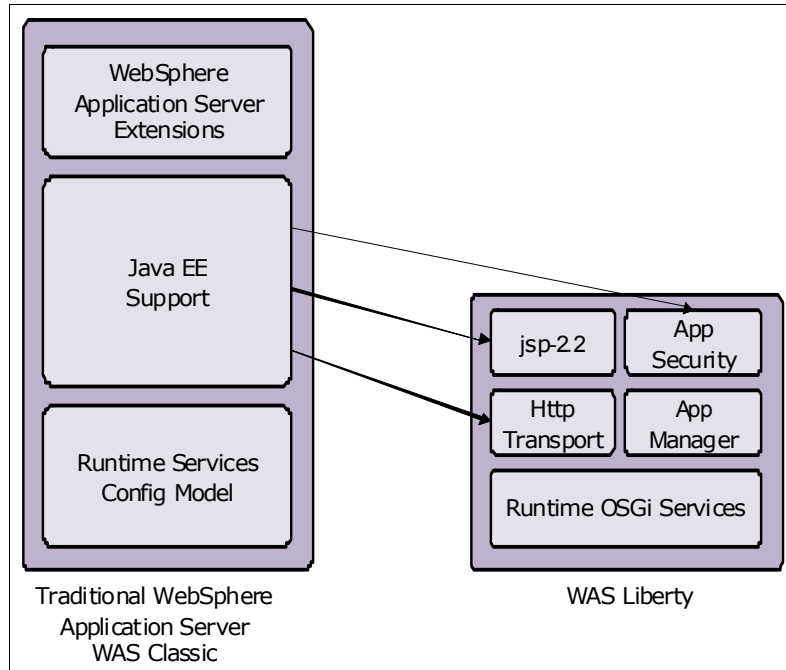


Figure 1-4 The WebSphere Application Server features as part of the Liberty stack

1.4 Feature configuration

Features are the units of functions that control the parts of the runtime environment that are loaded into a Liberty server. Each Liberty server is configured using a `server.xml` configuration file and features are specified in this file. Each feature has its own version identifier, so multiple versions of the same feature can run in the same server.

Features can define programming models, administrative capabilities, security features, and more. The set of features differs between Liberty distributions depending on the support provided with the edition. A list of key features supported by Liberty can be found in the IBM Knowledge Center in the “Liberty features” topic specific to the packaging option. For example, a full list of features for Network Deployment can be found at the following web address:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/rwlp_feat.html?cp=SSAW57_8.5.5%2F3-0-2-3-0

Each feature has a version identifier. This identifier is provided so that multiple versions of the same feature can be used in subsequent releases.

Some features include other features. For example, the `jsp-2.2` feature includes the `servlet-3.0` feature. This situation is because to run the JSP page, you need a web container. Similarly, the `jsf-2.0` feature includes the `jsp-2.2` feature.

The feature manager maps each feature name to a list of bundles that provide the feature. When a feature configuration is changed, the feature manager recalculates the list of required bundles. It stops and uninstalls those bundles that are no longer needed, and then installs and starts any additions. It also skips any features that are already loaded. All features are designed to cope with other features that are added or removed dynamically.

Figure 1-5 shows an overview of dynamic feature management in Liberty.

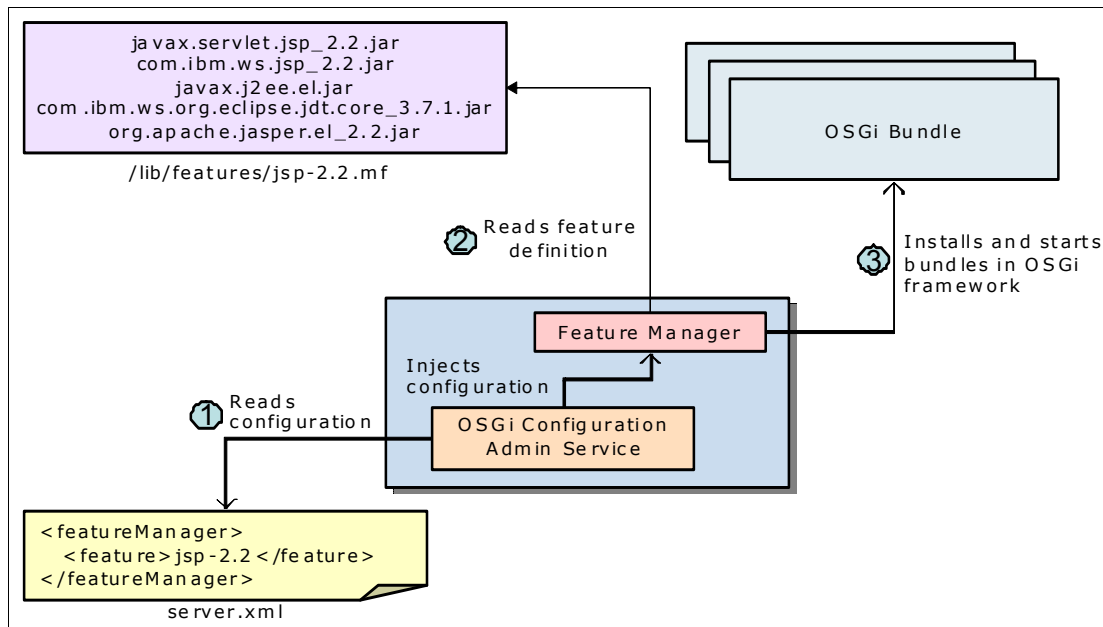


Figure 1-5 Dynamic feature management in Liberty

Features are included in a Liberty server in the following steps:

1. The OSGi Configuration Admin service reads the `server.xml` file and injects the feature configuration into the feature manager service.
2. The feature manager then maps each feature name to a list of bundles that provide the feature.
3. With all of the appropriate bundles ready, the feature manager installs and starts the features in the OSGi framework.

The feature manager also responds to configuration changes by dynamically adding and removing features while the server is running.

Figure 1-6 lists the Liberty V8.5.5.7 features (Java EE 7) that are supported in Liberty of each WebSphere Application Server package edition.

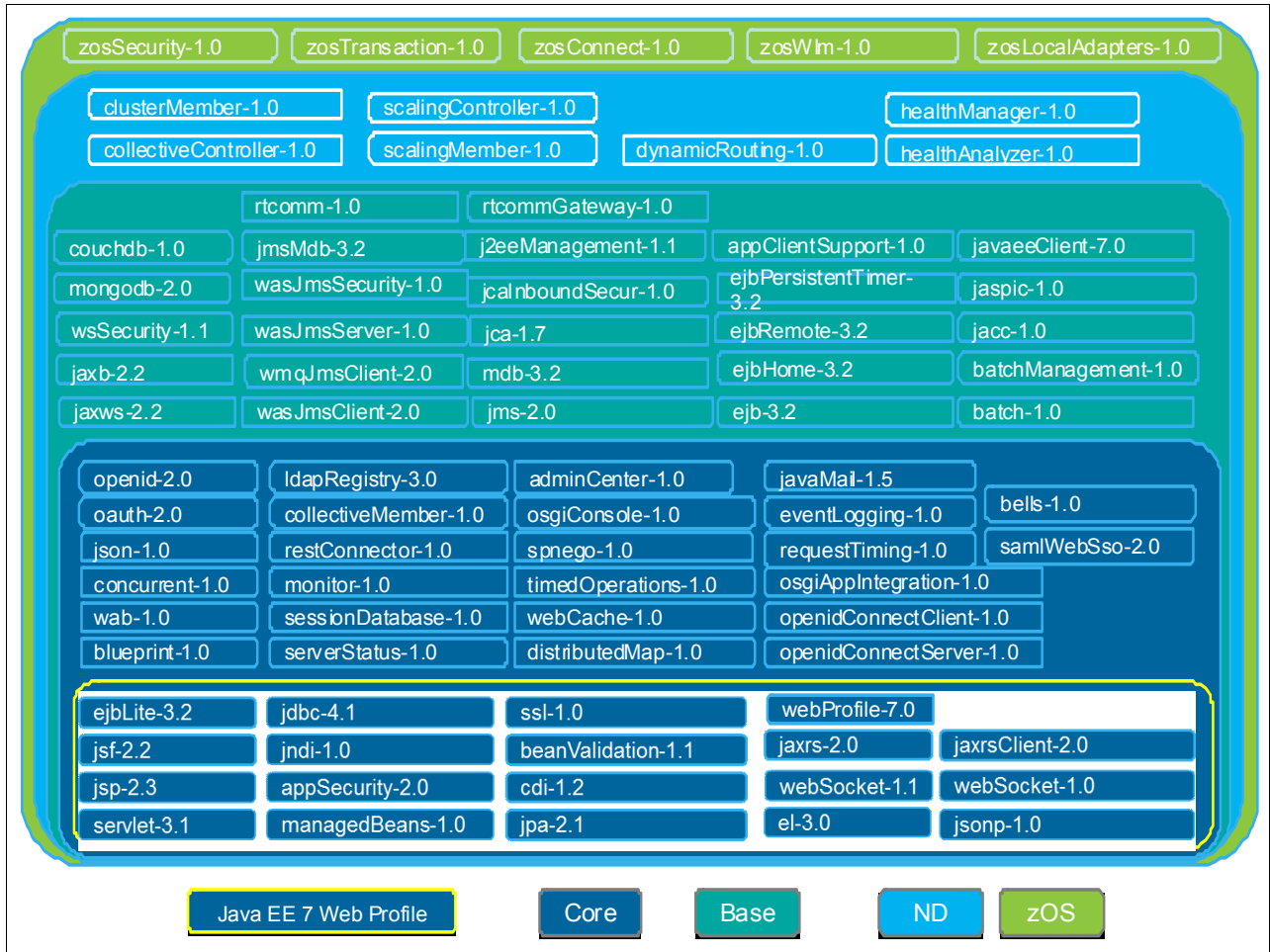


Figure 1-6 Java EE 7 features supported in Liberty V8.5.5.7 of each WebSphere Application Server package edition

Figure 1-7 on page 13 lists the Liberty V8.5.5.7 features (Java EE 6) that are supported in Liberty of each WebSphere Application Server package edition.

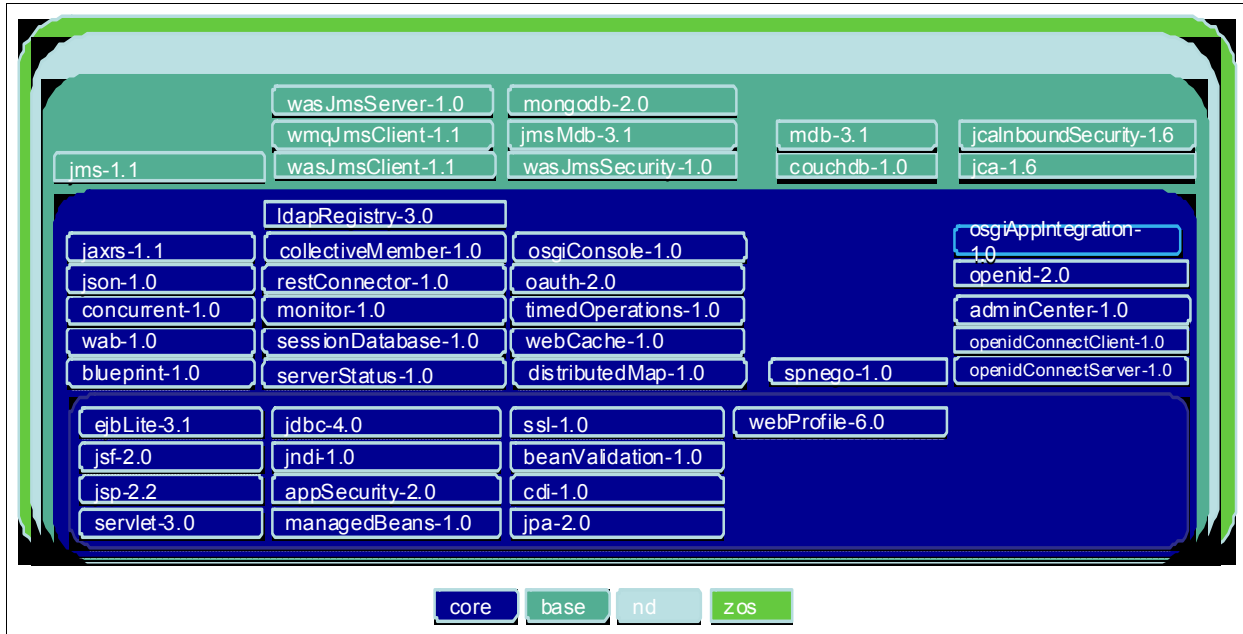


Figure 1-7 Java EE 6 features supported in Liberty V8.5.5.7 of each WebSphere Application Server package edition

1.5 Directory structure

The Liberty directory structure has three distinct areas that, by default, are all nested under the same parent location. However, they can easily be separated through the use of two environment variables. These three areas are:

- ▶ The *product files*, which may be modified by IBM service application and should not be customized by the user. These files can be placed on a read-only file system.
- ▶ The *user files*, including configuration and applications, which will not be modified by IBM service application and, which can be placed on a read-only file system.
- ▶ The *server output area*, which will be modified by the server instance as it operates and must be on a read/write file system.

Figure 1-8 shows Liberty files and the directory structure.

Tip: Not all files or folders are displayed after the installation of Liberty. Many of the files are optional and are not required by the Liberty runtime environment. The Liberty installation directory is often represented by the `{wlp.install.dir}` variable in configuration files.

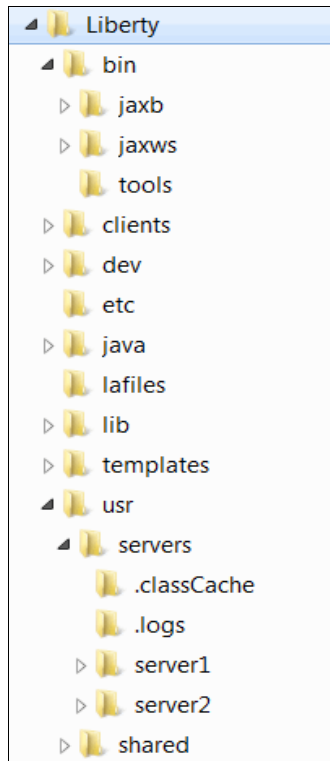


Figure 1-8 Liberty files and directory structure

The product directories include `bin`, `clients`, `dev`, `lafiles`, `lib`, and `templates`. The user directories include `etc` and `usr`. The server output directory is, by default, the `server_name` directory. For example, `server1` and `server 2`, as shown in Figure 1-8.

The key Liberty directories are:

► `bin`

This directory contains scripts used to manage the Liberty server instance for Windows (ending with `bat` extension) and UNIX (without any extension) operating systems:

- **server** and **server.bat**: Used to create, start, stop, run, package, or create dumps of the Liberty server. The IBM Knowledge Center has more information about using the **server** command at the following web address:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_command_server.html?cp=SSAW57_8.5.5%2F3-3-11-0-3-2-1-0

- **securityUtility** and **securityUtility.bat**: Used to encode passwords included in `server.xml` configurations and to create a default certification for use during a server configuration. The IBM Knowledge Center has more information about using the **securityUtility** command at the following web address:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/rwlp_command_securityutil.html?cp=SSAW57_8.5.5%2F3-3-11-0-4-1-2-0

- **featureManager** and **featureManager.bat**: Used to install a feature package as a subsystem archive (`esa`) and generate an XML list of all features included in this installation of Liberty. The IBM Knowledge Center has more information about using the **featureManager** command at the following web address:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/rwlp_command_featuremanager.html?cp=SSAW57_8.5.5%2F3-3-11-0-1-2-2-0

- **productInfo** and **productInfo.bat**: Provides a list of all features included in this installation of Liberty. Compares iFixes (applied to this current installation and a new fixpack level) and lists any interim fixes not in the fix pack. It can also compare with a supplied list of interim fixes and note if they are included in the current version. Used also to validate a production installation against a product checksum file. The IBM Knowledge Center has for more information about using the **productInfo** command at the following web address:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/rwlp_command_productinfo.html?cp=SSAW57_8.5.5%2F3-3-11-0-1-3-0

- **binaryLog** and **binaryLog.bat**: Used to view or copy the contents of a binary logging repository, or list the available server process instances in the repository. For more information about using the **binaryLog** command, go to the IBM Knowledge Center at web address:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/rwlp_logviewer.html?cp=SSAW57_8.5.5%2F1-0-2-10-0

- **configUtility** and **configUtility.bat**: Used to download configuration snippets from the IBM WebSphere Liberty Repository and to replace configuration snippet variables with your input values. The IBM Knowledge Center has for more information about using the **configUtility** command, at the following web address:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/rwlp_command_configutil.html

- **installUtility** and **installUtility.bat**: Used to install assets in your Liberty profile environment and view required asset information. Before you can access the IBM WebSphere Liberty Repository by using the **installUtility** command, you must install the beta version of the Liberty profile for WebSphere Application Server. More information about using the **installUtility** command is available in the IBM Knowledge Center at the following web address:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/t_install_assets_installUtility.html?cp=SSAW57_8.5.5%2F3-3-11-0-1-2-1

- **isadc** and **isadc.bat**: Used to run the IBM® Support Assistant Data Collector for IBM WebSphere® Application Server, a tool that you can run to gather data from your Liberty server system for problem determination purposes. More information about using the **isadc** command is available in the IBM Knowledge Center at the following web address:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/ttrb_isadcstart.html

Additional command scripts are added by some features as they are added to your Liberty environment.

- ▶ **etc**

This directory is optional. It can be used to customize the Liberty installation. The settings applied to files in this directory will apply to all Liberty servers. If there are no **etc** directory or configuration files, the default settings for the JVM and Liberty runtime are used:

- **jvm.options**: Used to customize default JVM runtime parameters
- **server.env**: Used to configure default Liberty server environment variables

For details about customizing the Liberty environment, go to IBM Knowledge Center at the following web address:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_admin_customvars.html?cp=SSAW57_8.5.5%2F3-3-11-0-3-2-0

- ▶ **lafiles**

This directory contains the Liberty license information files.

- ▶ **lib**

This directory contains the Liberty libraries.

- ▶ **templates**

This directory contains samples for specific configurations and a sample **server.xml** file for Liberty.

- ▶ **usr**

By default, this directory contains server instances with their configuration inside a *servers* directory, applications, and any resources that can be shared between servers inside the *shared* directory.

- ▶ **usr/servers/<server_name>/configDropins**

This directory is used to dynamically add a configuration to the server by placing configuration files in the directory.

In addition to the Liberty structure, there is a set of directories and files for the Liberty server instance. This root directory is often referred to by using the `${server.config.dir}` variable in **server.xml**.

Figure 1-9 illustrates the structure of a Liberty server instance called *server2*.

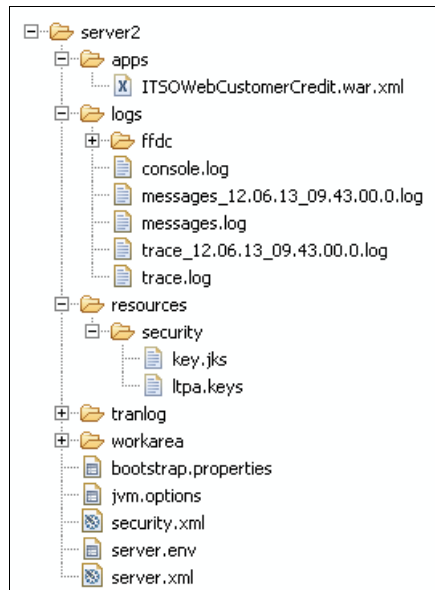


Figure 1-9 Liberty server instance files and directory structure

The key Liberty instance directories include:

► apps

This directory is optional. It can contain deployed applications or application descriptors when the applications are deployed using the WebSphere developer tools. This directory is the default location for the Liberty server to look for applications.

► dropins

This directory is created by default and is automatically monitored. If you drop an application into this directory, the application is automatically deployed on the server. Similarly, if the application is deleted from the directory, the application is automatically removed from the server. The `dropins` directory can be used for applications that do not require additional configuration, such as security role mapping. If you put your applications in the `dropins` directory, you must not include an entry for the application in the server configuration. Otherwise, the server tries to load the application twice and an error might occur.

There are several options for placing applications in the `dropins` directory. Each option provides a way for the server to determine the application type. The following list describes the placement options:

- Place the archive file with its identifying suffix (`ear`, `war`, `wab`, and so on) directly into the `dropins` directory:

```
${server.config.dir}/dropins/myApp.war
```

- Extract the archive file into a directory that is named with the application name and the identifying suffix:

```
${server.config.dir}/dropins/myApp.war/WEB-INF/...
```

- Place the archive file or the extracted archive into a subdirectory that is named with the identifying suffix:

```
${server.config.dir}/dropins/war/myApp/WEB-INF/...
```

- ▶ `configDropins`

You can place configuration dropins files in either the `configDropins/defaults` directory or in the `configDropins/overrides` directory.

- If you place these files in the `defaults` directory, the configuration is applied before the server configuration. In this case, the files provide default values, which you can override in the main `server.xml` file or the included files.
- If you place the configuration dropins files in the `overrides` directory, the configuration is applied after the server configuration. In this case, the files override the main `server.xml` or included files.

- ▶ `logs`

This directory contains the logs produced by the Liberty server. By default, this is the place where the trace or message logs are written. It also contains the first failure data captures (FFDCs). If you enable the HPEL log system, two directories are created inside the `log` directory. The two directories created are `logdata`, containing the HPEL binary database; and `tracedata`, containing the binary trace data database.

- ▶ `resources`

This directory contains additional resources for the Liberty server instance. For example, keystores generated by the Liberty server are located at this directory.

- ▶ `tranlog`

This directory contains the transactional logs that are produced by the server runtime and the applications. The transactional logs are used to commit or roll back transactional resources.

- ▶ `workarea`

This directory is created during the first server run. It contains the Liberty server operational files.

For more information about Liberty files and directories, go to the IBM Knowledge Center at the following web address:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/rwlp_dirs.html?cp=SSAW57_8.5.5%2F3-3-11-0-2-0

1.6 Configuration files

Runtime services provide their configuration defaults so that the configuration you need to specify is kept to a minimum. At server start (or when the user configuration files are changed), the kernel parses your configuration and applies it over the system defaults. The set of configuration properties belonging to each service is injected into the service each time the configuration is updated.

The Liberty configuration is described in XML files that are small, easy to back up, and easy to copy to another system. Because they are XML files, they are human readable and editable in a text editor. The files are composed so that they are easily customized. You can add features to add more configurations to the system easily.

A Liberty server can be customized using a few simple files:

- ▶ `server.xml`

The primary configuration file for the Liberty server. This file is the one non-optional configuration file. It has a simple XML format that is suitable for text editors.

► `bootstrap.properties`

An optional text file used to customize the kernel bootstrap process or to specify additional variables for use in `server.xml`.

► `jvm.options`

An optional file used to specify JVM options for the server. If this file is present, it supersedes the `jvm.options` file in the `/etc` directory (only one file is used).

► `server.env`

An optional file used to customize environment variables used to launch the server. If both this file and the `/etc/server.env` file are present, the contents of both are merged together with values specific to the server superseding values specified for the installation.

The optional files are not created by default, whereas `server.xml` is always created. For more information about these files, see the `README.TXT` file in the installation directory or the IBM Knowledge Center at the following web address:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/twlp_admin_customvars.html?cp=SSAW57_8.5.5%2F3-3-11-0-3-2-0

To effectively manage the configuration files of multiple servers, the administrator can create a globally accessible file share where the configuration files reside. Each server has access to this file share and uses it as its main configuration repository. A similar solution can be applied to a shared application's directory, so only one version of the application is used by all Liberty servers.

These techniques enable the administrator to control the server runtime configuration from a single place. For environments where there are multiple administrators with different roles who manage different aspects of the server, the configuration of the server can be placed in separate files. Each file contains configuration fragments dedicated to a given administrator and is referenced by the main configuration file using the *include* tag, as illustrated in Figure 1-10 on page 20.

Authorization to the configuration files can be achieved on the operating system level. For example, a deployer has access to use a shared applications directory and has permissions to write to the `apps.xml` file. Similarly, the configuration of the user registry can be dedicated to a separate user who is authorized to the `ldapRegistry.xml` file.

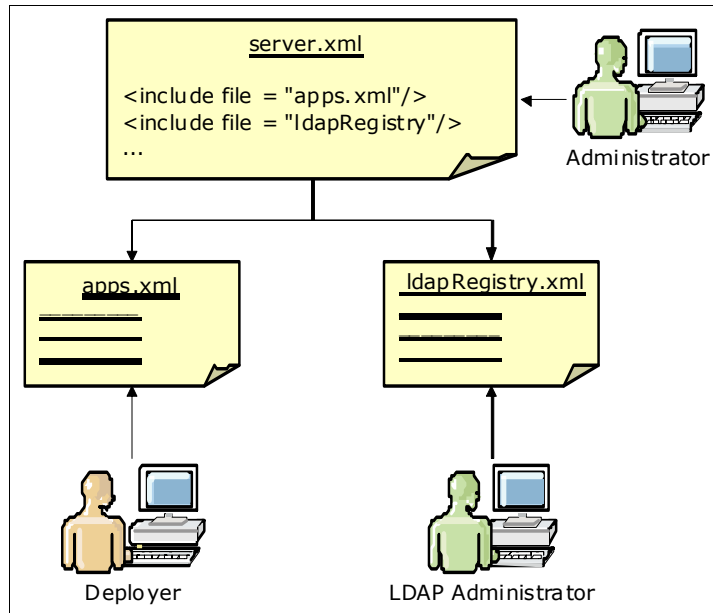


Figure 1-10 Liberty management in multi-administrator environments

1.7 System management

Typical administration actions in a Liberty environment include creating, starting, and stopping a server, querying the status of a server, packaging a server, and performing troubleshooting actions, such as creating a dump for diagnosis.

Liberty servers are administered using line commands, scripts, the Admin Center web UI, APIs, or from the WebSphere developer tools. Liberty Management APIs include Java Management Extensions (JMX) MBeans and Representational State Transfer (REST) application programming interfaces (APIs). Liberty can be administered through JMX calls to the server MBeans and by direct modification or replacement of the configuration files; no command-line tool is required. REST APIs can be used for mapping to MBeans or for file transfer.

Liberty does not ship a scripting language runtime. However, for simple and flexible management by using scripting, you can use the following items:

- ▶ Any Java enabled language such as Jython, JRuby, Groovy, or others
- ▶ Any REST-capable language such as Python, CURL, Go, or others
- ▶ Various sample scripts on WASdev.net

On a z/OS platform, you can use IBM MVS™ operator commands to start, stop, or modify Liberty. Liberty servers can also be accessed from JMX clients, such as using the jConsole tool provided in the Java SDK to monitor data.

Using scripts: You can find administration scripts on the WASdev.net website. Select the **Downloads** option; then, search on **Admin scripts** to see a list.

The WASdev.net website can be found at the following link:

<https://developer.ibm.com/wasdev>

The WebSphere Liberty Administrative Center (Admin Center) feature is available for Liberty, which provides a web-based graphical interface for Liberty servers and resource management. It has been designed around a toolbox model, so you can select tools in a customized Admin Center instance. Do not confuse this with the Integration Solutions Console, also called the Admin Console, in WAS Classic. The Liberty Admin Center is designed to be more goal and task oriented.

The Admin Center tools allow operations against both stand-alone servers and collectives. You can view the whole collective topology and can control servers and applications, and deploying server packages to host machines that are registered to the collective. You can also control the set of tools that you have access to from the Admin Center.

For more information about using the Admin Center web UI, see Chapter 4, “Working with Liberty profile servers” on page 55.

1.7.1 Topologies

One of the benefits of Liberty is that you have a wide choice of options when configuring a deployment environment. You can deploy Liberty as a stand-alone individual server or as part of a collective, which is used to manage multiple servers from a single management domain. You can also deploy Liberty as part of a traditional WebSphere Application Server cell.

Liberty, being small, easy to run and use, and flexible, it lends well to running in the cloud. You can deploy Liberty in a platform as a service (PaaS), various on-premises cloud offerings, and in containers. Liberty is being designed to run in any environment that you want to run it in.

Liberty servers can be deployed and administered individually, from a job manager, or by a collective controller as part of a collective. The collective controller provides for a centralized administrative control point to perform operations.

The Job Manager is a server type that was added to support flexible management. Both the Job Manager and the collective controller provide agent-less management of Liberty servers. Either tool allows you to create, update, and remove servers, and to update server configurations and applications. The collective controller, like the Job Manager, allows you to monitor server status but without having to submit a job. The collective controller however, also allows you to cluster Liberty servers for high availability and scalability.

In V8.5.5, the WebSphere Network Deployment Assisted Lifecycle model was extended to include Liberty servers. Under this model, existing Liberty servers can be started, stopped, and monitored from a deployment manager process in a traditional WebSphere Application Server cell. Server configuration and log files can be uploaded to and downloaded from the deployment manager process, and viewed in a text editor in the WebSphere administration console. Unlike the Job Manager model, Assisted Lifecycle can also manage Liberty servers in dynamic clusters, meaning that the servers can be automatically stopped and started in response to changes in workload. Servers cannot be created using this model, and node agents are required on the Liberty host machines.

Liberty in the cloud and containers

Liberty is an enterprise Java application platform that provides separation of concern and independence from the target hosting environment. This platform can include:

- ▶ On-premises bare metal hardware
- ▶ Hypervisors or lightweight container (Docker for virtualized environments)
- ▶ On-premises cloud offerings (IBM PureApplication Systems)

- ▶ Several infrastructure as a service (IaaS) providers (IBM SoftLayer®, Amazon AWS, and Microsoft Azure)
- ▶ PaaS providers (Bluemix, Cloud Foundry, OpenShift, and Heroku)

Docker is an open source platform that uses Linux containerization and a layered file system. A Docker image containing the IBM WebSphere Application Server for Developers Liberty V8.5.5 is available on Docker Hub. This image allows you to get the lightweight Liberty server up and running quickly in your Docker environment with only a single command. Docker enhancements include ease of configuring a Liberty server into a Docker container and run it anywhere. Liberty Docker images are available as an official repository on Docker Hub. For more flexibility, you can easily build a custom image using the Dockerfiles (build scripts for Docker images) from the open source GitHub repository on WASdev.net. It is important to note that running Liberty inside a Docker container is fully supported for production use.

For more information about Docker and Liberty, go to the following web address:

https://hub.docker.com/_/websphere-liberty

Liberty collectives move nicely to IaaS clouds, such as SoftLayer and IBM PureSystems®, which provide efficient deployment of cells and collectives both on-premises and off-premises. If you want more control over the environment, you can let IBM manage the hardware and the operating system and get started immediately on your project with the IBM SoftLayer IaaS offering. With flexible billing, on-demand deployment, and single-screen management, it is a great way to build your Liberty cloud. You deploy your applications in a Liberty package compressed file along with Java, extract, and run. You can set up a complete deployment in a few hours instead of having to wait for hardware.

IBM Bluemix™ is a platform as a service (PaaS) offering that delivers quick and easy cloud capabilities to deploy and maintain your web application, with minimal hassle and overhead. Hosting applications on Bluemix provides users with many advantages. Bluemix is an end-to-end offering that provides developers with a complete set of DevOps tools and integrated services to simplify development, test, build, and deploy applications. Moreover, applications that are hosted by Bluemix have access to the capabilities of the underlying cloud infrastructure. Such an infrastructure provides the best support for non-functional requirements (such as scalability, performance, availability, and security) that are needed for enterprise applications. All that you need is your web application. IBM supplies everything else. You can use a command-line client or lightweight Eclipse tools to deploy your application into the Bluemix cloud, and it is immediately available over the Internet. The Liberty runtime is available through the Liberty build pack, which can automatically bind your application to many of the Bluemix services, so they are quick and easy to use.

For a list of Liberty features supported in Bluemix, go to the following web address:

<https://www.ng.bluemix.net/docs/starters/liberty/index.html#libertyfeatures>

1.8 Security

Security is an essential component of any enterprise-level application. Liberty provides support for securing the server runtime environment and applications by using user registries, authentication, and authorization. For secure communication between the client and the server, you can enable SSL for Liberty. A minimal or detailed configuration can be done by adding the ssl-1.0 server feature to the server configuration file.

For authenticating users, Liberty supports the following configurations:

- ▶ A basic user registry that defines user and group information for authentication to the Liberty server.
- ▶ A Lightweight Directory Access Protocol (LDAP) server used for authentication.
- ▶ System Authorization Facility (SAF) registry for authorization on z/OS.
- ▶ Federated LDAP registries, where two or more LDAP registries are defined so that the operations, such as a search for a user, are executed on all the registries.
- ▶ Custom user registries installed as an extension to Liberty.
- ▶ Integration with a third-party security service using trust association interceptors (TAIs). A TAI is used to validate HTTP requests between a third-party security server and a Liberty server. The TAI can be called before or after single sign-on (SSO).
- ▶ SSO so that web users can authenticate once when accessing Liberty resources such as HTML, JSP files, and servlets. Users can also authenticate once when accessing resources in multiple Liberty servers that share Lightweight Third Party Authentication (LTPA) keys.
- ▶ A custom Java Authentication and Authorization Service (JAAS) login module to make additional authentication decisions or to make finer-grained authorization decisions inside an application.

The sync-to-OS-thread feature for z/OS allows the synchronization of a Java thread identity (or JAAS subject) with the OS thread identity during the current Java EE application request. If you do not choose this option, the OS thread identity value is the same as the servant identity value.

To configure authorizations for an application, you can add authorization tables to the application. The server then reads the deployment descriptor of the application to determine whether the user or group has the privilege to access the resource.

Authorization to resources by using the OAuth 2.0 protocol is also supported. *OAuth* is an open standard for delegated authorization. With the OAuth authorization framework, a user can grant a third-party application access to their information stored with another HTTP service without sharing their access permissions or the full extent of their data.

The enforcement of security constraints to applications, and to web service and JMS transports, can be added incrementally to the server configuration by using distinct features. This approach aids the application developer in performing simple unit testing of the application function first, then enabling security for a second, iterative phase of testing.

Liberty provides a utility to encrypt passwords that are stored in the server.xml file. Three mechanisms are supported: Exclusive decisions and merges (XOR) (the default), Advanced Encryption Standard (AES), and hash. It is important to note that encrypting passwords in the configuration files is a common corporate requirement, but does not, in isolation, make the passwords secure. Either the passwords themselves or the encryption key, must be kept in a separate (included) configuration file that is protected by operating system file permissions or some similar secure mechanism.

JMX clients connecting to a Liberty server have two options. The client can use the local connector, which is protected by the policy implemented by the SDK in use. Currently, that policy requires that the client runs on the same host as the Liberty server, and under the same user ID. Clients that want to connect to a remote Liberty server use the REST connector. Remote access through the REST connector is protected by a single administrator role and the use of Secure Sockets Layer (SSL).

There are several security configuration examples on the WASdev.net website for reference when configuring security for your applications on Liberty, available at the following web address:

<https://www.ibmdev.net/wasdev/category/repo/config-snippets>

The Liberty server also provides various plug points that extend the security infrastructure.

Liberty supports the following key security capabilities:

- ▶ SAML 2.0 enables WAS Liberty to support SAML 2.0 web browser single sign-on profile. A web user authenticates to a SAML identity provider (IdP), and Liberty makes the authorization decision based on assertion from IdP without the requirement of on-premises user registry.
- ▶ The Simple and Protected GSS API Negotiation Mechanism (SPNEGO) to enable SSO mechanism for Liberty in Kerberos environments. This feature provides capabilities similar to those available for WAS Classic.
- ▶ OpenID and OpenID Connect protocols to help simplify the task of authenticating and authorizing mobile and cloud centric applications.
- ▶ Open Trusted Technology Provider Standard (OTTP-S) Accreditation: WebSphere Application Server is now the leading OTTP-S accredited application server provider.

For more details about securing Liberty, go to the related IBM Knowledge Center web page at the following web address:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/cwlp_sec.html

1.9 Multi-server environments

Individual developers typically work in single-server environments, but at the same time, Liberty is also suited for production. Using multiple Liberty servers can provide the availability and scalability for running critical applications.

1.9.1 Using multiple non-clustered Liberty servers

Before V8.5.5, Liberty did not offer the option of clustering servers for availability and scalability, but you can distribute work among multiple servers. Figure 1-11 illustrates an example of a Liberty topology that meets these requirements.

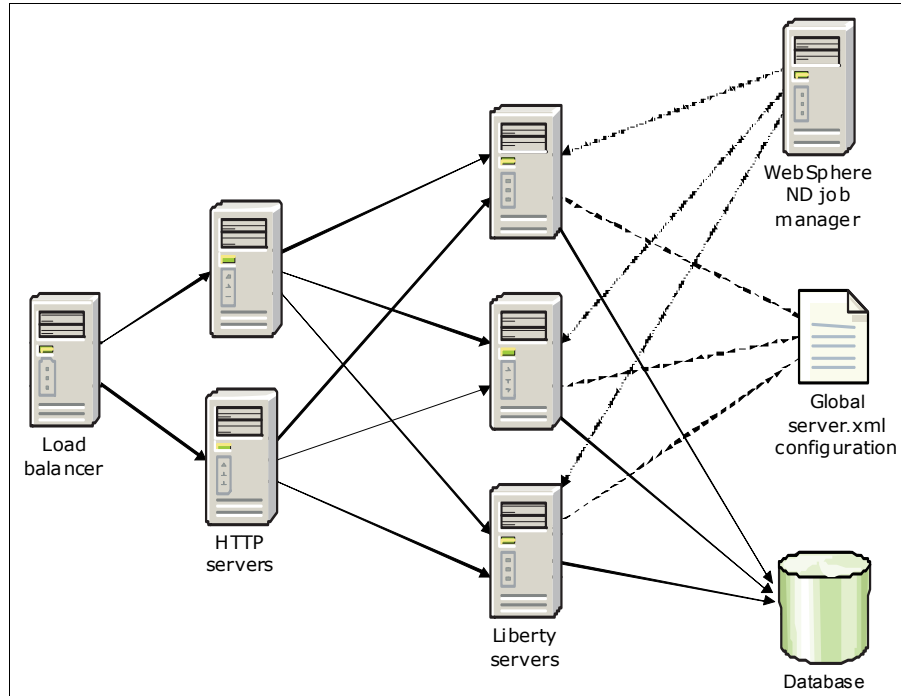


Figure 1-11 Example of a multiple Liberty server topology

To achieve such a topology, multiple Liberty servers are installed. The inbound traffic to the applications is spread by the HTTP servers, which are configured with the Liberty servers using the HTTP server plug-in or configuration specific to the HTTP proxy of choice. To distribute incoming requests, a load balancer might also be used (for example, a hardware appliance or IBM WebSphere Application Server Edge Components). Notice that a database is used for session persistence to allow failover of stateful HTTP sessions. For this purpose, the `sessionDatabase-1.0` feature is used.

To improve the administration aspect of such a topology, the WebSphere Network Deployment job manager profile is used in this example to deploy and manage all of the Liberty servers using the WebSphere centralized installation manager component. This reduces the time needed to manage all the Liberty servers because it can be automated and done remotely.

Configuration files: To ensure that the Liberty servers run with the same resources, a global configuration file can be made available for all instances, for example, using a shared file system or remote HTTP server. All Liberty servers read the shared `server.xml` configuration file but keep their local `bootstrap.properties` file where their environment configuration is kept. This process allows the Liberty servers to be installed on different operating systems. The same method can be used to configure applications, where they are placed in a single, shared directory that is available to all server instances.

1.9.2 Using Liberty collectives and clustered servers

Liberty servers can now be administered as a part of a common management domain, called a *Liberty collective*. This structure is added to the administration options for Liberty for operational efficiency and convenience and to introduce high availability features.

A collective comprises at least one Liberty server configured as a collective controller and possibly one or more Liberty servers configured as collective members. Membership in a Liberty collective is optional. For a member to be part of a collective, it has to join a collective controller. A member can join only one collective. The collective can have more than one controller for failover and workload balancing reasons, but the member only communicates with one controller at a time. The communication between the member and the controller is done over the IBM JMX Rest Controller with MBean operations. Communication between controllers and members is always authenticated and protected using SSL.

A Liberty server that is configured as the collective controller can optionally provide full lifecycle management to all members in the collective, including product installation and maintenance, and operational access to all servers in the collective, without requiring an agent. The collective controller includes operations to start and stop servers, invoke administrative operations, and perform file transfer in support of configuration changes and application installation. All Liberty servers can be members of a collective, but only Network Deployment or WebSphere Application Server for z/OS provide the support needed to create a collective controller.

A set of collective controllers is called a *replica set*. There can be only one replica set per collective and all controllers must be part of it. When there is more than one collective controller, each collective controller replicates its data to the other collective controllers in the replica set to allow for high availability and data protection. The replica set is logically present even when only one controller is in use.

Liberty servers in a collective can be clustered to provide scalability and availability of applications. The cluster can be treated as a single object in the collective, simplifying the operational management of the servers in the cluster. The members of the cluster can be configured individually, or can share a configuration. A single collective can have multiple clusters, but a server can only be part of one cluster at a time.

A Liberty server that is configured into a collective can join a cluster by enabling the proper feature and configuring the cluster name. All members that specify the same cluster name are members of that cluster. The recommendation for a replica set is that it contains at least three collective controllers. A web server plug-in is used to distribute work across the servers in the cluster.

Figure 1-12 on page 27 illustrates a collective with clustered servers.

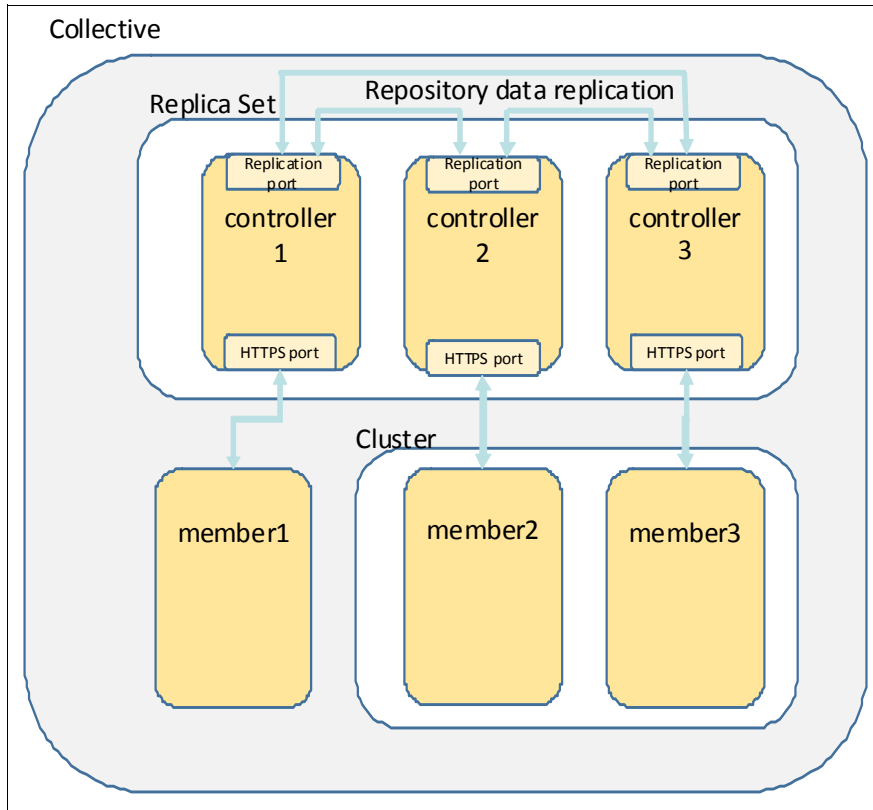


Figure 1-12 Liberty administrative domain

The collective controller provides support for managing the servers in the cluster as one object, including starting and stopping the servers, updating the configuration of the servers, installing and uninstalling applications. The collective controller also provides you the capability of adding capacity to an existing cluster and generating the merged web server plug-in configuration for the cluster.

Liberty in production

Liberty is suitable for production environments, providing a fast start, a small footprint, clustering and failover capabilities, and simple administration. The following scenario illustrates a common usage case for Liberty.

The administrator for the production system has received the applications from the developers, which are packaged for deployment. The administrator builds a production environment that consists of a cluster of several Liberty servers that are spread across multiple systems. Using a cluster provides scalability and availability features that are required for a production application environment. The administrator can start, stop, or check the status of the servers in the cluster as a single entity.

To use the cluster capabilities, the administrator builds a collective that consists of one Liberty server that acts as a collective controller and four Liberty servers to act as collective members. If failover of the controller itself is required (for highly available central management), a replica set of three or more controllers can be used (always an odd number).

Then, the collective members are configured as the application cluster. The time that is required for this is only a few minutes. Two simple commands create and configure a server as a collective controller or member. A server becomes a part of a cluster with the addition of the clusterMember-1.0 feature and naming the same cluster name in a clusterMember element. Servers can be dynamically added to and removed from the cluster by simply updating their configuration. In this scenario, the configuration for the application is stored in a common location and that file is pointed to with an include element in each server configuration file, thus deploying the application to the cluster.

The administrator generates a web server plug-in that is used by the web server to route requests among the cluster members. When the web server receives requests for the application, the plug-in routes the requests to the servers in a round-robin manner.

For more information about administering collectives and collective controllers, see Chapter 5, “Administering the WebSphere Liberty profile” on page 71.

1.10 Serviceability and troubleshooting

Liberty provides basic implementations of logging, trace, and first failure data capture (FFDC) services to help you identify and diagnose problems. Messages are sent to a single log file that contains INFO and other (AUDIT, WARNING, ERROR, FAILURE) messages. The log is a plain-text file that can be read using a text editor. If trace is enabled, the trace entries are sent to a separate trace data file. Tracing and logging settings can be set in the server.xml file, or to diagnose errors with server start (before server.xml is read), these properties can be set in the bootstrap properties.

The binary logging implementation in WAS Classic has been ported to Liberty, providing you with the same performance benefits that you find with WAS Classic. Log and trace entries are stored in a binary format in a log data or trace data repository. The binary data can be copied into a plain text format for viewing with the **binaryLog** command. Binary logging is designed to perform better than the commonly used text logs. All data that is generated by the server is stored in a repository and only formatted in a human readable form when required. Binary logging stores data in large blocks, which is more efficient than storing the same amount of data in smaller blocks.

Timed operations is a feature that tracks the duration of Java Database Connectivity (JDBC) operations running in the server and logs a warning when operations take more or less time to run than expected. A report is created in the server log file, which contains details about which operations took the longest amount of time to execute. To use timed operations, you configure the timedOperations-1.0 feature. Then, you can run the **server dump** command and the timed operations feature generates a report containing information about all operations that it has tracked. This feature helps you to see when certain actions in the server are operating more slowly than you expect, for example, to help locate bottlenecks in database access.

Liberty also provides the **server dump** command for problem diagnosis for a Liberty server. The result file that is obtained from this command contains server configuration, log information, and details of the deployed applications in the work area directory. Usually, a running server includes the following information:

- ▶ State of each OSGi bundle in the server
- ▶ Wiring information for each OSGi bundle in the server
- ▶ A component list that is managed by the Service Component Runtime (SCR)
- ▶ Detailed information about each component from SCR

Liberty also provides a **server javadump** command to help you diagnose problems on a running server at the JVM level, such as hung threads, deadlocks, excessive processor, excessive memory consumption, memory leaks, and defects in the virtual machine.

Request timing allows you to provide better troubleshooting in your environment providing early detection of failing servers. Request timing is a feature where you specify a threshold, beyond which a request is too slow or the request is hung. Implemented by using the requestTiming-1.0 feature, it provides diagnostic information when the duration of any request exceeds the configured threshold. Then, a warning message is written in the messages log file. To use this feature, you indicate either the slowRequestThreshold, hungRequestThreshold, or both in the server configuration file giving a particular value. Also, when a request is detected to be hanging, a series of three thread dumps is initiated. After the completion of the three thread dumps, further thread dumps are created only if the new requests are detected to be hanging.

Complementary to request timing is the event logging feature. The event logging feature logs events as they flow through the system, or, when the application requests are running in the Liberty server. Each request is associated with a unique correlator called the request ID and the context information that helps you to understand the request-specific data. This allows you to track a request from the beginning to the end, where you can examine the duration of the request in the exit event entries. For example, you can track servlet executions that exceed some specific time limit duration.

To help you avoid problems, Liberty provides monitoring support for the following runtime components:

- ▶ JVM
- ▶ Web applications
- ▶ Thread pools
- ▶ Database connection pools
- ▶ Messaging
- ▶ Web services

1.11 Application development and deployment tools

There are two developer tools for use with Liberty: IBM Rational Application Developer for WebSphere Software and WebSphere Application Server Developer Tools for Eclipse. Typically, an administrator would not use application development tools, but if you are running the Liberty servers in a test environment, or learning how to configure servers and deploy applications, using a developer tool provides visual tools that can help you create your configuration.

WebSphere developer tools: Rational Application Developer for WebSphere Software and WebSphere Application Server Developer Tools for Eclipse provide many of the same developer features and the process to use these features is the same regardless of the tool. When you see a reference to “WebSphere developer tools” in this book, we are referring to either of these products.

IBM Rational Application Developer for WebSphere Software V9 provides a development environment for building applications that run on WebSphere Application Server. This tool supports all Java EE artifacts that are supported by WebSphere Application Server, such as servlets, JavaServer Pages (JSP), JavaServer Faces (JSF), Enterprise JavaBeans (EJB), Extensible Markup Language (XML), Session Initiation Protocol (SIP), Portlet, and web services. It also includes integration with the OSGi programming model.

The workbench contains wizards and editors that help build standards-compliant, business-critical Java EE, Web 2.0, and service-oriented architecture applications. Code quality tools help teams find and correct problems before they escalate into expensive problems. Rational Application Developer for WebSphere Software can be used to develop applications for both WAS Classic and Liberty.

For more information about Rational Application Developer for WebSphere Software V9, go to the following web address:

<http://www.ibm.com/software/awdtools/developer/application>

The IBM WebSphere Application Server Developer Tools for Eclipse V8.5.5 provides a development environment for developing, assembling, and deploying Java EE, OSGi, Web 2.0, and Mobile applications, and supports multiple versions of WebSphere Application Server. When combined with Eclipse SDK and Eclipse Web Tools Platform, WebSphere Application Server Developer Tools for Eclipse provides a lightweight environment for developing Java EE applications.

WebSphere Application Server Developer Tools for Eclipse is a no-charge edition for developer desktop and includes Eclipse adapters. With V8.5.5, WebSphere Application Server and WebSphere Application Server Developer Tools for Eclipse editions are provided at no charge for developer desktops and supported under production runtime licenses. While not as rich in features as Rational Application Developer for WebSphere Software, this tool is an attractive option for developers using both Liberty and WAS Classic.

For more information about WebSphere Application Server Developer Tools for Eclipse and access to the tool, go to the following web address:

https://www.ibm.com/developerworks/community/blogs/wasdev/entry/downloads_final_releases?lang=en



Liberty for the cloud

Liberty, with its small runtime size, low memory footprint, and fast start time, is the only Java application server that is designed to provide a runtime environment, specifically for the cloud. There are many choices of deployment environments, such as IBM Bluemix, another platform as a service (PaaS), or containers.

The following topics are discussed in this chapter:

- ▶ Liberty in the cloud
- ▶ Administration of Liberty on cloud platforms

2.1 Liberty in the cloud

When running Java applications in the cloud, the supporting runtime needs to fit the cloud. The Liberty cloud-ready runtime environment is ideal for the cloud, with features that promote efficiency and ease-of-use.

2.1.1 Why Liberty is an ideal runtime for the cloud

Liberty is suited for the cloud for the following reasons:

- ▶ Small runtime size

Liberty is provided as a considerably small-size runtime. For example, a Java Platform, Enterprise Edition 7 Web Profile package is 63 MB. The non-feature kernel package is only 11 MB. In addition, you can use the minify package operation to repackage your Liberty server runtime environment with only the required features and your applications, and then deploy to your environment. This means you can save deployment time and reduce storage costs.

- ▶ Low memory footprint

Liberty is a highly composable and dynamic runtime environment. It only activates features that you configure, so the memory footprint is quite small. For example, a benchmark application named *TradeLite* runs under 64 MB of Java heap size. This means you can run many application instances per machine, which can diminish the megabyte-hours that you pay for.

- ▶ Fast start time

Liberty starts and stops quickly because of the lightness of the runtime size and memory footprint. It can start in just a few seconds. Liberty promotes application elasticity, enabling applications to rapidly scale-up and scale-down as the workload changes.

- ▶ Easy instance creation

Liberty can package your application, configuration, and runtime environments into a single package file, and then the application can be deployed to a new instance by unpacking, making deployment fast and easy. You also have the benefit of elastic scaling, which involves creating and destroying application instances.

- ▶ Easy migration

Liberty product files are well separated from user files. Configuration files are simple and are completely under the control of the users. If Liberty provides a new function, it is supplied as an additional feature package, and so, in most cases, does not affect your applications and configurations. This means you can easily update your Liberty runtime environment without impacting the existing applications running on your cloud, and you can keep all of your instances running at the latest middleware version to reduce risks such as security.

2.1.2 Liberty licensing

You can start your business on the cloud by using the no-charge version of Liberty. As your business grows, you can purchase an upgrade and a fee-based license.

For more information about Liberty licensing, see 1.2.1, “Liberty licensing” on page 7.

2.2 Administration of Liberty on cloud platforms

Liberty provides a common runtime environment for applications running in different cloud environments, but the administration of Liberty in these different environments tends to vary widely. At one end of the spectrum, a PaaS environment can provide a completely installed, configured, and managed runtime, with little for an administrator to do other than monitor the health of the application. At the other end of the spectrum, an infrastructure as a service (IaaS) environment simply provides a hosted virtual machine (VM), and the administrator needs to install, configure, and manage the runtime environment in much the same way as in a data center. The cloud provides a great deal of choice, and the degree of administrative involvement and control is often a key factor in deciding which type of cloud service to use. Administrative operations and responsibilities increase as you move from Instant Runtimes (Cloud Foundry¹), Containers (Docker²), and Bluemix Virtual Machines on OpenStack³, to IBM WebSphere Application Server on Cloud. You can choose the best infrastructure for your environment, or use one of these products in combination with your current application, data, and services.

Figure 2-1 shows a Liberty cloud quick compare chart.

On - premises	SoftLayer	PureApplication Service on Service	IBM Application Server on Cloud	Containers	Liberty Buildpack
CodeCode	CodeCode	CodeCode	CodeCode	CodeCode	CodeCode
Data	Data	Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware	Middleware	Middleware
OS	OS	OS	OS	OS	OS
Virtualization	Virtualization	Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking	Networking	Networking
Fully Customer Managed		Platform Managed		Pattern Managed via Console	
				Single UI Management for WAS	

Figure 2-1 Liberty cloud quick compare chart

¹ Cloud Foundry is an open source project (<https://www.cloudfoundry.org>)

² Docker is an open source project (<https://www.docker.com>)

³ OpenStack is an open source project (<https://www.openstack.org>)

2.2.1 Software as a service

In a software as a service (SaaS) platform, everything is provided for you except the application data. The vendor provides the application code, and sometimes the developer has limited access to modify the software in use. SaaS is typically not selected for deploying custom applications because the vendor provides the entire software package. Moreover, SaaS offers limited control over the hosted application, and it is often difficult to integrate external workflows into the system. Application servers that are available from IBM for use in the cloud are described as follows.

IBM Application Server on Cloud

IBM Application Server on Cloud is an offering with two platforms: IBM SoftLayer and Bluemix.

SoftLayer as software as a service

SoftLayer is a dedicated offering that provides WebSphere Application Server with a customized and entitled instance of IBM PureApplication Service. It is a simplified orchestration environment and operations console, entitled for Liberty Core for Bluemix. It is a pay-as-you-go offering with self-service, predefined patterns. PureApplication Service on SoftLayer enables the seamless extension of compute resources from on-premises to cloud environments. You can use WebSphere Application Server Patterns to create virtualization patterns for Liberty. The Bring Your Own Software and License (BYOSL) option of WebSphere Application Server lets you deploy applications on SoftLayer for a superior self-service experience.

Bluemix as software as a service

Bluemix provides Liberty as a Java runtime platform. It is an open-standard cloud platform for building, running, and managing applications. With Bluemix, developers can focus on building excellent user experiences with flexible compute options, a choice of DevOps tooling, and a powerful set of IBM and third-party application programming interfaces (APIs) and services. SoftLayer provides IaaS for off-premises cloud deployment with WebSphere Application Server.

2.2.2 Containers

Containers allow you to package an application and all of its dependencies into a standardized unit for software development and deployment. Containers can enable applications to run reliably and easily when moved from one computing environment to another. Containers guarantee that applications run in the same way, regardless of the environment the container is running in. This can be from a developer's computer to a test environment, from a staging environment to production, or from a physical machine in a data center to a VM in a private or public cloud. Every container runs applications in an isolated environment. Moreover, the isolation allows you to run many containers simultaneously on one host.

IBM Containers

IBM Containers can be used to run Docker containers in a hosted Bluemix cloud. Docker adds an engine that deploys an application to the virtual environment that you use for running your containers. Docker also provides an environment that you can use to run code. When you are ready, Docker provides the means for transferring code from development to test, and ultimately to production.

Features of IBM Containers include integrated tools, such as log analytics, performance monitoring and delivery pipeline, elastic scaling, zero downtime deployments, and automated image security and vulnerability scanning. You will also have access to the Bluemix catalog of over 100 cloud services, including IBM Watson, analytics, the Internet of Things (IoT) mobile, and more.

For more information about IBM Containers, see the following website:

https://www.ng.bluemix.net/docs/containers/container_index.html

IBM Containers provide a Liberty image. See the following website:

https://www.ng.bluemix.net/docs/containers/container_images_creating_ov.html#container_images_liberty

2.2.3 Platform as a service

In platform as a service (PaaS), everything is provided except the application code, users, and data. Typically, when using a PaaS, the vendor maintains the application server, databases, and all of the necessary operating system components. PaaS provides a complete environment that is actively managed. For most runtimes, PaaS provides scaling without modification.

IBM Bluemix Instant Runtimes

Instant Runtimes is a PaaS service provided by Bluemix. You can use runtimes to get your application up and running quickly, with no need to set up and manage VMs and operating systems. Instant Runtimes is based on Cloud Foundry, which means that community buildpacks or tooling plug-ins for Cloud Foundry also work with Instant Runtimes.

Bluemix Instant Runtimes provides a dashboard for you to create, view, and manage your applications and services, and to monitor application resource usage. With the Bluemix dashboard, you can also manage organizations, spaces, and user access. It provides access to a wide variety of services that can be incorporated into an application.

With Bluemix Instant Runtimes, you can simply push Java EE applications (WAR or EAR file formats) and select what services are needed for your application. Server configuration is generated by the Liberty buildpack. However, you might need to modify the `server.xml` file for a standard Liberty deployment. For this and similar cases, you need to package your Liberty server and push the package to Bluemix instead of the WAR or EAR file. This package includes everything on the Liberty server, including the modified `server.xml` file and your application. In Instant Runtimes, no direct involvement of an administrator is needed.

For more information about Bluemix Instant Runtimes, see the following site:

https://www.ng.bluemix.net/docs/starters/rt_landing.html

For more information about Liberty for Java runtime, powered by the Cloud Foundry Liberty build pack, see this site:

<https://www.ng.bluemix.net/docs/starters/liberty/index.html>

There are several boilerplates in Bluemix. A *boilerplate* contains an application, the associated runtime environment, and predefined services for a particular domain. You can find boilerplates that use Liberty for Java for its runtime in the Bluemix catalog at this site:

<https://console.ng.bluemix.net/catalog>

2.2.4 Infrastructure as a service

Infrastructure as a service (IaaS) is a platform where an infrastructure is provided for you. It provides basic services, such as virtual servers and data storage, in one platform. With one click, you can create VMs hosted by a provider running the operating system of your choice. The vendor providing the machine is responsible for connectivity and for initial provisioning of the system. You are responsible for all other details. For example, the vendor provides the machine and an operating system, and you install all of the software packages, application runtimes, servers, and the databases that your application requires.

Generally, IaaS requires that you have one or more system administrators to manage the system and apply firewall rules, patches, and security errata on a frequent basis. You have complete control over every aspect of the system. However, you are responsible for system uptime and security, so you need system administration knowledge or a team of administrators to maintain the system. The involvement of administrators here is similar to that of on-premises, non-cloud system administration.

IBM Bluemix Virtual Machine

Bluemix Virtual Machine is an IaaS provided by Bluemix as a beta release. It uses OpenStack software to run and manage VMs. Key OpenStack services, such as auto scaling, and object storage, can be used in conjunction with Bluemix services to build and run hybrid applications without any of the overhead of managing physical servers.

OpenStack gives the administrator full control and responsibility. WebSphere Application Server on Cloud is a ready-to-use WebSphere Application Server based on IBM PureApplication. Little or no changes are required on the existing applications to migrate from an on-premises system.

For more information about Bluemix Virtual Machines, see this site:

https://www.ng.bluemix.net/docs/virtualmachines/vm_index.html

You can run applications using any of the built-in starter packages in Bluemix Virtual Machines.

Important: You can use one of the default images that are provided with Bluemix. However, to create a VM instance from a VM image for which the operating system requires a license, you must provide the corresponding license to run it in Bluemix.



Installing and updating Liberty

You can install the Liberty application-serving environment by using several different methods. In this chapter, each method is presented so that you can use the most suitable one depending on your environment.

This chapter includes the following topics:

- ▶ Configuring the Java Runtime
- ▶ Installation using downloaded files and archives
- ▶ Installation by IBM Installation Manager using the GUI
- ▶ Installation on z/OS
- ▶ Considerations for upgrading Liberty V8.5.0 to V8.5.5
- ▶ Installing content from Liberty Repository
- ▶ Updating Liberty

In addition to the installation methods discussed in this chapter, developers also have the option of installing Liberty by using WebSphere developer tools. More information can be found in *WebSphere Application Server Liberty Profile Guide for Developers*, SG24-8076.

You can obtain the latest Liberty package from the following WASdev community website:

<https://www.ibm.com/developerworks/mydeveloperworks/blogs/wasdev/entry/download>

3.1 Configuring the Java Runtime

Liberty requires a Java Runtime Environment (JRE) to run in. You can install IBM WebSphere SDK Java Technology Edition for Liberty by using IBM Installation Manager. In addition, ZIP archive packages including IBM SDK Java Technology Edition Version 8 are available in Microsoft Windows, Linux x86, Linux PPC, and Linux PPC Language Environment platform. Otherwise, you must provide a JRE in other ways, and you must specify the JRE location to use it. The JVM on the system PATH is used by default.

Important: The minimum supported Java levels are described in the IBM Knowledge Center. See “Troubleshooting the Liberty profile → Runtime environment known restrictions” in your selected edition’s document. Following is the Network Deployment edition URL:

http://www.ibm.com/support/knowledgecenter/SS7K4U_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_restrict.html#rwlp_restrict__rest13

Liberty is supported by any compliant Java SE 6, Java SE7, or Java SE 8 runtime environment. You can use any Java SE-compliant runtime with Liberty. Where possible, an IBM provided Java is the ideal choice because IBM supports the entire stack, including Java, but you can use a JVM from Oracle or the OS vendor where applicable.

On Microsoft Windows systems, you can use the syntax in Example 3-1 to set the **JAVA_HOME** property by providing your JRE installation directory. These commands set the environment variables, and the set command is only valid in the command prompt window that you are currently in (or shell if you are on UNIX).

Example 3-1 Setting the JRE for Liberty on Windows

```
set JAVA_HOME=C:\IBM\jre6
set PATH=%JAVA_HOME%\bin;%PATH%
```

The Liberty runtime searches for the java command in the following order: **JAVA_HOME**, **JRE_HOME**, installed Liberty JRE (directed in wlp/java/java.env), and **PATH**.

Additionally, you can use the Liberty server.env configuration file to set up a specified Java Runtime. To configure the Java runtime using this file, add the following line to the file:

```
JAVA_HOME=C:\IBM\jre6
```

The server.env file can exist in server configuration directories and the etc directory in the Liberty installed location. For more information, see 1.5, “Directory structure” on page 13 and 1.6, “Configuration files” on page 18. Settings in etc/server.env are overwritten by server settings. If the file does not exist, you can create it manually.

3.2 Installation using downloaded files and archives

There are two methods for installing Liberty. You can use IBM Installation Manager or use downloaded archive files. For an archive installation, you can choose from two types of archives:

- ▶ Java archive (JAR) file

If you have a license for the WebSphere Application Server product, this type of archive file can be downloaded as an edition-specific JAR file from Passport Advantage Online (<http://www.ibm.com/software/howtobuy/passportadvantage>). The associated service is available from Fix Central (<http://www.ibm.com/support/fixcentral>). The fix packs for the archive are complete replacements, so download from Fix Central to get the latest version. You can use it in a production environment with guaranteed service levels and IBM support.

In addition, a no-charge, unsupported edition can be downloaded from the following site:

<http://WASdev.net>

- ▶ ZIP archive

Installing Liberty from the ZIP files enables no-charge, unsupported, unlimited use in development environments and unsupported limited use in small-scale test and production environments.

This type of archive file can be downloaded from Fix Central or WASdev.net.

If you download and install Liberty from an unsupported JAR or ZIP file, you can later purchase a supported edition and upgrade the license for your existing installation.

Note: On the z/OS platform, installing Liberty by extracting an archive file is not supported.

3.2.1 Installation by extracting a Java archive file

You can install Liberty by extracting a JAR file. The JAR file does not contain new features such as `javaee-7.0`. You can install additional features from the Liberty Repository; see 3.6, “Installing content from Liberty Repository” on page 47.

There are three types of archives for each edition of WebSphere Application Server:

- ▶ Runtime JAR files: `wlp-<edition>-runtime-<version>.jar`

This archive contains Liberty server and core features. It is Java EE 6 Web Profile certified. In the Network Deployment edition, Collective Controller and Static Cluster Member features are initially included.

- ▶ Extended Programming Models JAR files: `wlp-extended-<version>.jar`

This archive includes Web Services, JMS, and MongoDB support.

- ▶ Extras JAR files: `wlp-extras-<version>.jar`

This archive includes the embeddable EJB Container and JPA client.

To manually install a Liberty archive, follow these steps:

1. Extract the contents to your preferred directory:
 - a. Run the command to extract the contents of the Liberty archive, for example:

```
java -jar wlp-nd-runtime-8.5.5.7.jar
```
 - b. Press x to skip reading the license terms, or press Enter to view them.
 - c. Press Enter to view the license agreement.
 - d. Press 1 if you agree to the license terms and want to proceed.
 - e. Provide the installation path for Liberty, for example C:\IBM\WebSphere, and press Enter.

Alternative: You can use command line options as follows:

```
java -jar wlp-nd-runtime-8.5.5.7.jar --acceptLicense C:\IBM\WebSphere
```

2. Optional: Extract the programming model extensions in the same manner, for example:

```
java -jar wlp-extended-8.5.5.7.jar --acceptLicense C:\IBM\WebSphere
```
3. Optional: Set the JAVA_HOME property for your environment; see 3.1, “Configuring the Java Runtime” on page 38.

Liberty installation location: In this document, we refer to the Liberty installation directory as *Liberty_Home*. In this example, *Liberty_Home* is C:\IBM\WebSphere\wlp directory.

3.2.2 Installation by extracting a ZIP archive file

You can install Liberty and optional features by extracting a ZIP archive file. These archive files are designed to help you quickly get started with Liberty. After you install Liberty, you can install additional features by using the `installUtility` command; see 3.6, “Installing content from Liberty Repository” on page 47.

Following is a list of the Liberty ZIP archive files. You can choose what fits your requirements:

- ▶ WAS Liberty Kernel: `wlp-kernel-<version>.zip`
This basic ZIP file includes only the kernel of the Liberty server and no features.
- ▶ WAS Liberty with Java EE 7 Web Profile: `wlp-webProfile7-<version>.zip`
This archive ZIP file includes the kernel and features that support the Java EE 7 Web Profile.
- ▶ WAS Liberty with Java EE 7 Web Profile with Java 8:
`wlp-webProfile7-java8-<platform>-<architecture>-<version>.zip`
These ZIP files include the kernel, IBM SDK Java Technology Edition Version 8, and features that support the Java EE 7 Web Profile. There are individual ZIP files for each available platform and architecture.
- ▶ WAS Liberty with Java EE 7 Full Platform: `wlp-javaee7-<version>.zip`
This ZIP file includes the kernel and features that support Java EE 7.
- ▶ WAS Liberty with Java EE 7 Application Client: `wlp-javaeeClient7-<version>.zip`
This ZIP file includes the kernel and the Java EE 7 application client.

To install from a ZIP file, use the following procedure:

1. Extract the ZIP file to your preferred directory. All of the files are stored in wlp directory. For example:

```
unzip wlp-webProfile7-java8-linux-x86_64-8.5.5.7.zip
```
2. Optional: If you extract a non Java include ZIP file, set the JAVA_HOME property for your environment; see 3.1, “Configuring the Java Runtime” on page 38.

3.3 Installation by IBM Installation Manager using the GUI

To install Liberty by using IBM Installation Manager, use the following procedure:

1. Install IBM Installation Manager and prepare to install Liberty. See the following URL:
http://www.ibm.com/support/knowledgecenter/SS7K4U_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_ins_installation_dist_im.html
2. Install Liberty and IBM WebSphere SDK Java Technology Edition for Liberty.
Optional: Install features and addons without connecting to the Internet.

3.3.1 Install Liberty and IBM WebSphere SDK Java Technology Edition for Liberty

If you are using IBM Installation Manager GUI, add repositories to your IBM Installation Manager preferences. Liberty has three options for accessing the product repositories to install:

- ▶ Access the physical media, and use local installation
- ▶ Download the files from the Passport Advantage site, and use local installation
- ▶ Access the live repositories, and use web-based installation

In addition, if you add repositories for IBM WebSphere SDK Java Technology Edition for Liberty, you can install it at the same time with Liberty. You can choose a Java from Version 6.0, 7.0, 7.1, or 8.0. Version 7.1 and 8.0 repositories are only available from the web, downloaded from Fix Central or installed directly from the live repositories.

Figure 3-1 illustrates using local installation repositories of Liberty and Java.



Figure 3-1 Obtain the product repositories

Figure 3-2 shows both Liberty and SDK IBM WebSphere SDK Java Technology Edition Version 8.0 are selected to install.

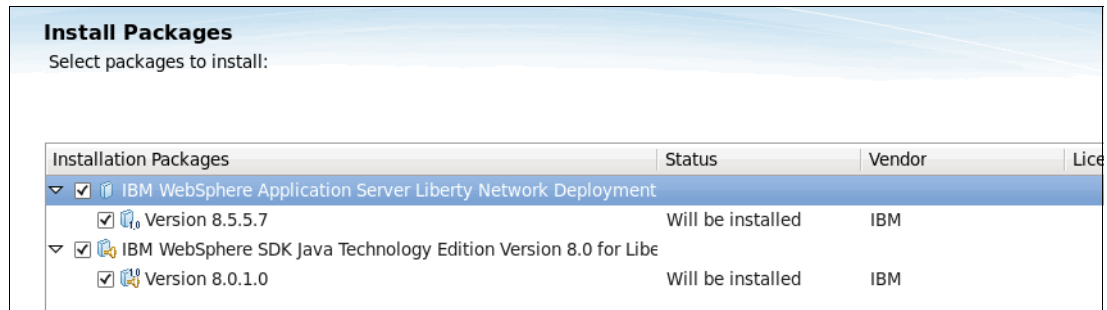


Figure 3-2 Select Liberty and SDK Java

Figure 3-3 indicates the selection of features that you want to install. You can select to install the embeddable EJB container and JPA client. After the installation is complete, this feature can also be added or removed by using IBM Installation Manager.

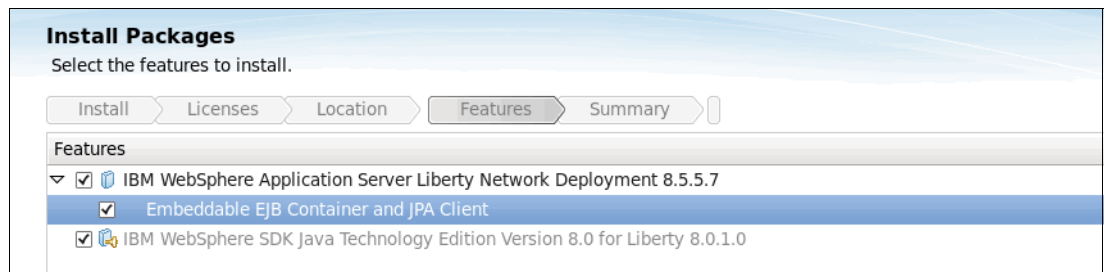


Figure 3-3 Select the features

In the next step, you can install additional Liberty Repository addons and features. If you do not want to select any additional addons and features, skip this step. After you complete the installation, you can install additional addons and features by using the `installUtility` command. To learn more about Liberty Repository and `installUtility`, see 3.6, “Installing content from Liberty Repository” on page 47.

Figure 3-4 is a selection whether you want to install addons and features from the IBM WebSphere Liberty Repository. If you choose not to connect to the IBM WebSphere Liberty Repository, you can still install addons and features from the configured location, see 3.3.2, “Install features and addons without connecting to the Internet” on page 43.

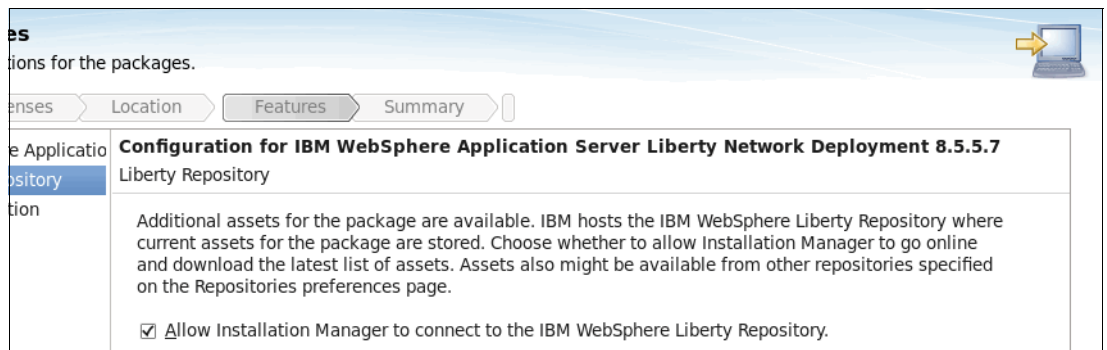


Figure 3-4 Select whether you want to install addons and features from the IBM WebSphere Liberty Repository

In this step, you can launch an Asset Selection wizard, which is shown in Figure 3-5. Select each addon or feature that you want to install.

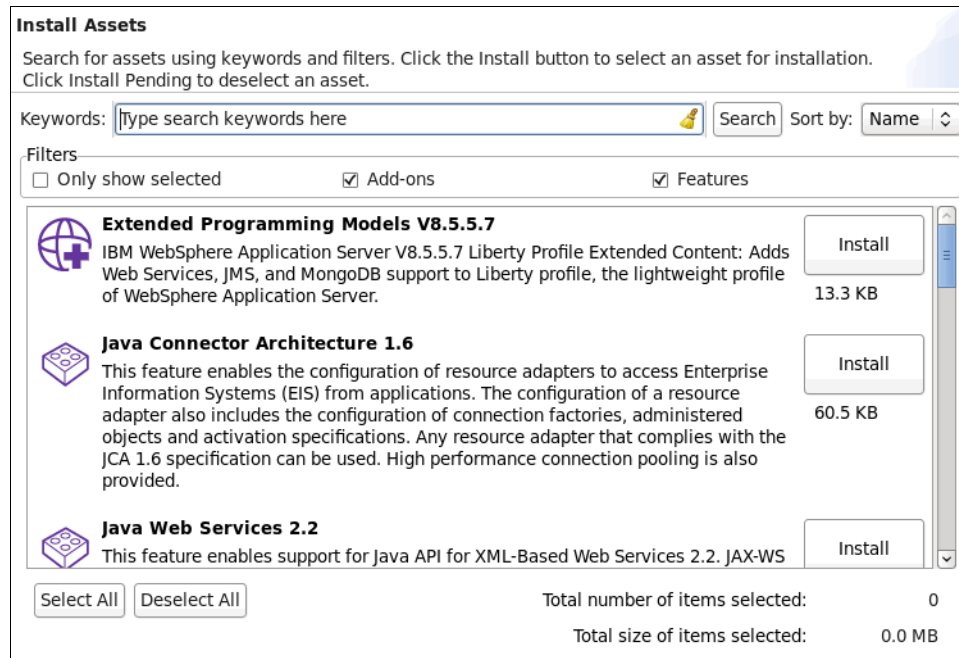


Figure 3-5 Asset Selection wizard

3.3.2 Install features and addons without connecting to the Internet

If your environment cannot connect to the IBM WebSphere Liberty Repository, you can install addons and features from configured directory-based repositories or an instance of the Liberty Asset Repository Service. You can add the repository URL or directory to your IBM Installation Manager preferences, as illustrated in Figure 3-6.

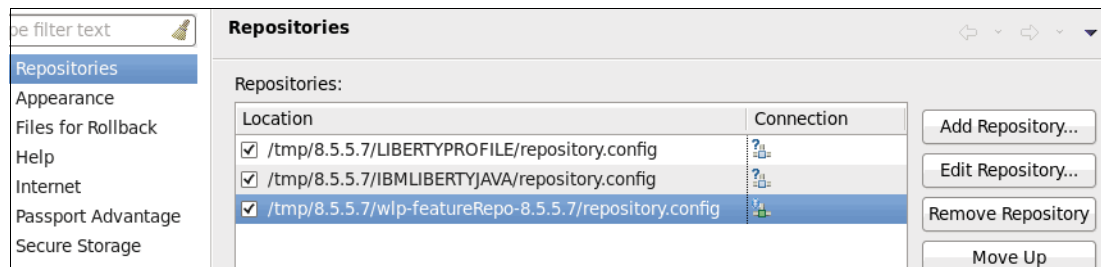


Figure 3-6 Add directory-based repositories to IBM Installation Manager repositories preference

This sample shows that you use an extracted wlp-featureRepo-8.5.5.7.zip downloaded from Fix Central to install features and addons for Liberty Version 8.5.5.7. In this way, you can install the newest features or addons at the Asset Selection wizard as illustrated in Figure 3-7, without connecting to the Internet.

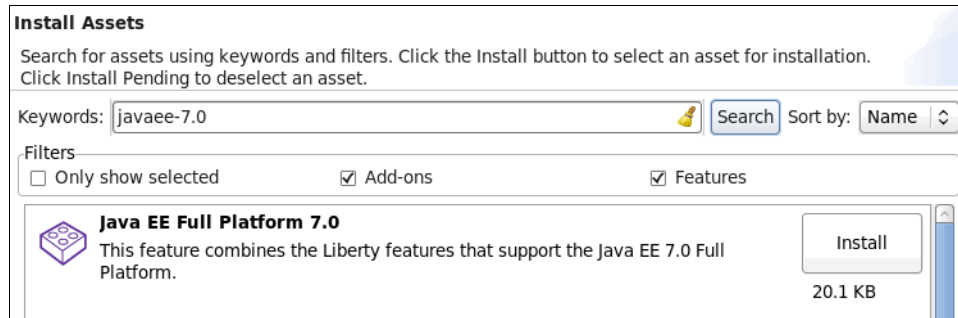


Figure 3-7 Javaee-7.0 feature from local directory-based repositories

3.4 Installation on z/OS

To install WebSphere Application Server Liberty for z/OS, follow these steps:

1. Obtain and create an IBM Installation Manager. See the following URL:
http://www.ibm.com/support/knowledgecenter/SS7K4U_8.5.5/com.ibm.websphere.wlp.z.series.doc/ae/twlp_ins_installation_zos_imkit.html
2. Obtain the product repositories.
3. Install WebSphere Application Server Liberty for z/OS.
 Optional: Install Liberty Repository features and addons.
4. Install IBM WebSphere SDK Java Technology Edition for Liberty.

3.4.1 Install WebSphere Application Server Liberty for z/OS

WebSphere Application Server Liberty for z/OS offerings is distributed as IBM Installation Manager repositories. The initial repository can be obtained by one of the following methods:

- ▶ SMP/E: Installing with SMP/E by using a ServicePac, SystemPac, or CBPDO
- ▶ Without SMP/E: Copying from the product physical media or ShopzSeries

New service levels can be installed by the following methods:

- ▶ If you use the SMP/E repository, you can apply program temporary fixes (PTFs) to add a single level of the service repository to the SME/E repository. Then, you can upgrade Liberty from the service repository by using IBM Installation Manager.
- ▶ Whichever you use, SMP/E repository or without SMP/E repository, you can download a service repository from Fix Central and upgrade Liberty from the downloaded service repository by using IBM Installation Manager.

To install Liberty, you use the `imc1` command with the following parameters:

<code><package name></code>	The package name to be installed
<code>-installationDirectory</code>	The directory where the package is installed
<code>-repositories</code>	The repository location

- sharedResourcesDirectory** A directory to store artifacts during installation (only specified the first time that you use this IBM Installation Manager to install a product)
- acceptLicense** Accepts the software license agreement

The following command is an example to install Liberty:

```
InstallationManager/bin/eclipse/tools/imcl install
com.ibm.websphere.liberty.zOS.v85 -installationDirectory
/usr/lpp/zWebSphere/Liberty/V8R5 -repositories ~/8.5.5.7/LIBERTYPROFILE
-acceptLicense
```

You can also follow the package name (and version) with a comma and a list of optional features separated by commas. The following features are available for WebSphere Application Server Liberty for z/OS. The keyword name for each feature is provided in parentheses:

- ▶ WebSphere Application Server Liberty (*liberty*)
 - This is a core feature of Liberty. It is always installed.
- ▶ Embeddable EJB container and JPA client (*embeddablecontainer*)
 - This option installs the embeddable EJB container and JPA client.

When you install a new copy of WebSphere Application Server Liberty for z/OS and do not specify the features to be installed, *embeddablecontainer* is installed by default. If you do not want to install *embeddablecontainer*, you can specify only *liberty* in the list.

3.4.2 Install Liberty Repository features and addons

If you want to install Liberty repository features with the installation by using IBM Installation Manager installation, specify the short name or symbolic names on the *user.feature* option of the **-properties** parameter. Multiple feature names are separated with double commas. The following example installs IBM z/OS Connect and WebSphere optimized local adapters for z/OS features:

```
InstallationManager/bin/eclipse/tools/imcl install
com.ibm.websphere.liberty.zOS.v85 -installationDirectory
/usr/lpp/zWebSphere/Liberty/V8R5 -repositories ~/8.5.5.7/LIBERTYPROFILE
-properties user.feature=zosConnect-1.0,,zosLocalAdapters-1.0 -acceptLicense
```

You can add the Extended Programming Model features by specifying the *user.addon* option:

```
InstallationManager/bin/eclipse/tools/imcl install
com.ibm.websphere.liberty.zOS.v85 -installationDirectory
/usr/lpp/zWebSphere/Liberty/V8R5 -repositories ~/8.5.5.7/LIBERTYPROFILE
-properties
user.feature=zosConnect-1.0,,zosLocalAdapters-1.0,user.addon=extendedPackage-1.0
-acceptLicense
```

You can also install addons and features from instances of the Liberty Asset Repository Service or local directory-based repositories. For more information about these asset repositories, see 3.6, “Installing content from Liberty Repository” on page 47. You can add the repository URL or directory on the **-repositories** parameter:

```
InstallationManager/bin/eclipse/tools/imcl install
com.ibm.websphere.liberty.zOS.v85 -installationDirectory
/usr/lpp/zWebSphere/Liberty/V8R5 -repositories
~/8.5.5.7/LIBERTYPROFILE,/u/mtres1/8.5.5.7/wlp-featureRepo-8.5.5.7 -properties
user.addon=zosBundle -acceptLicense
```

3.4.3 Install IBM WebSphere SDK Java Technology Edition for Liberty

You can install IBM WebSphere SDK Java Technology Edition Version 7.0, 7.1, or 8.0 for Liberty by using IBM Installation Manager.

The repository for Version 7.0 is part of WebSphere Application Server for z/OS Version 8.5 product. You can install the repository with SMP/E, download the installation files from IBM Fix Central, or install directly from the service repository. As for Version 7.1 and 8.0, you can download the installation files from IBM Fix Central, or install directly from the service repository. These are not available in SMP/E format.

The following command is an example to install Version 8.0:

```
InstallationManager/bin/eclipse/tools/imcl install
com.ibm.websphere.liberty.IBMJAVA.v80 -installationDirectory
/usr/lpp/zWebSphere/Liberty/V8R5 -repositories ~/8.5.5.7/IBMLIBERTYJAVA
-acceptLicense
```

You must set the **installationDirectory** parameter to the same location as the Liberty installation.

3.5 Considerations for upgrading Liberty V8.5.0 to V8.5.5

In WebSphere Application Server version 8.5.5, Liberty was promoted from an optionally installable feature to an independent offering. If your V8.5.0 installation has Liberty installed with IBM Installation Manager, a few extra steps are needed to update your system.

If you have only Liberty installed, or you have both WAS Classic and Liberty installed and you are going to upgrade only Liberty (not WAS Classic), take the following actions to update your system:

1. Install Liberty V8.5.5 offering for your edition.
2. Migrate your user data to the new installation. You can copy the `usr` folder to the new installation, or you can set the `WLP_USER_DIR` environment variable in `etc/server.env` to point to the `usr` directory of the original installation. These are the same steps typically used for an archive update.
3. Ensure that you have the required SDK installed.

If you have both WAS Classic and Liberty installed and you are going to upgrade both of them, take the following actions to update your installation:

1. In this case, you select to install both WAS Classic and Liberty offerings in IBM Installation Manager:
 - a. During the update flow, you receive a message advising you of the change in packaging. When you proceed, WAS Classic updates to V8.5.5.

Liberty feature is removed from WAS Classic offering, and the installed Liberty folder is backed up to `wlp.bak_<timestamp>` within the installation image. You can continue using this Liberty runtime from the backup folder, or you can move it or delete it.
 - b. Liberty will be installed in a different location as an independent offering from WAS Classic.
2. Migrate your user data to the new installation.
3. Ensure that you have the required SDK installed.

3.6 Installing content from Liberty Repository

Liberty Repository provides an online mechanism to deliver Liberty and additional content, enabling a single point of access for various asset types. Liberty Repository provides early access to supported new content, including new product capabilities, when they are delivered, rather than waiting for a new release.

Asset types that are available from Liberty Repository are as follows:

Addons	Artifacts that are packaged to add new capabilities over an existing Liberty installation.
Admin Scripts	Sample scripts for common Liberty administrative tasks.
Config Snippets	Samples of Liberty server configurations for specific tasks.
Features	Individual units of server functionality that can be installed in the Liberty runtime environment.
Open Source Integration	Artifacts that provide simple Liberty integration with commonly used open source projects.
Products	Simple archive installation packages of the Liberty server runtime environment.
Product Samples	Sample server applications that demonstrate the use of Liberty runtime capabilities.
Tools	Tools to enable development and testing of Liberty-based applications and runtime extensions.

In addition to accessing assets in the public, online Liberty Repository, you can create the following types of repositories to enable on-premises or offline access to Liberty Repository assets:

- ▶ “The Liberty Asset Repository Service”
- ▶ “Local directory-based repositories”

3.6.1 Installing assets by using the installUtility command

After you install Liberty, you can install addons, features, open source integrations, and product samples by running the `installUtility` command. The `installUtility` command automatically installs dependencies.

For example, the following command installs the `mongoDBSample`:

```
Liberty_Home/bin/installUtility install mongoDBSample
```

This command automatically installs a dependent feature, prerequisite library, and server with sample application as follows:

- ▶ `mongodb-2.0` feature to `lib/` and `lib/features/`
- ▶ Java MongoDB Driver library to `usr/shared/resources/MongoDBSampleLibs/`
- ▶ `mongoDBSample` server with sample application to `usr/servers/mongoDBSample/`

3.6.2 The IBM WebSphere Liberty Repository

This is a public, IBM hosted repository that is accessible through the Internet.

If you need to access the Liberty Repository through a firewall, ensure that you have access to the following hosts and ports:

- ▶ `public.dhe.ibm.com` on port 443
- ▶ `asset-websphere.ibm.com` on port 443

By default, the `installUtility` command is configured to install assets only from this repository. If you want to install assets from local or your own intranet location, see 3.6.3, “The Liberty Asset Repository Service” on page 48 and 3.6.4, “Local directory-based repositories” on page 50.

3.6.3 The Liberty Asset Repository Service

This open source service enables you to create an on-premises repository. It is remotely accessible behind the firewall. Also, if you develop a Liberty feature, it can be distributed remotely.

To get the latest information about LARS, see the `WASdev/tool.lars` repository on GitHub:

<https://github.com/WASdev/tool.lars>

Following is a quick way to get started with LARS:

1. Install LARS server:

a. Install Liberty:

```
java -jar wlp-runtime-8.5.5.7.jar
```

b. Install any LARS prerequisite features using `installUtility`:

```
Liberty_Home/bin/installUtility install cdi-1.0 servlet-3.0 mongodb-2.0  
jaxrs-1.1 cdi-1.0 servlet-3.0 mongodb-2.0 jaxrs-1.1
```

c. Download a self-extracting jar file installer from the following site:

https://developer.ibm.com/wasdev/downloads/#asset/tools-Liberty_Asset_Repository_Service

- d. Run `larsServerPackage.jar` to install prebuild LARS server into your Liberty environment:


```
java -jar larsServerPackage.jar
```
 - e. Install and set up MongoDB:

See the following installation guide about how to install MongoDB on your operating system:

<http://docs.mongodb.org/manual/installation>
 - f. Edit the `wlp/usr/servers/larsServer/server.xml` file to configure the LARS server. You need to configure the following by uncommenting:
 - i. Uncomment a commented out `<basicRegistry>` element and change a password for admin user. It is assumed 'adminpassword' here.
 - ii. Uncomment a commented out `<application-bnd>` element.
 - g. Start the LARS server:


```
Liberty_Home/bin/server run larsServer
```
2. Install the LARS client:
 - a. Download the package file from the following site:

https://developer.ibm.com/wasdev/downloads/#asset/tools-Liberty_Asset_Repository_Service_Client
 - b. Unpack `larsClient.zip` to any directory:


```
unzip larsClient.zip
```
 - c. To verify, issue the **larsClient** command:


```
bin/larsClient help
```
 3. To upload a feature:

You can add a feature to the LARS repository by specifying the following command:

```
bin/larsClient upload --url=http://localhost:9080/ma/v1 --username=admin --password=adminpassword my_feature.esa
```
 4. Configure Liberty to use the LARS repository:

If you create a property file `Liberty_Home/etc/repositories.properties` as shown in Example 3-2, you can install a registered feature from the LARS repository.

Example 3-2 repositories.properties for LARS

```
lars.url=http://localhost:9080/ma/v1
useDefaultRepository=false
```

To disable access to the public IBM WebSphere Liberty Repository, set the `useDefaultRepository` property to `false` as shown in Example 3-2. The public repository is enabled by default.

5. Install a feature from the LARS repository by using **installUtility**:

You can find, install, and download a feature from the LARS repository. Following is an example of finding the feature that you uploaded by using the **larsClient** command in Step 3 on page 49:

```
Liberty_Home/bin/installUtility find my_feature
```

Example 3-3 installUtility find output

```
Establishing a connection to the configured repositories...  
This process might take several minutes to complete.
```

```
Successfully connected to all configured repositories.
```

```
Searching assets. This process might take several minutes to complete.
```

```
feature : my_feature : my_feature
```

You can find the feature if the feature is appropriate for your Liberty version.

3.6.4 Local directory-based repositories

This type of repository can be created by using the **installUtility** download action or downloaded from IBM Fix Central:

► Create by using **installUtility**

You can download assets to your local file system by running the **installUtility** command. After you download assets to your local file system, you can add a local directory to your repository configuration so that you can install assets from the directory.

► Download from IBM Fix Central

As an alternative to downloading individual assets, you can download and extract a `wlp-featureRepo-<version>.zip` file from IBM Fix Central. The `.zip` file contains all features and add-ons for the particular fix pack, with the same structure that was created using **installUtility**.

For example, the following command downloads the `adminCenter-1.0` feature to the `c:\temp\download` directory:

```
Liberty_Home/bin/installUtility download adminCenter-1.0  
--location=c:\temp\download
```

The following directory structure is created, and the related features are put in the directory.

Example 3-4 Directory-based repositories structure

```
c:\temp\download\  
  repository.config  
  features\  
    8.5.5.7\  
      com.ibm.websphere.appserver.adminCenter-1.0.esa  
      :  
      com.ibm.websphere.appserver.adminSecurity-1.0.esa  
      :
```

You can download additional assets on the same existing directory.

If you create a property file, *Liberty_Home/etc/repositories.properties* as shown in Example 3-5, you can install downloaded assets from the local directory.

Example 3-5 repositories.properties for directory-based repositories

```
local-rep2.url=file:///c:/tmp/download  
useDefaultRepository=false
```

To disable access to the public IBM WebSphere Liberty Repository, set the `useDefaultRepository` property to `false` as shown in Example 3-5. The public repository is enabled by default.

3.7 Updating Liberty

There are two approaches to update the Liberty runtime:

- ▶ In-place update

The new version of the product is installed into the directory that it is currently installed in. The old version is overwritten.

- ▶ Side-by-side update

The new version of the product is installed as another instance. The old version remains in-place.

In addition, the *rip-and-replace* approach is described in 5.2, “Flexible deployment” on page 73.

Liberty has two methods for installing the runtime environment: Either by using the downloaded archive file or by using IBM Installation Manager. Archive file installation can only use the side-by-side update approach, and the IBM Installation Manager installation can use either approach (in-place or side-by-side).

There are three types of resources to consider when you update Liberty:

- ▶ Product files

Liberty product files that are in `bin`, `clients`, `dev`, `lafiles`, `lib`, and `templates` directories. Also, Java product files in `java` directory are included.

- ▶ User files

Files that are in the user directory include server configuration files, applications, and shared resources.

- ▶ User output files

Files that are generated by the server, for example, log files and temporary disk storage.

Fix pack instructions: All fix packs that update your environment contain detailed instructions. Always refer to these instructions before applying the update.

3.7.1 In-place update

This method overwrites the product files with the new version of the files. IBM Installation Manager uses this as the default method. Figure 3-8 shows an example of updating a Liberty product file.

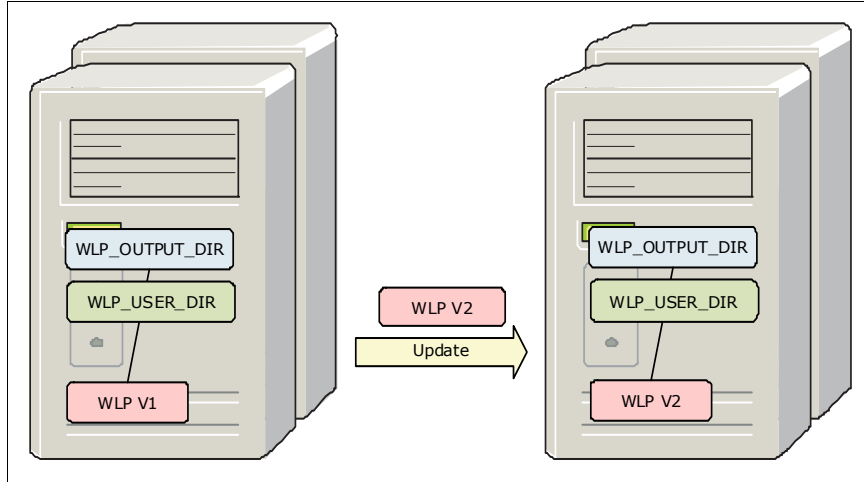


Figure 3-8 Updating Liberty by in-place update

Note: If you choose the in-place update method to update the Liberty product files (also Java runtime), changes affect all servers that use those product files. You must stop all servers on this installation during the Liberty update.

If this drawback is not acceptable for you, consider using the side-by-side update method described in 3.7.2, “Side-by-side update” on page 52.

3.7.2 Side-by-side update

If you used a JAR or ZIP archive-based installation, you must use this approach to update Liberty. You can also use this approach to an IBM Installation Manager installation by a procedure in which you install another copy of the Liberty product as a new group.

This approach uses the following process:

1. Install a new version of the product files into a location that is different from the previous installation.
2. Create the `etc/server.env` file in the new Liberty installed location. Set the `WLP_USER_DIR` environment variable to locate the user directory from the previous environment.
3. Stop the old version of the Liberty server.
4. Start the new version of the Liberty server.

Figure 3-9 illustrates the continued use of the user files and user output directory by setting the `WLP_USER_DIR` environment value on `etc/server.env`.

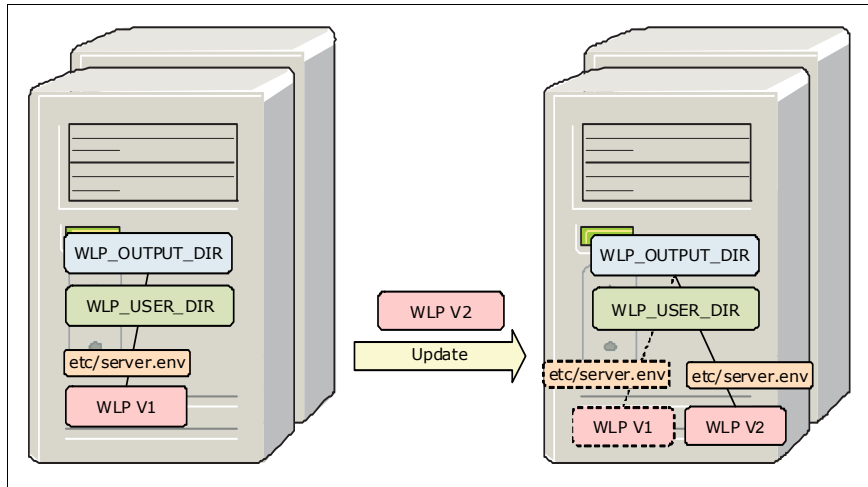


Figure 3-9 Updating Liberty by using the side-by-side update

In this procedure, the old product files are not overwritten. Therefore, you can easily do a rollback by stopping the new version of the Liberty server and starting the old one.

Note: The side-by-side approach, using `etc/server.env` and `WLP_USER_DIR` environment variable for separating product files and user files, is the suggested practice.



Working with Liberty profile servers

Liberty profile servers are defined by using configuration files. This chapter provides information about configuring Liberty profile servers.

In this chapter, the following topics are discussed:

- ▶ Working with the bootstrap.properties file
- ▶ Working with the server.xml file
- ▶ Using WebSphere developer tools to work with the configuration
- ▶ Liberty command-line utilities
- ▶ Use the configuration dropins folder to specify server configuration
- ▶ Configuring dynamic application updates
- ▶ Starting and stopping the server using the command line
- ▶ Classloaders and shared libraries

4.1 Working with the bootstrap.properties file

The `bootstrap.properties` file initializes the runtime environment for a particular server. Generally, it contains attributes that affect the initialization of the runtime core. This file is not required and by default it is not created. If the file is needed, it is created in `${server.config.dir}`. Changes to this file require that the server be restarted. To configure the `bootstrap.properties` file, use *key-value* pairs. To make comments, use the `#` sign.

Example 4-1 Sample content of the bootstrap.properties file

```
# trace logging settings
com.ibm.ws.logging.trace.specification = *=all=enabled
```

The *name-value* pairs are available to `server.xml` (see 4.2, “Working with the `server.xml` file” on page 56). For more information about how these *name-value* pairs can be used from within `server.xml`, see the following site:

https://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/twlp_inst_bootstrap.html?lang=en

4.2 Working with the server.xml file

Each Liberty profile is configured by using a `server.xml` configuration file. The only required entry in the `server.xml` file is the *server* tag, which defines a server configuration scope. Example 4-2 gives an example of the basic `server.xml` file.

Example 4-2 The simplest configuration of the Liberty profile server

```
<server> <server/>
```

4.2.1 Adding new configuration options

The configuration in Example 4-2 enables you to start the Liberty profile server, but more settings must be added to run real applications. Example 4-3 gives you a view of what a typical configuration looks like.

Example 4-3 Sample configuration for Liberty profile server

```
<server description="server2">

  <featureManager>
    <feature>jsp-2.3</feature>
  </featureManager>

  <httpEndpoint host="localhost" httpPort="9081" httpsPort="9444"
    id="defaultHttpEndpoint"/>

</server>
```

One of the most important sections in the configuration file is the *feature* section. This is the place where you configure the server runtime to use the required features for your applications. For example, if your application uses only servlets and JavaServer Pages (JSPs), the feature configured in Example 4-3 is enough.

Adding a new feature is as simple as adding a new `<feature>` element to the `<featureManager>` element. After the file is saved, the file monitor service discovers the change, and the Liberty profile server applies the new features to the runtime.

4.2.2 Using include syntax

The `server.xml` file also allows you to configure other parameters, such as the server listening ports and data sources. Although keeping all of the configuration settings for a server in a single file eases the complexity of server configuration, the file can grow to a substantial size. This is one reason to use the *include* syntax to move some of the configuration into other XML files. Furthermore, the included XML files can also include other configuration files. Figure 4-1 illustrates an example of such a case.

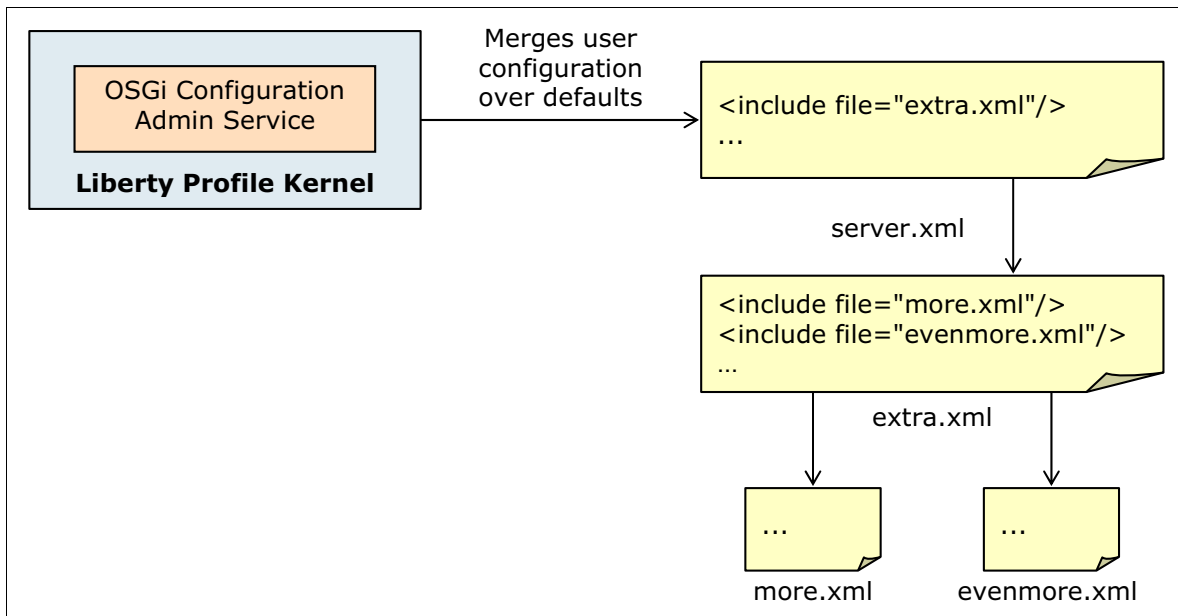


Figure 4-1 Splitting the configuration of the Liberty profile into multiple files

The *include* syntax provides a flexible and powerful way to share all or part of a configuration between different servers on the same or even different host machines. You can control how the configuration is structured and which pieces are shared by which servers. Included XML files can be on the local file system or hosted in the network. The monitor service detects changes to the `server.xml` file and any of the included files.

By using the *include* tag, you can rewrite Example 4-3 on page 56 with the configuration given in Example 4-4 to add a data source definition.

Example 4-4 Rewritten server.xml configuration file using the include tag

```
<server description="server2 main configuration">
  <featureManager>
    <feature>jsp-2.2</feature>
  </featureManager>

  <httpEndpoint host="localhost" httpPort="9080" httpsPort="9443"
    id="defaultHttpEndpoint"/>

  <!-- Configuration of external resources -->
  <include location="datasourcesConfig.xml"/>

</server>
```

The configuration now includes an additional `datasourcesConfig.xml` file. The *location* is a direct path. If you use just the name of the file, both the source and the included file have to be in the same directory. Example 4-5 shows the content of the new `datasourcesConfig.xml` file. Notice that the included file also has to contain the *server* tags.

Example 4-5 Configuration of an additional `datasourcesConfig.xml` configuration

```
<server description="database configuration for server2">
  <featureManager>
    <feature>jdbc-4.0</feature>
  </featureManager>

  <jdbcDriver id="DerbyEmbedded" libraryRef="DerbyLib"/>
  <library filesetRef="DerbyFileset" id="DerbyLib"/>
  <fileset dir="C:/Derby/lib" id="DerbyFileset" includes="derby.jar"/>
    <dataSource id="MyDataSource" jdbcDriverRef="DerbyEmbedded"
      jndiName="jdbc/MyDataSource"
      syncQueryTimeoutWithTransactionTimeout="false"
      type="javax.sql.DataSource">
      <properties.derby.embedded createDatabase="create" databaseName="CUSTDB1"/>
    </dataSource>
  </server>
```

The included configuration contains the `jdbc-4.0` feature along with the data source definition, which is merged by the OSGi Configuration Admin Service as a single configuration.

Alternatively, instead of pointing to a file on a file share, you can use a URL under which it is accessible, for example:

```
<include location="http://myfileserver/config/server2/datasourcesConfig.xml"/>
```

Note: It is possible to use the `include` syntax to include these files directly from a version management system. See the documentation for your specific version of the management system for details about how files can be delivered based on URL access.

4.2.3 Using variables in configuration files

Variables can be used in the configuration of a WAS Liberty server to avoid hardcoding values that might change as the server is reused in different environments. Variables can be defined in either the server configuration file or in the bootstrap properties file. Changes in the server configuration file do not require a server restart to take effect. Changes made in the `bootstrap.properties` file do require a server restart for the changes to take effect.

It is recommended that variables for a particular server, such as port numbers, be specified in the bootstrap properties file, allowing the `server.xml` file to be shared across multiple servers. Values that are shared across servers are better defined in a shared xml file that can be included in the `server.xml` of a particular server.

There are a number of predefined variables that can be referenced. These are:

- ▶ JVM system properties
- ▶ Process environment variables
- ▶ Directory properties defined by the WAS Liberty environment. Following are the key variables:
 - `wlp.install.dir`: Root of the WAS Liberty installation
 - `wlp.user.dir`: The `usr` directory under the `wlp.install.dir`
 - `server.config.dir`: The specific server directory under the `wlp.user.dir/servers`

More information about the directory structure and available properties can be found in the IBM Knowledge Center at:

https://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/rwlp_dirs.html?lang=en

Defining variables

Variables can be defined in a number of ways. How they are defined determines the scope of the variable. To define a variable in the `bootstrap.properties` file, the variables are entered as a key-value pair as shown in Example 4-6.

Example 4-6 Defining a variable in the bootstrap.properties file

```
HTTP_default_var=8006
```

Variables defined in this manner are global in scope.

Variables can be defined in the server configuration files by using the `<variable>` tag. Variables defined in this manner are also global in scope. Note that if variables are defined both in included files and the `server.xml` file, those defined in the `server.xml` take precedence. Example 4-7 shows how to define the `HTTP_Default_var` in the `server.xml` file.

Example 4-7 Defining variables using the <variable> tag

```
<variable name="HTTP_Default_var" value="8007" />
```

Variables can also be defined within the configuration files with a specific scope, that being the scope of the configuration elements to which they belong. Another way to define the `HTTP_Default_var` that we previously defined in our bootstrap properties is as shown in Example 4-8.

Example 4-8 Defining a variable scoped to its containing configuration element

```
<httpEndpoint id="defaultHttpEndpoint" HTTP_Default_var="8008" .....
```

The order of precedence of the variable declarations is that the `<variable>` declaration overrides the `bootstrap.properties` declaration, which overrides the scoped declaration.

Variables substitution

The variable substitution syntax is `${variable name}`. An example of how to use the value of the `HTTP_Default_var` within a configuration file is shown in Example 4-9.

Example 4-9 Using variable within a configuration script

```
<httpEndPoint id="defaultHttpEndpoint" HTTP_Default_var="8008"
  host="*"
  httpPort="${HTTP_Default_var}" />
```

To use process environment variables, the syntax used for variable substitution is `$(env.variable name)`. If the `HTTP_Default_var` was defined as a process variable, the substitution syntax is as shown in Example 4-10.

Example 4-10 Using a process environment variable in a configuration script

```
<httpEndPoint id="defaultHttpEndpoint" HTTP_Default_var="8008"
  host="*"
  httpPort="$(env.HTTP_Default_var)" />
```

4.2.4 Encrypting passwords

WAS Liberty profile configuration, including any passwords, is kept in text files. To improve security, WAS Liberty provides the `securityUtility` that supports plain text encryption and SSL certificate creation.

To encode a password, `securityUtility` is used with the syntax defined in Example 4-11.

Example 4-11 Syntax for the securityUtility to encode a password

```
$ securityUtility encode --encoding=encoding_type --key=encryption_key --notrim
text_to_encode
```

The encoding type can be any one of the following:

- ▶ Exclusive or (XOR); this is the default
- ▶ Hash
- ▶ Advanced Encryption Standard (AES)

The encryption key is used when encoding using AES encryption. When using AES encryption, the encryption key used for decrypting can be overridden from the default by setting the `wlp.password.encryption.key` property. This property should not be set in the `server.xml` file that stores the password, but in a separate configuration file that is included by the `server.xml` file. This separate configuration file should contain only a single property declaration, and should be stored outside the normal configuration directory for the server.

The `--notrim` parameter specifies whether space characters are removed from the beginning and end of the text to encode. If no parameters are given, the command works in interactive mode.

For more information about encryption syntax, see the Liberty profile Security topic in the IBM Knowledge Center at this website:

https://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/rwlp_command_securityutil.html?lang=en

4.3 Using WebSphere developer tools to work with the configuration

You can use the WebSphere developer tools to work with the `server.xml` file. Wizards and windows in the Eclipse workbench help to configure all of the server properties. To configure the `server.xml` file using WebSphere developer tools, double-click **Server Configuration**, as illustrated in Figure 4-2.

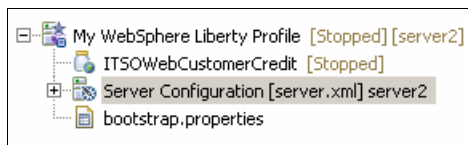


Figure 4-2 Configuring a Liberty profile server from the WebSphere developer tools

The current server configuration is shown in a new window with all of its features and properties listed, as illustrated in Figure 4-3.

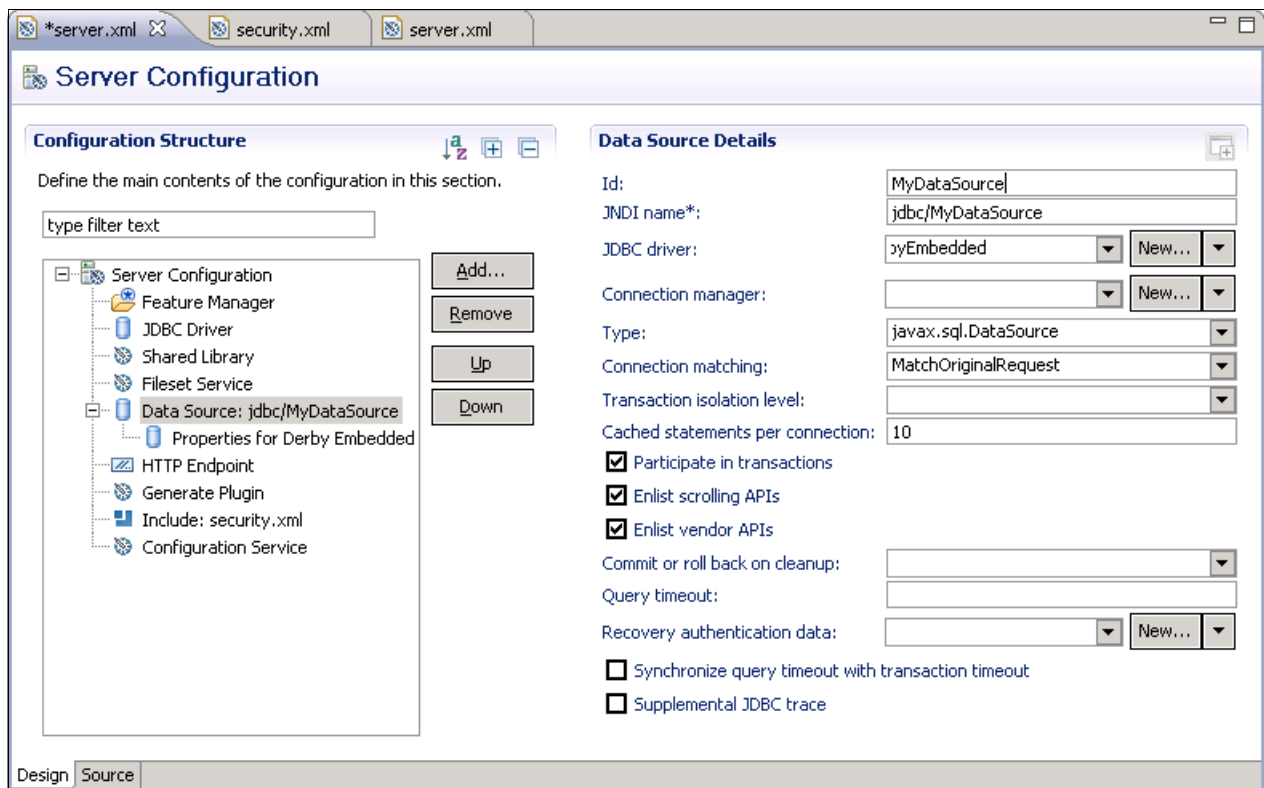


Figure 4-3 Configuring a Liberty profile server data source using the WebSphere developer tools

Instead of typing the name of each of the configuration elements or their numerous properties, you can simply click **Add**, and the WebSphere developer tools guide you with the possible values and properties to configure, as shown in Figure 4-4.

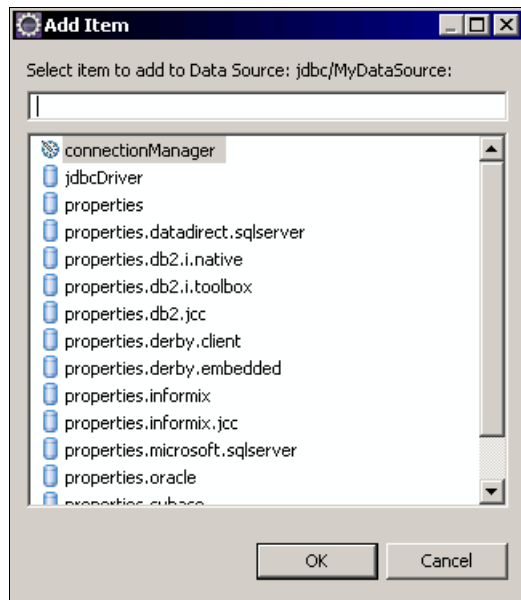


Figure 4-4 Available properties for the Liberty data source using WebSphere developer tools

4.4 Liberty command-line utilities

This section provides information about some of the command-line tools that are available within a Liberty installation. The utilities covered are not exhaustive for details. For the complete set of supported utilities, refer to the IBM Knowledge Center here:

https://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/twlp_admin_script.html?lang=en

4.4.1 Packaging a Liberty server

From the command line, it is possible to create a package file that contains the Liberty runtime, the files in the shared resources directory, a specific server, and the applications that are embedded in the server. This package can be used to deploy to hosts in a Liberty collective, distribute it to colleagues, or even embed it in a product distribution. The server installation that you want to package cannot already be joined to a collective. The server to be packaged should not be running.

To package the server, the `server` command is used from the `${wlp.install.dir}` directory. The format of the package command is shown in Example 4-12.

Example 4-12 Use the server command to create a server package

```
$ server package myServer --archive=MyServer_Package.zip --include=all
```

MyServer is the name of the Liberty profile server and is mandatory. The variable `--archive` specifies the archive to package the server into. It is optional and, if not specified, the archive is created as `<server name>.zip` into the `${server.config.dir}` of the server being packaged. The `--include` variable is optional and supports the following values:

- ▶ The `all` value packages the runtime binary files and the relevant files in the `WLP_USER_DIR` directory.
- ▶ The `usr` value packages only relevant files in the `WLP_USER_DIR` directory and excludes the runtime binary files.
- ▶ The `minify` value packages only those parts of the runtime environment and the files in the `WLP_USR_DIR` that are required to run the server. This option significantly reduces the size of the resulting archive because it only includes those aspects of the runtime needed to support the features defined for the server.

4.4.2 Installing config snippets with the configUtility

The `configUtility` provides the capability to download configuration snippets from the IBM WebSphere Liberty Repository. The Liberty repository configuration snippets are samples of Liberty server configurations for specific tasks. The utility provides the capability to replace configuration variables with the user's own input values. To use the `configUtility`, the Liberty repository needs to be set up as documented in 3.6, "Installing content from Liberty Repository" on page 47.

To find a configuration snippet, the `configUtility` is invoked with the `find` action and the item to search for, as shown Example 4-13, where the search is for a configuration snippet that contains `Remote`.

Example 4-13 Using the configUtility to find a config snippet

```
$ configUtility find Remote
Retrieving snippets that are related to "remote".
```

```
remoteAdministration
$
```

The `find` operation in Example 4-13 shows that there is one snippet in the repository that contains the search string. The `configUtility` can be used to install the configuration snippet by using the `find` action and optionally the `--createConfigFile` parameter as shown in Example 4-14.

Example 4-14 Using the config utility to install the config snippet

```
$ configUtility install remoteAdministration
--createConfigFile=/home/itsousr/myConfigSnippet
Downloading the requested configuration snippet...
```

```
<include location="/home/itsousr/myConfigSnippet" />
```

Please ensure administrative security is configured for the server.

The snippet is downloaded to the local file system. The sample snippet is shown in Example 4-15.

Example 4-15 Sample remoteAdministration config snippet

```
<server>

  <!-- NOTE: This file is for reference only. -->

  <!-- Enable restConnector-1.0 feature -->
  <featureManager>
    <feature>restConnector-1.0</feature>
  </featureManager>

  <!-- Simple administrative security configuration. -->
  <!-- TODO: Set the security configuration for Administrative access -->
  <quickStartSecurity userName="{adminUser}" userPassword="{adminPassword}"/>

  <!-- TODO: Set the SSL keystore password -->
  <keyStore id="defaultKeyStore" password="{keystorePassword}"/>

  <!-- TODO: Set HTTP Endpoint attributes -->
  <httpEndpoint id="defaultHttpEndpoint"
    host="*"
    httpPort="9080"
    httpsPort="9443"/>

  <!-- TODO: Use readDir and writeDir to specify directories that remote
  clients are allowed to have read and write access. There can be
  multiple readDir and writeDir elements. Replace writePath and readPath
  variables with your choice of locations or remove them if not needed. -->
  <remoteFileAccess>
    <writeDir>${writePath}</writeDir>
    <readDir>${readPath}</readDir>
  </remoteFileAccess>

</server>
```

The configuration snippet in Example 4-15 contains a number of variables that could be replaced with user-specific values by using the **--v parameter** as shown in Example 4-16.

Example 4-16 Using the config utility to replace variables with user-defined values

```
configUtility install remoteAdministration --vadminUser=itsousr
--vadminPassword=secret --vkeystorePassword=secret --vwritePath=/home/itsousr
--vreadPath=/home/itsousr
```

More information about this utility can be found in the IBM Knowledge Center:

https://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/rwlp_command_configutil.html?lang=en

4.4.3 Application client commands

Liberty supports Java applications in the application client container, which can access application components (for example, Enterprise JavaBeans (EJB)) that are running on a Liberty server. The application client container capability is available as part of the `javaeeClient-7.0`. If this feature is not available in the Liberty installation, it can be installed by using the `installUtility` as shown in Example 4-17.

Example 4-17 Installing the application client capability

```
$ installUtility install javaeeClient-7.0
```

As part of this installation, the `client` command is made available in the `${wlp.install.dir}/bin` directory. The `client` command can be used to create, run, debug, and package application clients. New clients are created using the syntax shown in Example 4-18.

Example 4-18 Creating an application client

```
$ client create MyClient
client MyClient created.
$
```

The application client is created in the `${wlp.install.dir}/usr/clients` directory. Like Liberty servers, application clients contain a single point of configuration: the `client.xml` file, which is available in the client directory `${wlp.install.dir}/usr/clientsMyClient`. See Example 4-18.

The generated `client.xml` includes the `javaeeClient-7.0` feature as shown in Example 4-19.

Example 4-19 Default application client configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<client description="new client">
  <featureManager>
    <feature>javaeeClient-7.0</feature>
  </featureManager>
</client>
```

A client application needs to be added to the client. Details about how to create or add a client application can be found in the IBM Redbooks publication: *IBM WebSphere Application Server Liberty Profile Guide for Developers*, SG24-8076:

<http://www.redbooks.ibm.com/abstracts/sg248076.html?Open>

The application client can be run by using the `client run` command as shown in Example 4-20.

Example 4-20 Running a client

```
$ client run MyClient
```

For more information about the application client capability, see the IBM Knowledge Center: https://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/twlp_setup_prepareappclient.html?lang=en

4.5 Use the configuration dropins folder to specify server configuration

Server configurations for Liberty servers can be created by creating configuration dropin files and placing these in the `configDropins` directory. The dropins can be used to either set a default configuration or override an existing configuration for a Liberty server standard configuration as defined in the `server.xml` and included files.

The `configDropins` directory is created under the `${server.config.dir}` directory, and it contains two subdirectories: `defaults` and `overrides`.

If an administrator wants to be able to override the configuration of a specific `server.xml`, a dropin file is created and put in the `${server.config.dir}/configDropins/overrides` directory. A server can be forced to use a specific port by using the `override.xml` file shown in Example 4-21 and placing this file in the `overrides` directory for a server.

Example 4-21 Override.xml to override the HTTP_Default_var variable

```
<server>
  <variable name="HTTP_Default_var" value="9083"/>
</server>
```

The server does not need to be restarted for this to take effect. The current listening port for the server will be changed to that defined in the `override.xml` file.

If an administrator wants to be able to provide defaults for a particular server instance, a file `default.xml`, with the same content as the `overrides.xml`, can be placed in the `${server.config.dir}/configDropins/defaults` directory. In our example, if no variable was set for the `HTTP_Default_var`, the value specified in `defaults.xml` would be used instead.

The configuration for the `configDropins` directory can be managed through the `<config>` element in `server.xml`. The ability to monitor changes to the configuration can be controlled through the `config` tag as shown in Example 4-22.

Example 4-22 Disable the monitoring for config updates

```
<config updateTrigger="disabled"/>
```

Other aspects of monitoring, such as the monitor interval and action to be taken on error, can be configured through the `config` tag as shown in Example 4-23.

Example 4-23 Modifying other aspects of the config monitor

```
<config updateTrigger="polled" monitorInterval="10m30s" onError="IGNORE"/>
```

4.6 Configuring dynamic application updates

The Liberty servers can be configured to be notified of dynamic updates, adding, updating, and removing applications. For all deployed applications, `server.xml` can be configured to specify whether application monitoring is enabled and how often to check for updates to applications. You can also change the default `dropins` directory.

To configure to only look for updates that are done through the Java Management Extension (JMX) managed beans (MBeans) and to disable the dropins directory, add the configuration as shown in Example 4-24.

Example 4-24 Change the application monitor to look for updates from the JMX MBeans

```
<applicationMonitor updateTrigger="mbean" dropinsEnabled="false"/>
```

The FileNotificationMBean can be used to notify the server which configuration file or files need to be dynamically reprocessed.

To allow only updates to the application files, you can use the polled trigger values, and the applicationMonitor element should be changed as in Example 4-25.

Example 4-25 Change the polling rate and monitored application dropins directory

```
<applicationMonitor updateTrigger="polled" pollingRate="5000ms"
dropins=${server.config.dir}/myDropins"/>
```

In this example, the pollingRate value specifies that the application monitor polls for changes every 5 seconds instead of the default value of half of a second. The default dropins directory was changed to monitor the `${server.config.dir}/myDropins` directory.

To disable all the monitoring of application changes, the monitorApplication element should be changed as shown in Example 4-26.

Example 4-26 Disable the monitoring of application changes

```
<applicationMonitor updateTrigger="disabled"/>
```

For more configuration monitoring examples, see the following IBM Knowledge Center website:

https://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/twlp_setup_dyn_upd.html?lang=en

4.7 Starting and stopping the server using the command line

The server can be controlled from a single **server** command in the `WLP_INSTALL_DIR/bin` directory. The following commands can be used to start and stop the server:

- ▶ `server start <servername>`
Starts the server running in the background.
- ▶ `server run <servername>`
Starts the server running in the foreground and writes the console output to the window. To stop the Liberty profile server in this mode, press Ctrl+c or kill its process or run `server stop` from another command window.
- ▶ `server debug <servername>`
Starts the server in debug mode.
- ▶ `server stop <servername>`
Stops the server.
- ▶ `server status <servername>`
Displays the current state of the server.

4.8 Classloaders and shared libraries

The Liberty classloaders hide the classes that make up the Liberty runtime from Java Enterprise Edition applications and only exposes the public API. This feature isolates the applications from the internal libraries used by Liberty. For example, one of the Open Source libraries that Liberty uses is the SLF4J version 1.5.6 library. If an application needs to use a different version of the same library, Liberty runtime copy does not interfere with the application's copy. This resolves the problem of accidental usage of the internal server libraries.

The following types of libraries can be configured with classloaders:

- ▶ `ibm-api` (IBM APIs)
- ▶ `spec` (standardized APIs)
- ▶ `third-party`

The default behavior of Liberty is to hide the third-party APIs. Example 4-27 shows a configuration of a sample application where only standardized APIs are available, and all IBM or third-party APIs are hidden.

Example 4-27 Configuration of the application classloader libraries visibility

```
<application location="SampleApplication.war" type="war">
  <classloader apiTypeVisibility="spec" />
</application>
```

The Liberty classloaders allow for sharing the class instances from libraries between applications or by using them privately within the scope of an application. These two models are called the *common library* and the *private library*. Example 4-28 shows a sample configuration of both libraries for the same sample application.

Example 4-28 Configuration of common and private libraries

```
<application location="SampleApplication.war" type="war">
  <classloader commonLibraryRef="mySharedLib" privateLibraryRef="myITSOLib" />
</application>

<library id="mySharedLib">
  <fileset dir="${server.config.dir}/mySharedLib" includes="*.jar"/>
</library>

<library id="myITSOLib">
  <fileset dir="${server.config.dir}/ITSOLib" includes="*.jar"/>
</library>
```

The listing defines two shared libraries: `mySharedLib` and `myITSOLib`. Class instances from libraries in the `mySharedLib` can be shared with other applications that include `mySharedLib` as a `commonLibraryRef`.

Instances of the `myITSOLib` are dedicated to the applications that reference them. There can be as many copies of `myITSOLib` as the applications that need those resources.

Liberty supports the older method of overriding the default behavior of the classloader configuration that uses the profile runtime classes. This configuration can be done by using the *delegation* attribute. The default value of this attribute is *parentFirst*. If you use *parentLast*, the specified shared classes are loaded with the application, overriding the server libraries. Example 4-29 presents this configuration.

Example 4-29 Configuration of delegation mode in the Liberty classloader for the sample application

```
<application location="SampleApplication.war" type="war">
  <classloader commonLibraryRef="mySharedLib" delegation="parentLast" />
</application>
```

To learn more about the class loaders, see the following IBM Knowledge Center website:

https://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/twlp_classloader.html?lang=en



Administering the WebSphere Liberty profile

This chapter describes the Liberty environment and how to use the various Liberty tools to administer the environment.

This chapter covers the following topics:

- ▶ Installing the sample environment
- ▶ Flexible deployment
- ▶ The Liberty Management API
- ▶ Liberty collectives

5.1 Installing the sample environment

In this chapter, a number of examples are created showing how collections of Liberty servers can be managed through the various administrative tools that are available.

5.1.1 The Liberty environment

A sample Liberty installation is created to exercise the examples in this chapter. Install the Liberty environment as described in section 3.2.2, “Installation by extracting a ZIP archive file” on page 40. The .zip file that is used for the examples in this section is `wlp-webProfile7-java8-linux-x86_64-8.5.5.7.zip`. All the examples are shown from a Linux perspective.

To support console scripting, the following environment variables are used in the examples performed in this chapter:

- ▶ `WLP_INSTALL_DIR` is equivalent to the Liberty variable `${wlp.install.dir}` and points to the installation directory of Liberty. For example, Liberty is installed in `/home/itsouser/wlp` so the `WLP_INSTALL_DIR = /home/itsouser/wlp`
- ▶ `SERVER_CONFIG_DIR` is equivalent to the Liberty variable `$(server.conf.dir)` and points to the server configuration directory under `$WS_INSTALL_DIR/usr/servers`. For example, server controller1 is `SERVER_CONFIG_DIR = $WS_INSTALL_DIR/usr/servers/controller1`

5.1.2 Installing Jython

Liberty supports scripting in any language though it is recommended that WebSphere Administrators use a Java scripting language (Jython, JRuby, Groovy, and so on). In the examples that follow, Jython is used. Liberty ships with a Jython library, `restConnector.py`, which provides a Jython interface to the Representational State Transfer (REST) Java Management Extensions (JMX) connector of Liberty. This connector has been tested with Jython and is compatible with versions 2.5.4 and higher. Liberty does not ship with a Jython run time so the user needs to download and configure the run time for themselves.

To use the Jython scripting capabilities, the Jython environment needs to be set up. Use the following steps to complete that process:

1. Set the **CLASSPATH** environment variable to include the `restConnector.jar` file from the `${wlp.install.dir}/clients` directory, as shown in Example 5-1.

Example 5-1 Add the restConnector.jar to the CLASSPATH

```
export CLASSPATH=$CLASSPATH:${wlp.install.dir}/clients/restConnector.jar
```

2. Set up the **JYTHONPATH** environment variable to include the `restConnector.py` file from the `${wlp.install.dir}/clients/jython` directory, as shown in Example 5-2.

Example 5-2 Add the restConnector.py to the JYTHONPATH

```
export JYHTONPATH=$JYTHONPATH:${wlp.install.dir}/clients/jython/restConnector.py
```

Notes:

- ▶ On some versions of Jython, the `JYTHONPATH` environment variable is not respected. In such cases, the `restConnector.py` can be copied to the `Lib` directory of the Jython installation.
- ▶ All the examples in this section use `localhost` as the host name. This is not recommended for production use. In fact, the host name used in a production environment should always be the fully qualified domain name. A host can be registered with the collective under different names. It is important that the host name specified for the collective `registerHost`, `updateHost`, and `unregisterHost` be consistent with the host name that is used for the registered collective members. The `defaultHostName` attribute, in the server member's `server.xml`, controls the host name to which the server considers itself to belong.

5.2 Flexible deployment

The flexible nature of Liberty has led to a different predominant deployment pattern, that of *rip-and-replace*. The complete, configured stack is generated as part of a DevOps flow, and is completely replaced with each update. Liberty has a utility to *package* a server. This utility operates on a configured server, with applications installed, and produces a `.zip` file. The `.zip` file contains the applications, user configuration and resources, and, optionally, the run time (product binary files) required by that server's configuration.

This customized, configured package can then be quickly deployed to a host machine through file transfer and unpackaged. These packages guarantee clean, identical clones on each host. The Liberty server packages are ideal for use in DevOps flows, where the server package is the output of the build. Tools like uDeploy, Chef, and the Liberty profile itself can be used to distribute and unpack that package.

After unpacking, overrides can be applied to individual hosts: Configuration variables can be set for values such as port numbers, and overrides to the packaged configuration can be enforced through use of the *config overrides* directory locations. This rip-and-replace approach to deploying prepackaged applications is shown in Figure 5-1.

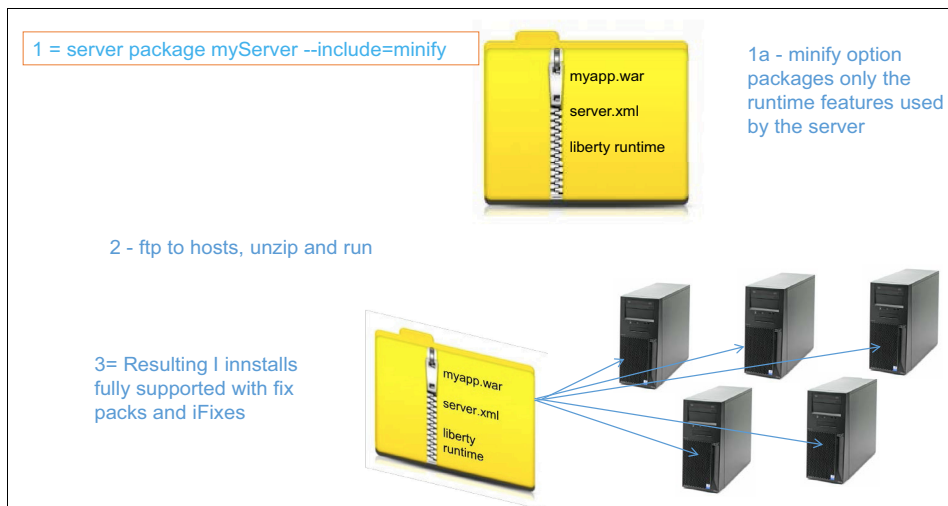


Figure 5-1 Flexible deployment model

5.3 The Liberty Management API

The management API of Liberty is based on the use of JMX. JMX is a framework that provides a standard way of exposing Java resources, in this case the Liberty servers, to JMX-enabled clients. An increasing number of Representational State Transfer (REST)-based interfaces are being created to manage Liberty in addition to the JMX interfaces. There are some new features, such as Batch, that provide only a REST interface and have no JMX equivalent. For more details about Batch, refer to the following website:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/twlp_batch_configrest.html?cp=SSAW57_8.5.5%2F3-3-11-0-5-11-1

Various tools can be used to connect to the JMX framework. The standard tools that are used to administer Liberty servers are shown in Figure 5-2.

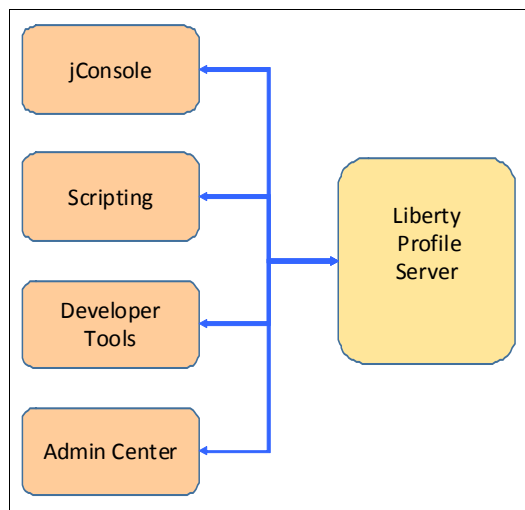


Figure 5-2 Administration tooling

As shown in Figure 5-2, there are four standard tools for managing Liberty (JConsole, scripting, Admin Center, and developer tools). Scripting, JConsole, and the Admin Center all require a JMX connector to be enabled within the Liberty server (these tools are discussed in upcoming sections). Managing Liberty through the development tools is discussed in detail in the Redbooks publication *IBM WebSphere Application Server Liberty Profile Guide for Developers*, SG24-8076 and is available at the following website:

<http://www.redbooks.ibm.com/abstracts/sg248076.html?open>

Note: Both WebSphere Application Server Classic (WAS Classic) and Liberty use JMX as the mechanism to expose administrative functions to client tools. However, the MBeans that are exposed are different in the two products and hence using the existing WebSphere Application Server Classic JMX administrative clients on Liberty does not work.

5.3.1 Connecting with JMX

Liberty provides two options for connecting to the JMX framework and are noted in the following list and discussed in greater detail in later sections:

- ▶ The local connector feature
- ▶ The REST connector feature

To explore these options, create a simple server by using the **server create** command from the WLP_INSTALL_DIR/bin directory, as shown in Example 5-3.

Example 5-3 Create a Liberty profile server to demonstrate JMX connectivity

```
$ server create server1
Server server1 created
$
```

The new server is created in the WLP_INSTALL_DIR/usr/servers/server1 directory, also known as the SERVER_CONFIG_DIR directory. The next sections describe how to enable the JMX connector features and connect to this simple server by using the standard tooling.

Connecting with the local connector

To provide a JMX connector to a local client, the local connector feature needs to be added to the newly created server. To do this, the localConnector-1.0 feature needs to be added to the server.xml found in the SERVER_CONFIG_DIR, as shown in bold in Example 5-4.

Example 5-4 server.xml file with the localConnector-1.0 feature enabled

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

  <!-- Enable features -->
  <featureManager>
    <feature>webProfile-7.0</feature>
    <feature>localConnector-1.0</feature>
  </featureManager>

  <!-- To access this server from a remote client add a host attribute to the
  following element, e.g. host="*" -->
  <httpEndpoint id="defaultHttpEndpoint"
    httpPort="9080"
    httpsPort="9443" />

</server>
```

The localConnector allows a client application (on the same host as the Liberty server) to access the MBeans of the server. Access through the local connector is protected by the policy implemented in the Java Software Development Kit (SDK) in use. Currently, the SDKs require that the client runs on the same host as the Liberty server, and under the same user ID.

To be able to test that the local connector is functioning correctly, start the newly created server by using the **server start** command, which is shown in Example 5-5.

Example 5-5 Start the server1 server

```
$ server start server1
Starting server server1
Server server1 started with process ID 25135
$
```

To connect to the newly created server from JConsole, start the **jconsole** command.

Note: A Java SDK version 1.6 or higher is needed to test connectivity to the server with the local connector feature enabled. The JConsole application is not shipped with the Java runtime environment (JRE) releases. If the installation feature described in section 5.1.1, “The Liberty environment” on page 72 has been followed, the JConsole application is available in the `{wlp.install.dir}/java/java/bin` directory.

Figure 5-3 shows the initial connection window for connecting to the Liberty server by using the JConsole application. To make the connection to the server, ensure **Local Process** is selected along with the **ws-server.jar server1 process**. Then, click **Connect**.

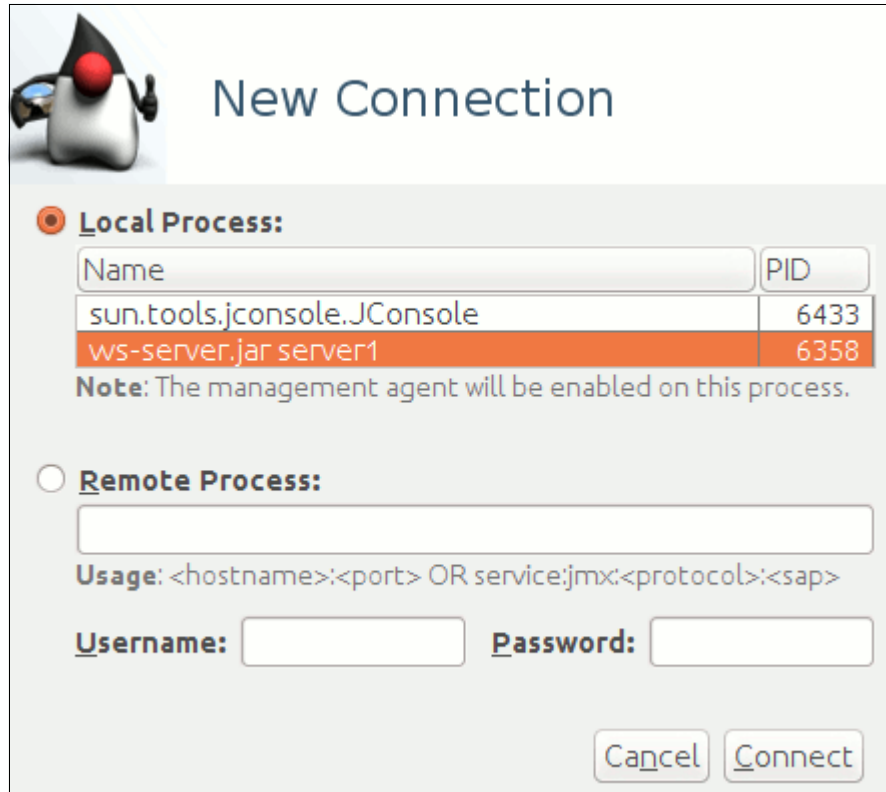


Figure 5-3 Connecting to the Liberty profile server with JConsole

The JConsole application initially tries to make a secure connection to the Liberty server. Because no security has been configured on the Liberty server, the initial secure connection fails and you are then prompted to either cancel or create an insecure connection. Select the option to create an **Insecure connection**.

JConsole uses the extensive instrumentation of the Java virtual machine running the Liberty server. JConsole uses this to provide information about the performance and resource consumption of the Liberty server itself and applications running within the server. Tabs are provided to show an overview of the resource usage. The last tab, the MBean tab shows the JMX resources that are available within the Liberty server runtime, and allows a user of the JConsole application to invoke the operations on these MBeans.

The MBeans that are available in the server1 server and their operations can be seen in Figure 5-4.

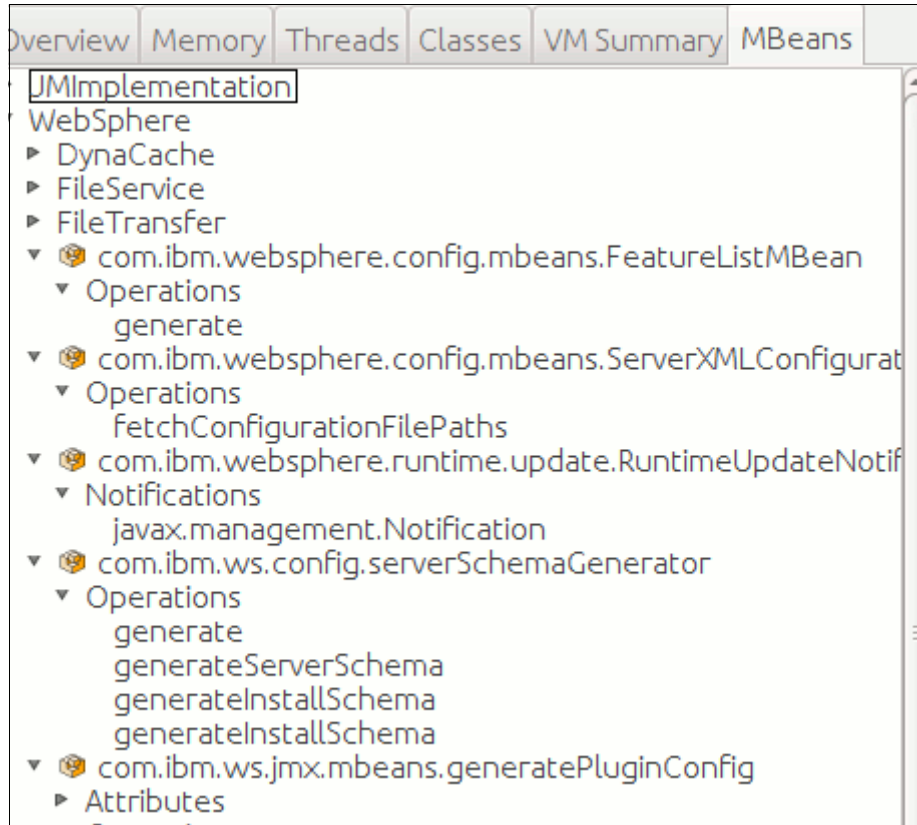


Figure 5-4 JConsole showing available MBeans

In summary, the local connector allows the use of JConsole to connect to a running Liberty server. The JConsole application allows the user to browse and invoke operations available on the MBeans provided by the Liberty server process

Connecting with the REST connector

To provide connections to remote clients, Liberty provides the REST connector feature. The REST connector provides secure access to the MBeans in the Liberty server. Access through the REST connector is protected by a single administrator role. Secure Sockets Layer (SSL) is required to keep the communication confidential.

To configure the server1 server with the REST connector, the `restConnector-1.0` feature needs to be added to the `server.xml`. The `quickStartSecurity` element is added to define the administrator for the Liberty administrative domain. In addition, the `keystore` element is added to define the keystore and password used for secure communication. The `host` attribute is added to the `httpEndPoint` element to allow access to the server from remote clients. The updated `server.xml` is shown in Example 5-6.

Example 5-6 The `server.xml` with the `quickStartSecurity` and a keystore defined

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

  <!-- Enable features -->
  <featureManager>
```

```

    <feature>webProfile-7.0</feature>
    <feature>localConnector-1.0</feature>
    <feature>restConnector-1.0</feature>
  </featureManager>

  <!-- To access this server from a remote client add a host attribute to the
  following element, e.g. host="*" -->
  <httpEndpoint id="defaultHttpEndpoint"
    host="*"
    httpPort="9080"
    httpsPort="9443" />

  <quickStartSecurity userName="admin" userPassword="adminPwd" />
  <keyStore id="defaultKeyStore" password="keystorePwd" />
</server>

```

Notes:

- ▶ All the examples in this chapter use the **quickStartSecurity** capability. The **quickStartSecurity** element can be used to quickly enable a simple (one user) security setup for the Liberty profile. The **quickStartSecurity** capability should not be used for production environments. For more information about securing the Liberty administrative domain, see the IBM Knowledge Center at the following website: https://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/cwlp_sec.html?lang=en
- ▶ In this chapter, all passwords are shown in clear text for readability. In a production environment, passwords should be encoded or encrypted. The admin password can be encoded by using the **securityUtility** as defined in 4.2.4, “Encrypting passwords” on page 60.

At this point, if the Liberty server `server1` is running, after saving the configuration changes made to the `server.xml` in Example 5-6 on page 77, the Liberty server process automatically reloads the new configuration.

If the Liberty server, `server1`, is not running, restart the server with the **server start** command. To see if the configuration successfully changed, review the `SERVER_CONFIG_DIR/logs/console.log`. The messages shown in Example 5-7 are present in the log file if the Liberty server started successfully.

Example 5-7 Console log showing server updating to use new rest connector

```

[AUDIT   ] CWWKG0016I: Starting server configuration update.
[AUDIT   ] CWWKG0017I: The server configuration was successfully updated in 0.907
seconds.
[AUDIT   ] CWWKF0012I: The server installed the following features:
[restConnector-1.0].
[AUDIT   ] CWWKF0008I: Feature update completed in 0.896 seconds.
[AUDIT   ] CWWKT0016I: Web application available (default_host):
http://localhost:9080/IBMJMXConnectorREST/

```

The addition of the **keyStore** element to the `server1` configuration, as shown in Example 5-6 on page 77, causes the server to create keys for the SSL communication in the `SERVEFR_CONFIG_DIR/resources/security` directory.

Now that the server1 process is running with the REST connector feature enabled, a number of REST APIs can be accessed by any REST-compliant client programming language. The API documentation can be accessed through the `https://localhost:9443/IBMJMXConnectorREST/api`, which documents the available APIs, which are shown in Figure 5-5. Access to the URL is secured by using the credentials defined in the `quickStartSecurity` element.



Figure 5-5 The IBM JMXConnectorREST API

File transfer

The `restConnector-1.0` feature includes the `FileTransfer` and `FileService` MBeans. The `FileTransfer` MBean supports delete, upload, and download operations to and from a running Liberty server. The `FileService` MBean provides access to directory lists and file metadata, and it also provides archive operations such as create and expand.

The `FileTransfer` and `FileService` MBeans are useful for carrying out remote operations on a Liberty server, such as updating the configuration or installing an application.

A configuration update can be performed remotely by uploading the following types of files:

- ▶ An updated `server.xml`
- ▶ Other configuration files such as `includes` (for placement in the `includes` or the `dropins` directories of the target Liberty server).

An application can be installed by the following methods:

- ▶ Uploading both the application archive and an updated `server.xml` file.
- ▶ Uploading the application archive to the monitored application dropins folder. For details, see 4.6, “Configuring dynamic application updates” on page 66.

The FileTransfer service can also be used to transfer a packaged Liberty server to a remote host with no Liberty servers running.

The FileTransfer MBean includes configurable read and write lists so that you can control the directories that can be read or written when using the FileTransfer MBean.

Liberty scripting for WAS Classic users

When scripting, administrators of WAS Classic can use the `wsadmin` scripting tool to administer production environments. The `wsadmin` tool provides a command-line interface to automate common tasks using Jacl or Jython scripts. The `wsadmin` tool provides a set of objects that can be used to configure and administer application servers, application deployment, and server runtime operations. Scripts use these objects to communicate with the MBeans that represent live objects running in a WAS Classic server. To run a Jython script, the `wsadmin` command is run from the command line as shown in Example 5-8.

Example 5-8 Command line to execute a jython script using wsadmin

```
$ wsadmin -jython -f sample.py
```

The Jython script automatically has access to the `wsadmin` objects and they can be accessed in the Jython script, as shown in Example 5-9.

Example 5-9 Sample wsadmin jython script stop a server

```
AdminControl.stopServer( 'serverName' )
```

The `wsadmin` tool does not apply to Liberty. In Liberty, the administrative scripts communicate directly with the MBeans running in the Liberty servers. The basic structure of a Jython script for administering the Liberty profile is shown in Example 5-10.

Example 5-10 A sample script showing key aspects of Jython scripting for the Liberty profile

```
#Import the required modules from the Liberty client. The client files can be
# found in the WLP_INSTALL_DIR/clients directory.
#The restConnector.jar file needs to be on the CLASSPATH.
import restConnector from JMXRESTConnector

#Allow one to create MBean ObjectName instance from string input
from javax.management import ObjectName

#Set up the trust store for secure communication between client and server
#the truststore variable points to a location on the file system
#that contains the keystore created when the server was created.
JMXRestConnector.trustStore =
WLP_INSTALL_DIR/usr/servers/server1/resources/security/key.jks

#The password is that used for the keystore when the server was created
JMSRESTConnector.trustStorePassword = keystorePwd

#Establish connectivity to the server.
#Use the administrative domain admin user and password
```



```

#as defined in the server.xml
connector.connect("localhost",9443,"admin","adminPwd")

# get an MBean server connection
mconnector = connector.getMbeanServerConnection()

#Identify the MBean to be invoked in this example we will se the File Transfer
MBean that is made available as part of the installation of the rest connector
fileTransfer = ObjectName(
"WebSphere:feature=restConnector,type=FileTransfer,name=FileTransfer" )

#Invoke the MBean operation downloadFile to download the server.xml from the
server to a local directory
mconnection.invoke( fileTransfer, "downloadFile",
    [${SERVER_CONFIG_DIR} + "/server.xml", "/home/itsouser/server.xml"],
    ["java.lang.string", "java.lang.string"])

#disconnect from the server
connector.disconnect()

```

The sample script is saved to a file called `testJMXConnection.py` on the local file system. The script is executed by using the Jython command, as shown in Example 5-11.

Example 5-11 Command to execute the testJMXConnection script

```

$ jython testJMXConnection.py
Connecting to the server...
Successfully connected to the server "localhost:9443"
$

```

The script has executed correctly if the server's `server.xml` file is available in the `/home/itsouser` directory.

The available MBeans are documented in the IBM Knowledge Center at the following website:

https://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/rwlp_mbeans_list.html

5.3.2 Connecting through the Admin Center

The Liberty Administrative Center (Admin Center) can be used to administer Liberty servers, applications, clusters, and hosts from a web browser on a smartphone, tablet, or computer. Admin Center offers the ability to view details about and perform operations (start, stop, restart, add and remove metadata, enable and disable maintenance mode) on resources within the collective. It also offers the ability to edit server configuration files to view bookmark information, to add custom tools to monitor server resources, and to deploy server packages on hosts within the collective.

To be able to connect to the Admin Center of a Liberty server, the Admin Center feature first needs to be enabled and configured. The **webProfile** installation does not include the Admin Center feature by default. To install the Admin Center feature, use the **installUtility** command (described in 3.6.1, “Installing assets by using the installUtility command” on page 48) and install the `adminCenter-1.0` feature.

To configure the server1 server with the Admin Center feature, the adminCenter-1.0 feature is added to the featureManager element in the server.xml of the server1 server. The updated server.xml is shown in Example 5-12.

Example 5-12 Updating the server.xml to enable the Admin Center

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

  <!-- Enable features -->
  <featureManager>
    <feature>webProfile-7.0</feature>
    <feature>localConnector-1.0</feature>
    <feature>restConnector-1.0</feature>
    <feature>adminCenter-1.0</feature>
  </featureManager>

  <!-- To access this server from a remote client add a host attribute to the
  following element, e.g. host="*" -->
  <httpEndpoint id="defaultHttpEndpoint"
    host="*"
    httpPort="9080"
    httpsPort="9443" />

  <quickStartSecurity userName="admin" userPassword="adminPwd" />
  <keystore id="defaultKeyStore" password="keystorePwd" />
</server>
```

Note: In example Example 5-12, the restConnector and adminCenter are both enabled in the featureManager. The restConnector is extraneous as the adminCenter feature loads the restConnector feature and the SSL feature.

If the server1 server is running, it notices the changes made to the server.xml and loads the Admin Center. If the load is successful, the messages.log file in the SERVER_CONFIG_DIR/logs should contain the messages shown in Example 5-13.

Example 5-13 Successfully loaded the Admin Center

```
SRVE0169I: Loading Web Module: The Liberty Admin Center.
SRVE0250I: Web Module The Liberty Admin Center has been bound to default_host.
```

To connect to the Admin Center, use the URL as documented in the messages.log file. This example uses https://localhost:9443/adminCenter. The login window displays as shown in Figure 5-6. Log in as the administrator by using the credentials that were added to the quickStartSecurity tag of the server.xml for server1.

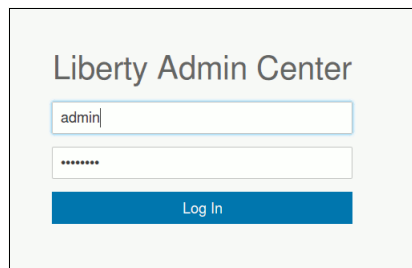


Figure 5-6 Login window for the Admin Center

After logging in, the Admin Center toolkit is displayed as shown in Figure 5-7.

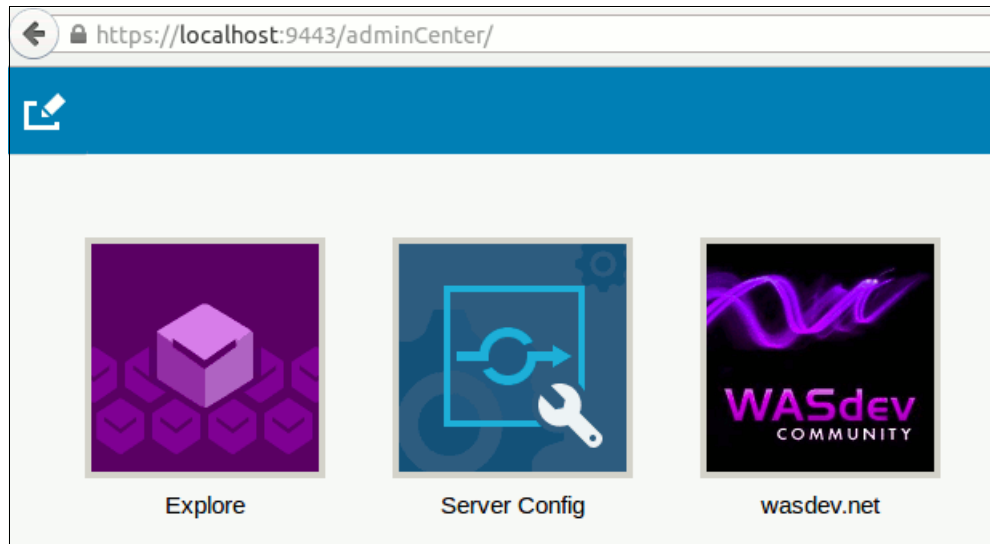


Figure 5-7 The default toolbox for the Admin Center for the `jmxconnectiontest` server

When first logging in to the Admin Center, the toolbox contains the Server Config and Explore tools and a bookmark to `wasdev.net`. If the Admin Center is run on a collective controller, the toolbox also has the Deploy tool. The Deploy tool is not available in this example because the server is a stand-alone server. The Deploy tool is investigated further in “Deploying packages using the Admin Center” on page 100.

By clicking the **Server Config** tool, and then selecting `server.xml`, the `server.xml` of this example’s server can be viewed (see Figure 5-8).

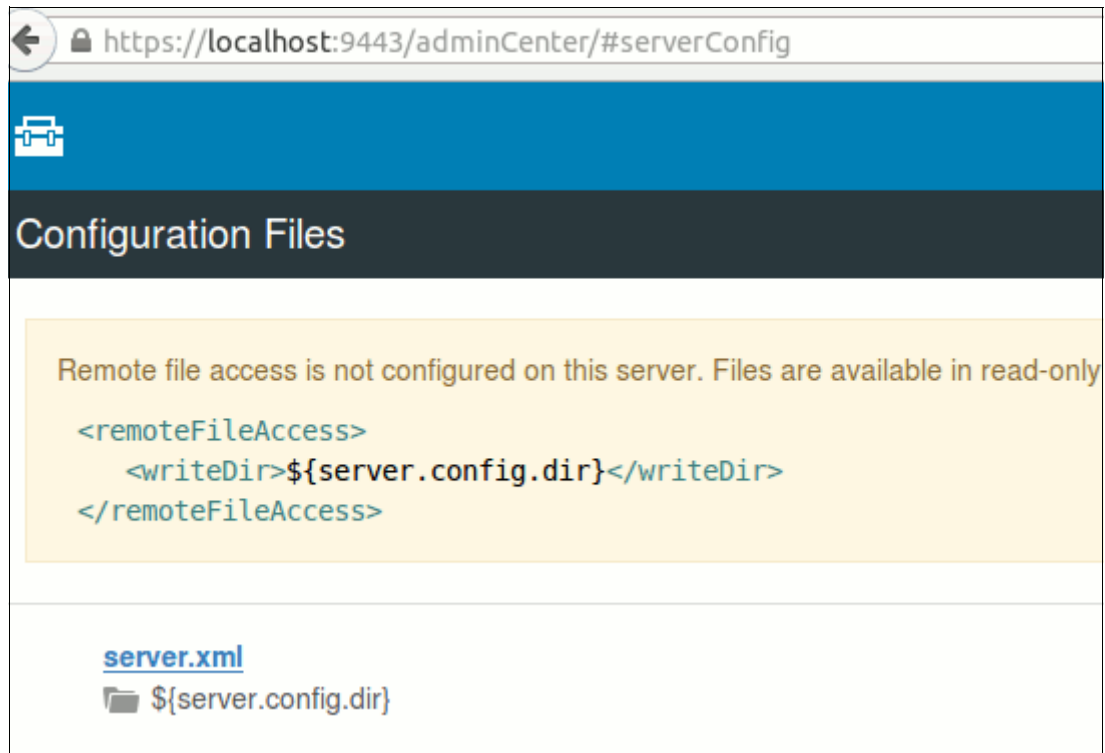


Figure 5-8 The server config tool

By default, when using the Admin Center, only read access is granted to the `server.xml`. If write access is needed to allow for `server.xml` updates from the Admin Center, the `server.xml` for `server1` needs to be updated to allow remote file access. The updated `server.xml` is shown in Example 5-14.

Example 5-14 Updated server.xml to support write access from the Admin Center

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

  <!-- Enable features -->
  <featureManager>
    <feature>webProfile-7.0</feature>
    <feature>localConnector-1.0</feature>
    <feature>restConnector-1.0</feature>
    <feature>adminCenter-1.0</feature>
  </featureManager>

  <!-- To access this server from a remote client add a host attribute to the
  following element, e.g. host="*" -->
  <httpEndpoint id="defaultHttpEndpoint"
    host="*"
    httpPort="9080"
    httpsPort="9443" />

  <quickStartSecurity userName="admin" userPassword="adminPwd" />
  <keystore id="defaultKeyStore" password="keystorePwd" />
  <remoteFileAccess>
    <writeDir>${server.config.dir}</writeDir>
  </remoteFileAccess>
</server>
```

The `server.xml` can now be edited, as shown in Figure 5-9.

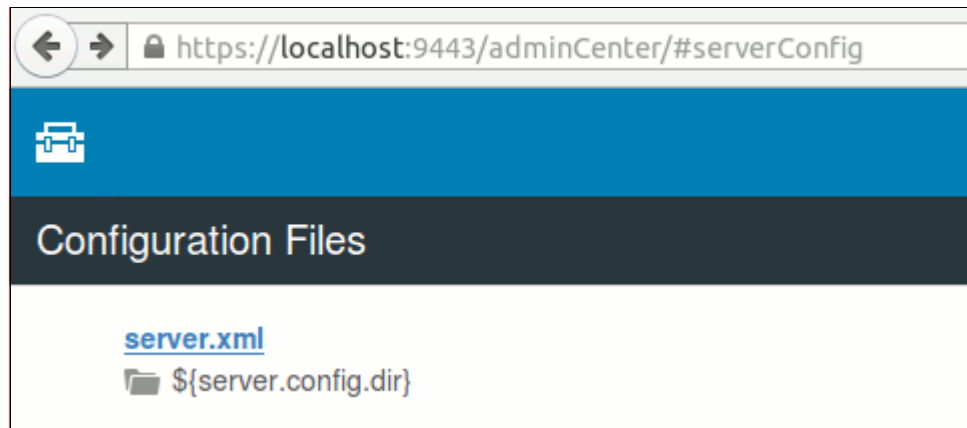


Figure 5-9 Admin Center with editable server.xml

Clicking `server.xml` allows for the configuration of the `server.xml` to be changed (see Figure 5-10 on page 85).

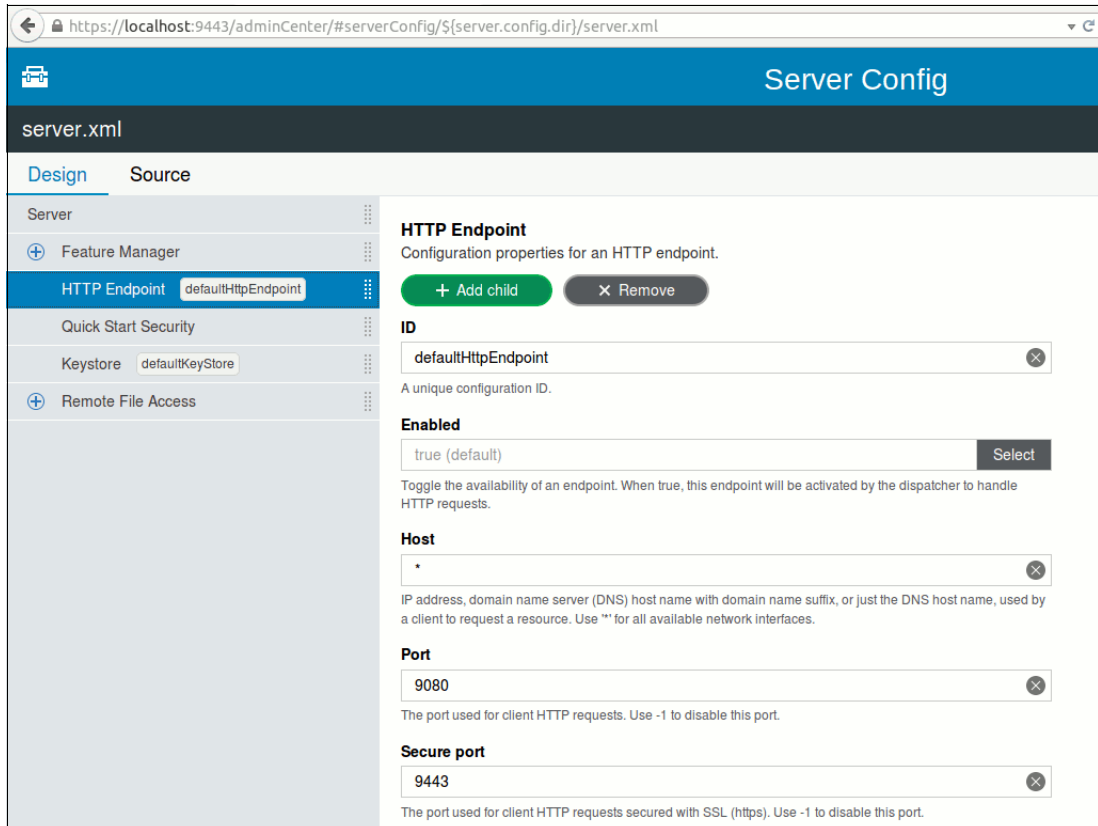


Figure 5-10 Editable server.xml in the Admin Center

The Explore tool provides the capability to monitor the server processes, start and stop applications, and if write is allowed, to also configure servers. The simple single-server configuration for this example's server1 server is shown in Figure 5-11 on page 86.

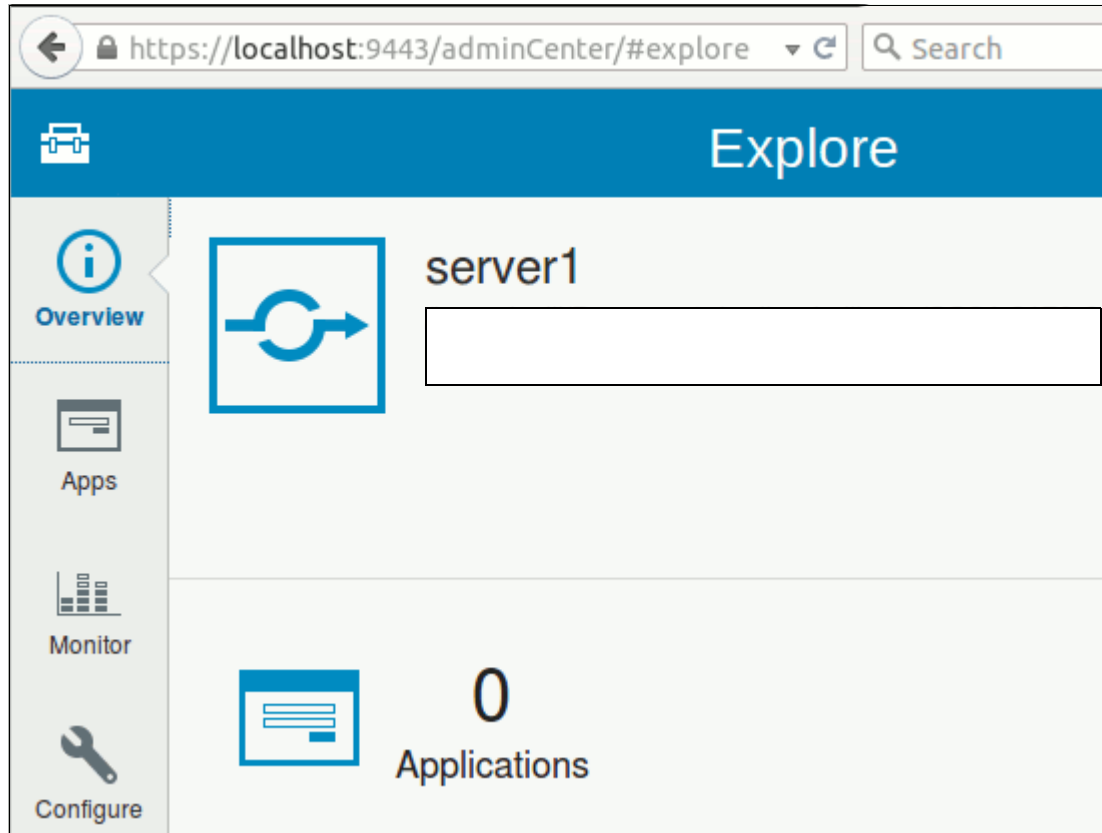


Figure 5-11 Liberty profile Admin Center Explore tool

For more information about the capabilities of the Admin Center, see the IBM Knowledge Center at the following website:

https://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/twlp_ui.html?lang=en

5.4 Liberty collectives

The set of Liberty servers in a single administrative domain is called a *collective*. The collective architecture is shown in Figure 5-12.

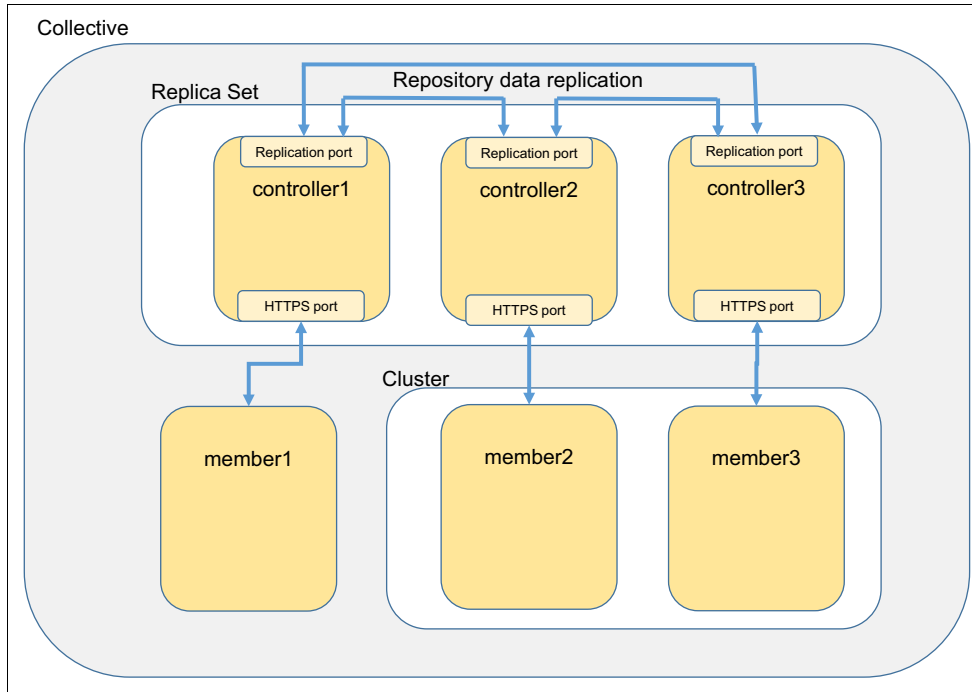


Figure 5-12 Sample Liberty collective architecture

A collective consists of at least one *collective controller*. A collective controller is a Liberty server configured with the collective controller feature. For production environments, it is recommended that three collective controllers are used for resiliency. A set of collective controllers is called a *replica set*. Members of a collective are Liberty servers configured with the collective member feature. Collective members can be clustered with the addition of the cluster member feature. Collective controllers can also be clustered for high availability and scaling purposes.

5.4.1 Comparing Liberty and WAS Classic

Though the administrative domains of the WAS Classic and Liberty profiles are similar in their capabilities, the way these capabilities are implemented are different. Table 5-1 on page 88 provides the comparison of the two profiles.

Table 5-1 Comparing the Liberty and WAS Classic profiles

Feature	WAS Classic	Liberty
Administrative domain name	The administrative domain is called a cell.	The administrative domain is called a <i>collective</i> .
Administrative server	The administrative server is called a <i>deployment manager</i> .	The administrative server is called a <i>collective controller</i> .
Administrative server process	The deployment manager is a dedicated process and executes no workloads.	The collective controller is a feature, which means that it can be enabled with any other feature. This allows the collect controller to run workloads. (Running workloads on a collective controller in production is not recommended).
Becoming part of administrative domain	The process for entities joining the domain is called <i>federation</i> . It is a tightly coupled process whereby servers federate into the cell and give up much of their autonomy.	Servers join the collective and become collective members. The process of joining is very lightweight and allows the members to be loosely coupled to the collective. It is easy for members to join and leave the collective.
Agent or Agentless	The cell environment uses agents to facilitate management activities. The node agent acts as a middleman between the deployment manager and the application servers running on a node.	There are no administrative agents. A collective controller manages the collective members directly.
Configuration control	The deployment manager owns all configuration of all the entities within the cell.	Each collective member owns its own configuration. Even when a member joins or leaves a collective, it retains complete control over its configuration.
Management API	The cell provides MBeans to provide management activities. It also provides the wsadmin command to script access to the MBeans through Jacl and Jython.	The collective provides MBeans to provide management activities. These MBeans are not the same as the MBeans provided by a cell. There is no wsadmin command. And scripting is supported using Jython. Additional languages are possible.

5.4.2 Configuring a Liberty collective controller

The collective controller acts as a command and control mechanism for the administrative functions of the collective. The collective controller also serves as a storage and collaboration mechanism for the collective and cluster members. The collective controller is a standard Liberty server with the `collective controller` feature enabled.

By enabling the `collective controller` feature, the `restConnector` is automatically enabled (enabling JMX client applications to connect to the controller). For web-based access to the collective controller, the `adminCenter` feature is also enabled, as shown in Figure 5-13.

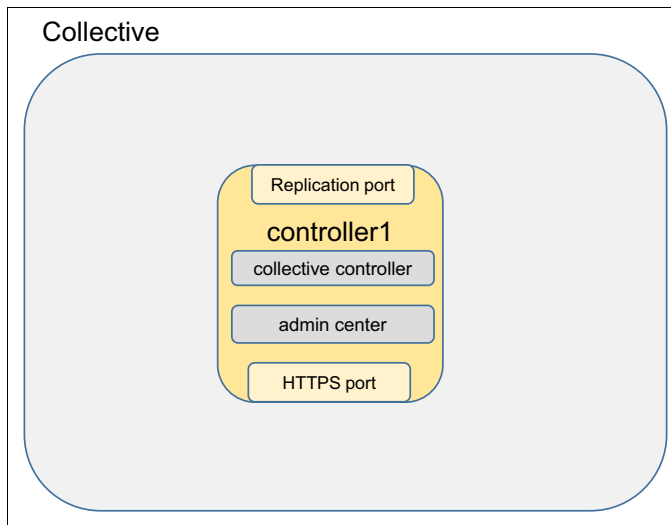


Figure 5-13 Liberty collective controller features

To create a collective controller, the following procedure should be used:

1. Create a Liberty server.
2. Configure the server as a collective controller.
3. Configure administrative security.
4. Enable the admin center feature for the collective controller.

The details for these steps are covered in the next sections.

Create a Liberty profile server

Create a server, `controller1`, by using the `server create` command as shown in Example 5-15.

Example 5-15 Create a Liberty server to act as the collective controller

```
$ ./server create controller1
Server controller1 created
$
```

Configure the server as a collective controller

To configure a Liberty server as a collective controller, use the `collective` command from the `WLP_INSTALL_DIR/bin` directory. This command adds the collective controller feature to the servers configuration and creates the required certificates to establish a collective. The certificates are created in the `SERVER_CONFIG_DIR/resources/security` directory. The format of the command is shown in Example 5-16.

Example 5-16 Configure the controller1 server as a collective controller

```
$ collective create controller1 --keystorePassword=keystorePwd
--createConfigFile=$SERVER_CONFIG_DIR/collective-create-include.xml
Creating required certificates to establish a collective...
This may take a while.
Successfully generated the controller root certificate.
```

Successfully generated the member root certificate.
Successfully generated the server identity certificate.
Successfully generated the HTTPS certificate.

Successfully set up collective controller configuration for controller2

Add the following lines to the server.xml to enable:

```
<include location="{server.config.dir}/collective-create-include.xml" />
```

Please ensure administrative security is configured for the server.
An administrative user is required to join members to the collective.

\$

By using the `--createConfigFile` option, the **collective** command creates the collective controller configuration in a separate file `collective-create-include.xml` in the `SERVER_CONFIG_DIR` directory. The contents of this file are shown in Example 5-17.

Example 5-17 collective-create-include.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<server description="This file was generated by the 'collective create' command on
2015-08-11 11:39:55 EDT.">
  <featureManager>
    <feature>collectiveController-1.0</feature>
  </featureManager>

  <!-- Define the host name for use by the collective.
  If the host name needs to be changed, the server should be
  removed from the collective and re-joined or re-replicated. -->
  <variable name="defaultHostName" value="localhost" />

  <!-- TODO: Set the security configuration for Administrative access -->
  <quickStartSecurity userName="" userPassword="" />

  <!-- clientAuthenticationSupported set to enable bidirectional trust -->
  <ssl id="defaultSSLConfig"
    keyStoreRef="defaultKeyStore"
    trustStoreRef="defaultTrustStore"
    clientAuthenticationSupported="true" />

  <!-- inbound (HTTPS) keystore -->
  <keyStore id="defaultKeyStore" password="{xor}NDomLCswLToPKDs="
    location="{server.config.dir}/resources/security/key.jks" />

  <!-- inbound (HTTPS) truststore -->
  <keyStore id="defaultTrustStore" password="{xor}NDomLCswLToPKDs="
    location="{server.config.dir}/resources/security/trust.jks" />

  <!-- server identity keystore -->
  <keyStore id="serverIdentity" password="{xor}NDomLCswLToPKDs="
    location="{server.config.dir}/resources/collective/serverIdentity.jks" />

  <!-- collective trust keystore -->
```

```

<keyStore id="collectiveTrust" password="{xor}NDomLCswLToPKDs="
  location="${server.config.dir}/resources/collective/collectiveTrust.jks" />

<!-- collective root signers keystore -->
<keyStore id="collectiveRootKeys" password="{xor}NDomLCswLToPKDs="
  location="${server.config.dir}/resources/collective/rootKeys.jks" />

</server>

```

At this point, the `SERVER_CONFIG_DIR/configDropin/default` directory can be used as the path for the `--createConfigFile` parameter. Using the directory as the path for the parameter means the `server.xml` does not have to be modified to include the `collective-create-include.xml`. See section “Use the configuration dropins folder to specify server configuration” on page 66 for details. For clarity in this example, the `server.xml` is modified and the newly created file is included. The ports are also updated to avoid port conflicts. The new `server.xml` is shown in Example 5-18.

Example 5-18 Collective controller server.xml modified to include collective-create-include.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

  <!-- Enable features -->
  <featureManager>
    <feature>webProfile-7.0</feature>
  </featureManager>

  <!-- To access this server from a remote client add a host attribute to the
  following element, e.g. host="*" -->
  <httpEndpoint id="defaultHttpEndpoint"
    host="*"
    httpPort="9081"
    httpsPort="9444" />
  <include location="${server.config.dir}/collective-create-include.xml"/>
</server>

```

Note: The collective controller feature enables a number of other features. One of those features enabled is the `restConnector-1.0` feature. The `restConnector-1.0` feature is discussed in “Connecting with the REST connector” on page 77. The entire set of features included when enabling the collective controller feature is available in the IBM Knowledge Center at the following website:

https://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/rwlp_feature_adminCenter-1.0.html?lang=en

Configure administrative security

All the JMX methods and MBeans accessed through the REST connector are currently protected by a single role named *administrator*. To be able to use the JMX methods from a remote client, the administrator role needs to be mapped onto a single user.

Example 5-19 on page 92 shows how to map a user, *Admin*, to the administrator role to secure the collective controller by using the `quickStartSecurity` element. Mapping to the administrator role is accomplished by updating the line in the `collective-create-include.xml` with the TODO comments. The actual user name and password details need to be updated, as shown in Example 5-19 on page 92.

Example 5-19 Update the collective-create-include.xml file with user details

```
<!-- TODO: Set the security configuration for Administrative access -->
  <quickStartSecurity userName="admin" userPassword="adminPwd" />
```

The **webProfile** installation does not include the collective controller feature by default. To install the collective controller feature, use the **installUtility** command described in 3.6.1, “Installing assets by using the installUtility command” on page 48 and install the **collectiveController-1.0** feature.

Ensure that the server, **server1**, is stopped by using the **server stop** command. At this point, the collective controller can be started to ensure that all is functioning correctly. To start the collective controller, use the **server run** command, as shown in Example 5-20.

Example 5-20 Run the collective controller

```
$ server run controller1
Launching controller1 (WebSphere Application Server
2015.8.0.0/wlp-1.0.10.20150728-1158) on IBM J9 VM, version pxa6470sr9-20150417_01
(SR9) (en_US)
[AUDIT   ] CWWKE0001I: The server controller1 has been launched.
[AUDIT   ] CWWKZ0058I: Monitoring dropins for applications.
[AUDIT   ] CWWKT0016I: Web application available (default_host):
http://localhost.localdomain:9080/ibm/api/collective/notify/
[AUDIT   ] CWWKT0016I: Web application available (default_host):
http://localhost.localdomain:9080/IBMJMXConnectorREST/
[AUDIT   ] CWWKT0016I: Web application available (default_host):
http://localhost.localdomain:9080/ibm/api/
[AUDIT   ] CWWKT0016I: Web application available (default_host):
http://localhost.localdomain:9080/ibm/adminCenter/explore-1.0/
[AUDIT   ] CWWKT0016I: Web application available (default_host):
http://localhost.localdomain:9080/ibm/adminCenter/deploy-1.0/
[AUDIT   ] CWWKT0016I: Web application available (default_host):
http://localhost.localdomain:9080/ibm/adminCenter/serverConfig-1.0/
[AUDIT   ] CWWKT0016I: Web application available (default_host):
http://localhost.localdomain:9080/adminCenter/
[AUDIT   ] CWWKF0012I: The server installed the following features:
[collectiveMember-1.0, webProfile-7.0, json-1.0, appSecurity-2.0, jaxrs-2.0,
jpa-2.1, cdi-1.2, restConnector-1.0, jaxrsClient-2.0, javaMail-1.5,
distributedMap-1.0, websocket-1.1, el-3.0, jdbc-4.1, ssl-1.0, beanValidation-1.1,
managedBeans-1.0, servlet-3.1, adminCenter-1.0, jsf-2.2, jsp-2.3, jndi-1.0,
jsonp-1.0, collectiveController-1.0, ejbLite-3.2].
[AUDIT   ] CWWKF0011I: The server controller1 is ready to run a smarter planet.
```

Tip: When you are starting the collective controller for the first time (or any server), use the **server run** command to start the controller rather than the **server start** command. This allows you to monitor the start messages to catch any configuration errors. The **server run** command does not return control to the command line. Interrupting the server by using CTRL-C causes the server to exit.

5.4.3 Registering host computers within a Liberty collective

A Liberty collective can span a number of host systems. For the host systems to be able to communicate, the hosts need to be registered with the Liberty collective controller.

Setting up RXA for Liberty collective operations

Before the hosts are added to the Liberty collective, the host operating system needs to be configured to support SSH. Liberty collective controllers use the IBM Tivoli® Remote Execution and Access (RXA) toolkit to perform selected operations on collective members. RXA uses Secure Shell (SSH) for communication. Liberty does provide an SSH client, but the host systems that are to be part of the collective need an SSH server installed. See the IBM Knowledge Center for information about setting up RXA:

https://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/twlp_set_rxa.html?lang=en

Using the collective command to register the host

A remote host needs to be registered with a Liberty collective controller for the collective controller to be able to access applications, command files, and other resources on the host. A host does not need any Liberty code installed. An RXA connection suffices (as defined in the previous section). Example 5-21 shows how a remote host is registered in the collective by using the `collective registerHost` command.

Example 5-21 Registering a remote host

```
$ collective registerHost remotehost --host=localhost --port=9444 --user=admin
--password=adminPwd --rpcUser=remoteUser --rpcUserPassword=remote_Pwd
Registering the host to the collective...
```

```
SSL trust has not been established with the target server.
```

```
Certificate chain information:
```

```
Certificate [0]
```

```
Subject DN: CN=localhost.localdomain, OU=controller1, O=ibm, C=us
```

```
Issuer DN: OU=controllerRoot, O=6c0ccd20-a340-428d-9cc4-22615c75cb84,
DC=com.ibm.ws.collective
```

```
Serial Number: 222,687,534,027,281
```

```
Expires: 8/8/20 3:26 PM
```

```
SHA-1 digest: 72:C0:6F:54:36:71:4D:FE:47:B9:C4:29:D7:5C:86:04:CD:8C:F1:CC
```

```
MD5 digest: 36:6F:E5:EF:90:68:9D:D6:29:15:BC:17:96:D3:D5:DB
```

```
Certificate [1]
```

```
Subject DN: OU=controllerRoot, O=6c0ccd20-a340-428d-9cc4-22615c75cb84,
DC=com.ibm.ws.collective
```

```
Issuer DN: OU=controllerRoot, O=6c0ccd20-a340-428d-9cc4-22615c75cb84,
DC=com.ibm.ws.collective
```

```
Serial Number: 222,684,490,785,018
```

```
Expires: 8/3/40 3:26 PM
```

```
SHA-1 digest: BD:C9:8E:F0:25:90:C7:8B:8B:B0:7A:80:1D:6D:F6:0A:B2:DF:0A:AC
```

```
MD5 digest: 7F:18:D7:5E:39:78:5A:23:6C:D7:F5:22:93:96:4B:5D
```

```
Do you want to accept the above certificate chain? (y/n) y
Host remotehost successfully registered.
```

Registration enables the collective controller to access applications, command files, and other resources on the host. To do this, the collective controller needs to be running. Certificates are needed for secure communication between the host computer and the collective controller. The certificate chain created by the collective controller needs to be accepted by the user.

5.4.4 Creating a collective member

A collective member is a Liberty server with the collective member feature enabled, as shown in Figure 5-14.

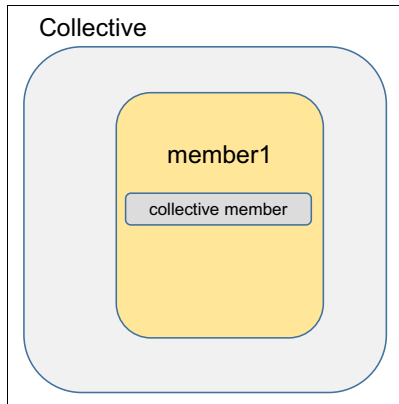


Figure 5-14 A collective member

To create a collective member, the following procedure is required:

1. Create the collective member server.
2. Add the `collectiveMember` feature.
3. Add the endpoint information for the server.
4. Join the collective member to the collective.

A collective member is created, as with any other Liberty server, by using the `server create` command as shown in Example 5-22.

Example 5-22 Create the collective member1 server

```
$ server create member1
Server member1 created
$
```

The server configuration is created in the `WLP_INSTALL_DIR/usr/servers/member1` directory. The `server.xml` needs to be modified because all servers are running on the same host and you need to avoid port conflicts.

The updated `server.xml` is shown in Example 5-23.

Example 5-23 Update the server.xml of member1 to avoid port conflicts

```
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>webProfile-7.0</feature>
    </featureManager>

    <!-- To access this server fy
    rom a remote client add a host attribute to the following element, e.g. host="*"
    -->
    <httpEndpoint id="defaultHttpEndpoint"
        httpPort="9082"
```

```
httpsPort="9445" />
```

```
</server>
```

This newly created server is then joined to the collective by using the **collective join** command, as shown in Example 5-24.

Example 5-24 Joining a member to the collective

```
$ collective join member1 --host=localhost --port=9444 --user=admin --  
password=adminPwd --keystorePassword=keystorePwd  
--createConfigFile=SERVER_CONFIG_DIR/collective-join-include.xml  
Joining the collective with target controller localhost:9444...  
This may take a while.
```

SSL trust has not been established with the target server.

Certificate chain information:

Certificate [0]

```
Subject DN: CN=localhost, OU=controller1, O=ibm, C=us  
Issuer DN: OU=controllerRoot, O=b222ec67-3a88-4bd9-9c6d-57ad19d0187e,  
DC=com.ibm.ws.collective  
Serial Number: 8,338,143,633,065  
Expires: 8/19/20 11:13 AM  
SHA-1 digest: 1B:CE:8C:7F:37:A6:5B:A3:06:DF:B9:C7:42:2D:5E:26:22:55:58:C7  
MD5 digest: BF:58:7B:7E:B1:A2:40:F0:A4:1C:86:D0:C2:CA:5E:32
```

Certificate [1]

```
Subject DN: OU=controllerRoot, O=b222ec67-3a88-4bd9-9c6d-57ad19d0187e,  
DC=com.ibm.ws.collective  
Issuer DN: OU=controllerRoot, O=b222ec67-3a88-4bd9-9c6d-57ad19d0187e,  
DC=com.ibm.ws.collective  
Serial Number: 8,333,307,608,526  
Expires: 8/14/40 11:13 AM  
SHA-1 digest: 2E:8A:2B:38:31:18:9A:CC:C8:55:6E:AD:C8:C8:E5:DC:22:8A:DC:42  
MD5 digest: FE:DF:00:F3:53:C5:79:E5:80:0E:4D:6A:54:04:47:0E
```

```
Do you want to accept the above certificate chain? (y/n) y  
Successfully completed MBean request to the controller.
```

Successfully joined the collective for server member1

Add the following lines to the server.xml to enable:

```
<include location="{server.config.dir}/collective-join-include.xml" />
```

The **collective join** command creates a configuration file, `collective-join-include.xml`, in the `SERVER_CONFIG_DIR` directory. This configuration file adds the `collectiveMember-1.0` feature to the `featureManager`. The `collective-join-include.xml` configuration file configures the `collectiveMember` to connect to the collective controller host and establishes the default SSL configuration and associated keystores.

See Example 5-25.

Example 5-25 Content of the collective-join-include.xml file

```
<?xml version="1.0" encoding="UTF-8" ?>
<server description="This file was generated by the 'collective join' command on
2015-08-21 11:47:24 EDT.">
  <featureManager>
    <feature>collectiveMember-1.0</feature>
  </featureManager>

  <!-- Define the host name for use by the collective.
    If the host name needs to be changed, the server should be
    removed from the collective and re-joined or re-replicated. -->
  <variable name="defaultHostName" value="localhost" />

  <!-- Connection to the collective controller -->
  <collectiveMember controllerHost="localhost"
    controllerPort="9444" />

  <!-- clientAuthenticationSupported set to enable bidirectional trust -->
  <ssl id="defaultSSLConfig"
    keyStoreRef="defaultKeyStore"
    trustStoreRef="defaultTrustStore"
    clientAuthenticationSupported="true" />

  <!-- inbound (HTTPS) keystore -->
  <keyStore id="defaultKeyStore" password="{xor}NDomLCswLToPKDs="
    location="{server.config.dir}/resources/security/key.jks" />

  <!-- inbound (HTTPS) truststore -->
  <keyStore id="defaultTrustStore" password="{xor}NDomLCswLToPKDs="
    location="{server.config.dir}/resources/security/trust.jks" />

  <!-- server identity keystore -->
  <keyStore id="serverIdentity" password="{xor}NDomLCswLToPKDs="
location="{server.config.dir}/resources/collective/serverIdentity.jks" />

  <!-- collective truststore -->
  <keyStore id="collectiveTrust" password="{xor}NDomLCswLToPKDs="
location="{server.config.dir}/resources/collective/collectiveTrust.jks" />

</server>
```

The `collective-join-include.xml` file is added to the configuration of the `member1` server by updating the `server.xml`, as shown in Example 5-26.

Example 5-26 Updated member1 server.xml with include to collective-join-include.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

  <!-- Enable features -->
  <featureManager>
```



```

    <feature>webProfile-7.0</feature>
  </featureManager>

  <!-- To access this server from a remote client add a host attribute to the
  following element, e.g. host="*" -->
  <httpEndpoint id="defaultHttpEndpoint"
    httpPort="9082"
    httpsPort="9445" />

  <include location="${server.config.dir}/collective-join-include.xml" />

</server>

```

5.4.5 Adding members to the Liberty profile collective

If the member1 server, as configured in the previous section, is started it will automatically join the collective controlled by controller1. Start the member by using the **server start** command, as shown in Example 5-27.

Example 5-27 Start the member1 server by using the start server command

```

$ server start member1
Server member1 started with process ID 14329.
$

```

The messages.log for the member1 server should show that the member1 server has joined the collective (see Example 5-28).

Example 5-28 Messages.log showing that the member has joined the collective

```

CWWKX8055I: The collective member has established a connection to the collective
controller.

```

In a production environment, a more flexible approach to deployment is needed. The majority of times, the collective members are running on hosts other than the host where the collective controller is running. To deploy to a remote host, a packaged Liberty server is required. To package a server and its configuration for deployment, first stop the member1 server using the **server stop** command. Then, remove the server from the collective by using the **collective remove** command, as shown in Example 5-29.

Example 5-29 Removing a collective member from the collective

```

$ collective remove member1 --host=localhost --port=9444 --user=admin
--password=adminPwd
Attempting to unregister the server from the collective...
Host: localhost
User Dir: WLP_INSTALL_DIR/usr/
Server Name: member1

```

SSL trust has not been established with the target server.

```

Certificate chain information:
Certificate [0]
Subject DN: CN=localhost, OU=controller1, O=ibm, C=us

```

Issuer DN: OU=controllerRoot, O=4e08be6d-173c-4015-a747-b9d0dc7844f1,
DC=com.ibm.ws.collective
Serial Number: 4,768,750,201,272
Expires: 8/22/20 8:37 AM
SHA-1 digest: D4:EC:C2:30:B0:AA:DB:78:B0:51:99:95:D8:2F:65:98:C9:A5:3D:B2
MD5 digest: EB:D3:89:F9:F8:6D:F0:35:5F:C6:A6:F6:65:C7:29:D3

Certificate [1]

Subject DN: OU=controllerRoot, O=4e08be6d-173c-4015-a747-b9d0dc7844f1,
DC=com.ibm.ws.collective
Issuer DN: OU=controllerRoot, O=4e08be6d-173c-4015-a747-b9d0dc7844f1,
DC=com.ibm.ws.collective
Serial Number: 4,761,813,674,008
Expires: 8/17/40 8:37 AM
SHA-1 digest: 52:26:D4:EE:C3:C1:B1:8C:EA:53:47:47:78:7B:4F:71:C8:D7:37:99
MD5 digest: EB:3A:6B:6F:3A:DF:F8:2E:1B:B0:67:D6:02:2D:15:0A

Do you want to accept the above certificate chain? (y/n) y
Server member1 successfully unregistered.

Attempting to remove resources for the collective from the server...
The resources for collective membership were successfully removed.
Removing all administrative metadata from the collective repository...
This may take a while.
Successfully completed the MBean request to the controller.

Please update the server.xml and remove any of the following elements:

```
<featureManager>  
  <feature>collectiveController-1.0</feature>  
  <feature>collectiveMember-1.0</feature>  
</featureManager>  
<collectiveMember ... />  
<hostAuthInfo ... />
```

Update the member1 server.xml to remove the member information. To update the server.xml, remove the include to the collective-join-include.xml and add the collectiveMember-1.0 feature.

Example 5-30 shows the updated server configuration of the member1 server (now a stand-alone server).

Example 5-30 Update member1 to be a stand-alone server

```
<?xml version="1.0" encoding="UTF-8"?>  
<server description="new server">  
  
  <!-- Enable features -->  
  <featureManager>  
    <feature>webProfile-7.0</feature>  
    <feature>collectiveMember-1.0</feature>  
  </featureManager>  
  
  <!-- To access this server from a remote client add a host attribute to the  
  following element, e.g. host="*" -->  
  <httpEndpoint id="defaultHttpEndpoint"
```

```
    host="*"
    httpPort="9082"
    httpsPort="9445" />
</server>
```

Now, the member1 server is packaged for deployment by using the **server package** command as shown in Example 5-31.

Example 5-31 Packaging the member1 server for deployment

```
$ server package member1 --archive=/home/itsouser/member1-deployment-package.zip
--include=minify
```

The archive parameter specifies where to create the package. The `--include=minify` option specifies that the package should include the minimum set of Liberty binary files and all user files for the server being packaged. For information about the **server package** command, see 4.4.1, “Packaging a Liberty server” on page 62.

Before packages can be deployed to a simulated *remote host*, the collective controller needs to be updated for read and write permission to the directory simulating the remote host. To do this, the **collective updateHost** command is used as in Example 5-32.

Example 5-32 Update the collective controller with details about the remote host

```
$ collective updateHost localhost --host=localhost --port=9444 --user=admin
--password=adminPwd --rpcUser=itsouser --rpcUserPassword=itsouserPwd
--hostReadPath=/home/itsouser/remoteHost --hostWritePath=/home/itsouser/remoteHost
--hostJavaHome=/usr/lib/java
```

The directory must be set as readable and writable for the controller. The controller is also provided with the local OS user credentials used to operate on the remote host directories. The `hostJavaHome` is set to the Java home directory for the machine.

Deploying packages with the deploy members script

To deploy the collective member to the collective, the sample script `deployMembers` are used. The script can be located at the following website:

https://developer.ibm.com/wasdev/downloads/#asset/scripts-jython-Deploy_Collective_Members

Details about how to use the script can be found in the `readme` file that is shipped with the script. The `rpcUser` and `rpcUserPassword` represent the credentials for the user on the remote host. In this example, the credentials of the logged in user are used. The `/home/itsouser` directory needs to have write permissions for the `rpcUser`. The script is downloaded and unpacked into a local directory. The script is executed by using the **jython** command as shown in Example 5-33.

Example 5-33 Using the `deployMembers` script to deploy member1 to the collective

```
$ jython deployMembers.py --zipFile=/home/itsouser/member1-deployment-package.zip
--installDir=/home/itsouser/remote_host --installHost=localhost --rpcUser=itsouser
--rpcUserPassword=itsouserPwd
--truststore=LIBERTY_CTRL/wlp/usr/servers/controller1/resources/security/trust.jks
--truststorePassword=keystorePwd --host=localhost --port=9444 --user=admin
--password=adminPwd
```

The script in Example 5-33 on page 99 performs the following actions:

1. Connect to the server.
2. Register the host system if necessary.
3. Copy the deployment package from the controller host to the member host.
4. Extract the deployment package.
5. Join the member to the collective.
6. Copy the relevant security information from the controller to the member configuration.
7. Start the member1 server.

The output from the **deployMembers** command should display as shown in Example 5-34.

Example 5-34 Output from the deployMembers command

```
Connecting to the server...
Successfully connected to the server "localhost:9444"
Assigning host context for: localhost
The host has already been registered, calling updateHost instead.
The host localhost connection information is configured.
Loading and expanding member1-deployment-package.zip on target machine location
/home/itsouser/remoteHost

Member member1 join the collective
Uploading needed security files to member1
Starting member member1
Server member1 started successfully
```

A new Liberty installation is created in the `/home/itsouser/remoteHost` directory (`REMOTE_WLP_INSTALL_DIR`). The logs for the server can be found in the `REMOTE_WLP_INSTALL_DIR/wlp/usr/servers/member1/logs` directory. The `messages.log` should show that the server start was complete, as shown in Example 5-35.

Example 5-35 Output of the messages.log of the member1 server

```
CWWKX8055I: The collective member has established a connection to the collective
controller.
```

Deploying packages using the Admin Center

All deployments only occur against a stand-alone Liberty server, so the server needs to be removed from the collective. Stop the member1 server with the **server stop** command and remove the member from the collective by using the **collective remove** command (as shown in Example 5-29 on page 97).

To use the Admin Center for deploying packages, the Admin Center feature is added to the controller1 server configuration, as shown in Example 5-36.

Example 5-36 Add the Admin Center feature to the controller1 server.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

  <!-- Enable features -->
  <featureManager>
    <feature>webProfile-7.0</feature>
    <feature>adminCenter-1.0</feature>
  </featureManager>
```

```

<!-- To access this server from a remote client add a host attribute to the
following element, e.g. host="*" -->
<httpEndpoint id="defaultHttpEndpoint"
             httpPort="9081"
             httpsPort="9444" />

<include location="{server.config.dir}/collective-create-include.xml"/>

</server>

```

Note: If the Admin Center feature cannot be loaded, check to see if the Admin Center feature is installed in the WLP_INSTALL_DIR. The feature can be installed from a Liberty repository as noted in 3.6.1, "Installing assets by using the installUtility command" on page 48.

Start the Admin Center by pointing a web browser to <https://localhost:9444/adminCenter>. The Admin Center toolbox should now show the Deploy tool as it is running in the context of a collective controller, as shown in Figure 5-15.

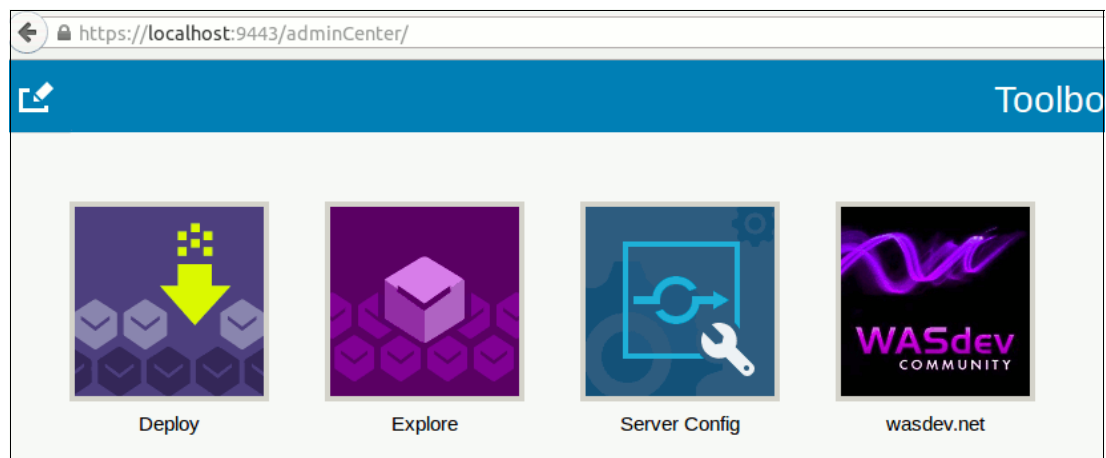


Figure 5-15 Admin Center with Deploy tool enables

To deploy the server package, select the **Deploy tool** and select the **localhost** entry to add it to the selected hosts list box, as shown in Figure 5-16.

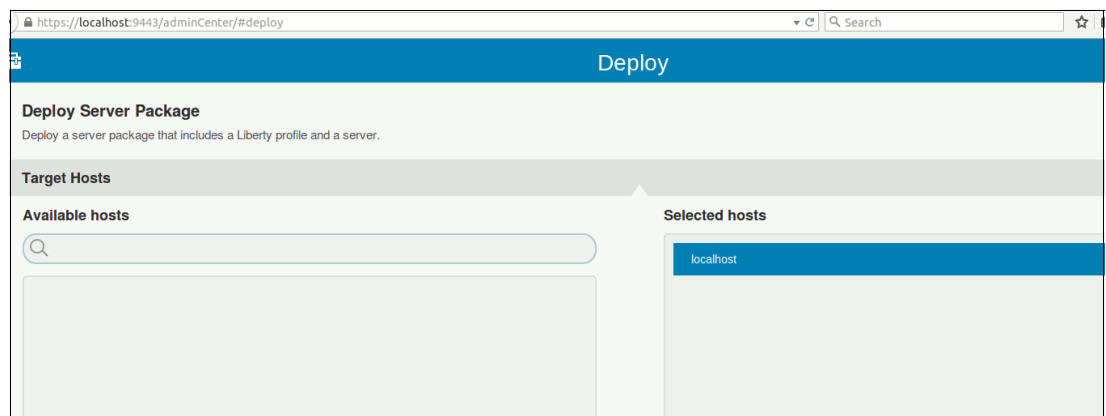
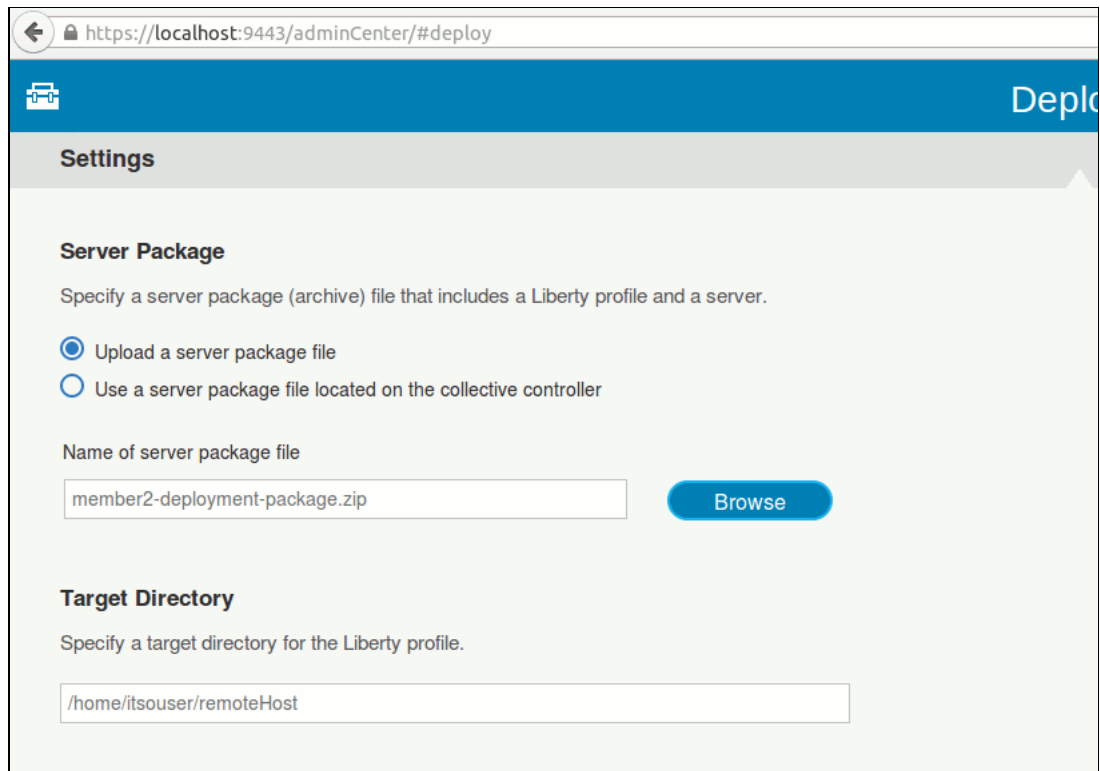


Figure 5-16 Select the host for deployment of the package

Add the package to be deployed by scrolling down on the same page and browse for the member2-deploy-pacakge.zip file. Then, specify the target directory, /home/itsouser/remoteHost, as shown in Figure 5-17.



The screenshot shows a web browser window with the URL `https://localhost:9443/adminCenter/#deploy`. The page has a blue header with a home icon and the word "Deploy". Below the header is a "Settings" section. Under "Settings", there is a "Server Package" section with the instruction "Specify a server package (archive) file that includes a Liberty profile and a server." There are two radio buttons: "Upload a server package file" (selected) and "Use a server package file located on the collective controller". Below this is a text input field containing "member2-deployment-package.zip" and a blue "Browse" button. The "Target Directory" section has the instruction "Specify a target directory for the Liberty profile." and a text input field containing "/home/itsouser/remoteHost".

Figure 5-17 Select package to deploy and target for the deployment

Configure the security credentials for deployment by scrolling down the same page and enter the keystore password and the remote management credentials. Use the connection method and credentials configured for each target host. You also need to enter the Liberty controller1 admin password because access to all operations on the collective controller is secured with the administrative role. See Figure 5-18 on page 103.

KeyStore password

Confirm KeyStore password

Remote Management Credentials

Specify how the collective controller will connect to each target host when managing the servers.

Use the connection method and credentials configured for each target host
 Use automatically-generated SSH keys for each server
 Use the following operating system user name and password on each target host

User name

Password

Confirm password

Your Liberty Administrative Password

The operation to join the deployed servers to the collective is run with your Liberty administrative user name and password.

Password

Figure 5-18 Deploy the package

To start the deployment, click **Deploy**. When prompted, close the window, as shown in Figure 5-19.

Uploaded server package file and started background deployment task

Figure 5-19 Close the deployment window

The deployment runs as a background task. To track the status of the background deployment task, select the **Background Tasks** icon in the upper right corner of the Admin Center. When the icon is clicked, it shows the background tasks that are running and completed.

If the deployment is successful, the task should show as successful (see Figure 5-20).



Figure 5-20 Deploy completed successfully

5.4.6 Configuring collective controller replica sets

A *replica set* provides highly available management capabilities for a Liberty administrative domain. A replica set is a set of collective controllers that are configured to work together. Each replica contains all the processed repository updates from the other replicas within the set. Therefore, there is no need for a member to connect to a particular collective controller each time that it interacts with the collective. Any of the collective controllers that are configured in the replica set can provide the same data. Each server that is part of the replica set should have the collective controller feature enabled as shown in Figure 5-21.

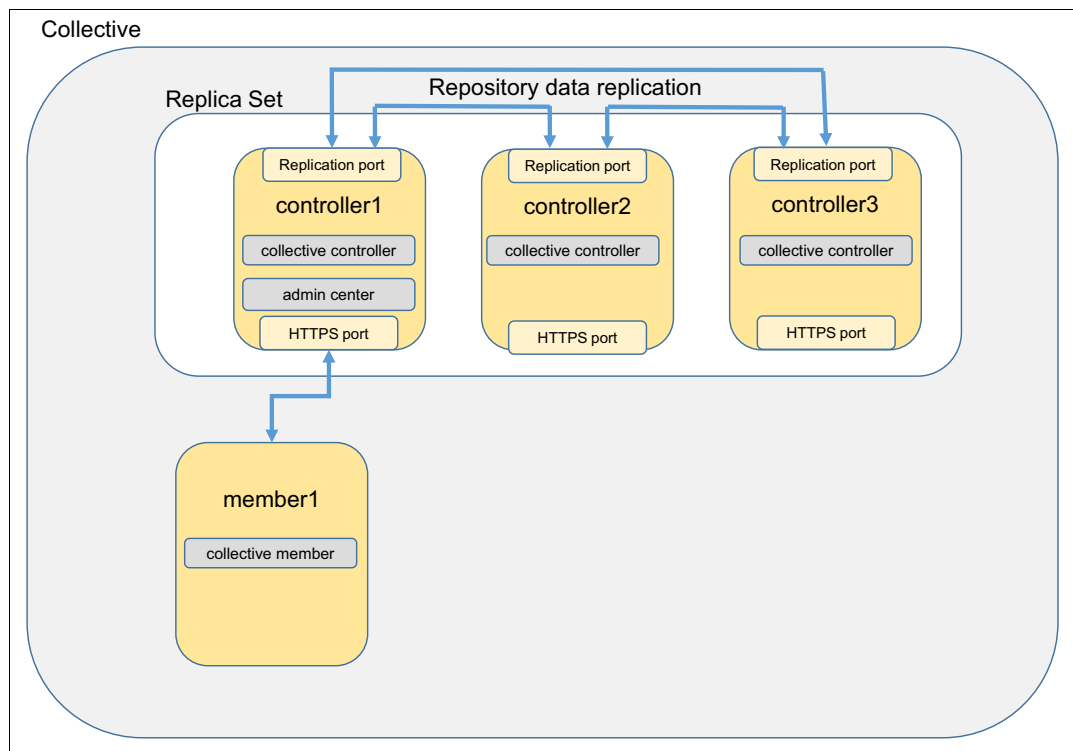


Figure 5-21 Creating replica sets for the Liberty profile collective controller

In a production environment, a replica set should have at least three replicas, preferably on different hosts. When the replicas are on different hosts, they can use the same port numbers. The example created in this section creates three replicas but installs them on the same host for simplicity. As a consequence, the port numbers for each collective controller in the replica set need to be different.

To create a replica set, the following procedure needs to be followed:

1. Start the initial collective controller (this example reuses the controller created earlier in this chapter).

2. Create the replicas of the initial collective controller.
3. Replicate the administrative domain security information.
4. Activate the replicas.

Start the initial collective controller

To create a replica set the initial controller, `controller1`, must be running. Ensure that the initial collective controller is started by using the `server status` command as shown in Example 5-37.

Example 5-37 Checking the status of controller1

```
$ server status controller1
Server controller1 is running with process ID 3527.
$
```

If it is not started, start it with the `server start` command, as shown in Example 5-38.

Example 5-38 Start the initial collective controller

```
$ server start controller1
Starting server controller1
Server controller1 started with process ID 25187
$
```

Create the replica collective controller

A new server instance is created to act as a collective controller replica. The new server is created by using the `server create` command, as shown in Example 5-39.

Example 5-39 Create the replica collective controller

```
$ server create controller2
Server controller2 created.
$
```

Replicate the administrative domain security information

For the new server, `controller2`, to become part of the replica set, the administrative domain security configuration needs to be copied from the initial controller, `controller1`. To do this, use the `collective replicate` command. The parameters for the replicate command are noted in the following list:

- ▶ Hostname of the host running the initial collective controller (`localhost`, in our example)
- ▶ Port the initial host is listening on
- ▶ Administrative user for the initial controller
- ▶ Password for the administrative user
- ▶ Password for the keystore of the new replica
- ▶ Optionally, the configuration file where the replica configuration is stored

To create the replica of the initial controller, controller1, the **collective replicate** command is issued, as shown in Example 5-40. A copy of the collective information is replicated from the controller1 configuration directory to the controller2 configuration directory. Note that SERVER_CONFIG_DIR in the script (Example 5-40) is for the new server configuration replica, controller2.

Example 5-40 Running the collective replicate command

```
$ collective replicate controller2 --host=localhost --port=9444 --user=admin  
--password=adminPwd --keystorePassword=keystorePwd  
--createConfigFile=SERVER_CONFIG_DIR/collective-replica-include.xml
```

The **collective replicate** command copies the collective controllers administrative domain security credentials to the new replica. The certificate chain needs to be accepted when prompted for response. The output of the **collective replicate** command is shown in Example 5-41.

Example 5-41 Output of the collective replicate command

```
Replicating the target collective controller localhost:9444...  
This may take a while.
```

```
SSL trust has not been established with the target server.
```

```
Certificate chain information:
```

```
Certificate [0]
```

```
Subject DN: CN=localhost, OU=controller1, O=ibm, C=us
```

```
Issuer DN: OU=controllerRoot, O=71a881f1-0b21-48d0-8fcf-d76e69793865,  
DC=com.ibm.ws.collective
```

```
Serial Number: 6,026,992,827,355
```

```
Expires: 8/12/20 9:13 AM
```

```
SHA-1 digest: 6B:C5:E3:C9:0A:83:B5:AD:AD:8B:2C:26:2A:72:EF:D7:45:25:E6:D4
```

```
MD5 digest: 05:25:E2:21:D8:F9:40:10:FB:D0:DB:85:BD:70:2C:B6
```

```
Certificate [1]
```

```
Subject DN: OU=controllerRoot, O=71a881f1-0b21-48d0-8fcf-d76e69793865,
```

```
DC=com.ibm.ws.collective
```

```
Issuer DN: OU=controllerRoot, O=71a881f1-0b21-48d0-8fcf-d76e69793865,
```

```
DC=com.ibm.ws.collective
```

```
Serial Number: 6,018,568,652,812
```

```
Expires: 8/7/40 9:13 AM
```

```
SHA-1 digest: 56:AC:E6:B1:37:43:1A:D7:AA:CC:1C:60:2E:7B:0B:57:FF:C4:3A:41
```

```
MD5 digest: 83:4C:80:D3:B7:98:3E:EE:56:DC:11:5C:A7:5A:2D:01
```

```
Do you want to accept the above certificate chain? (y/n) y
```

```
Successfully completed MBean request to the controller.
```

```
Successfully replicated the controller as server controller2
```

```
Add the following lines to the server.xml to enable:
```

```
<include location="{server.config.dir}/collective-replica-include.xml" />
```

Please ensure administrative security is configured for the new server exactly as the current collective controller. Also set the password for

the collectiveRootKeys to the correct password.
\$

To continue configuring the new replica, the server.xml of the controller2 server needs to be updated. The update covers the correct port information and the include to the collective replica information. See Example 5-42.

Example 5-42 Update the controller2 server.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>webProfile-7.0</feature>
    </featureManager>

    <!-- To access this server from a remote client add a host attribute to the
    following element, e.g. host="*" -->
    <httpEndpoint id="defaultHttpEndpoint"
        host="*"
        httpPort="9083"
        httpsPort="9446" />

    <include location="{server.config.dir}/collective-replica-include.xml" />
</server>
```

The configuration of the newly created replica, controller2, needs to be updated with the correct administrative user credentials and replication ports. To do so, the collective-replica-include.xml file needs to be updated. The update includes the correct replication ports, the admin user and password from the controller1 configuration, and also the password for the collectiveRootKeys is copied from the controller1 server configuration. See Example 5-43.

Example 5-43 Update the replication ports in the collect-replica-include.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<server description="This file was generated by the 'collective replicate' command
on 2015-08-14 14:01:03 EDT.">
    <featureManager>
        <feature>collectiveController-1.0</feature>
    </featureManager>

    <!-- Define the host name for use by the collective.
    If the host name needs to be changed, the server should be
    removed from the collective and re-joined or re-replicated. -->
    <variable name="defaultHostName" value="localhost" />

    <!-- Configuration of the collective controller replica.
    TODO: If this replica is on the same host as the original controller,
    change the replicaPort.
    TODO: If the target controller's replica port is not 10010
    (the default) change the value in replicaSet. -->
    <collectiveController replicaPort="10011"
        replicaSet="localhost:10010"
        isInitialReplicaSet="false" />
```

```

<!-- TODO: Define the security configuration exactly as defined in the
      target controller from which this was replicated. -->
<quickStartSecurity userName="admin" userPassword="adminPwd" />

<!-- clientAuthenticationSupported set to enable bidirectional trust -->
<ssl id="defaultSSLConfig"
     keyStoreRef="defaultKeyStore"
     trustStoreRef="defaultTrustStore"
     clientAuthenticationSupported="true" />

<!-- inbound (HTTPS) keystore -->
<keyStore id="defaultKeyStore" password="controller2Pwd"
          location="${server.config.dir}/resources/security/key.jks" />

<!-- inbound (HTTPS) truststore -->
<keyStore id="defaultTrustStore" password="controller2Pwd"
          location="${server.config.dir}/resources/security/trust.jks" />

<!-- server identity keystore -->
<keyStore id="serverIdentity" password="controller2Pwd"
          location="${server.config.dir}/resources/collective/serverIdentity.jks" />

<!-- collective truststore -->
<keyStore id="collectiveTrust" password="controller2Pwd"
          location="${server.config.dir}/resources/collective/collectiveTrust.jks" />

<!-- collective root signers keystore
      TODO: set password to the collectiveRootKeys password in the
      original controller -->
<keyStore id="collectiveRootKeys" password="controller1Pwd"
          location="${server.config.dir}/resources/collective/rootKeys.jks" />

</server>

```

The replica can now be started by using the **server start** command, as shown in Example 5-44.

Example 5-44 Starting the controller2 replica

```

$ server start controller2
Starting server controller2
Server controller2 started with process ID 20255.
$

```

Ensure that the new replica, controller2, has started correctly by verifying that the initial controller, controller1, can communicate with the new controller2 replica. Check the messages.log file of the controller1 server. The file is found at SERVER_CONFIG_DIR/logs. The audit message, CWKX6009I, should be present in the logs, as shown in Example 5-45 on page 109.

Example 5-45 Message indicating that the replicas can connect to each other

```
CWWKX6009I: The collective controller successfully connected to replica
127.0.0.1:10011. Current active replica set is [127.0.0.1:10010]. The configured
replica set is [127.0.0.1:10010]. The connected standby replicas are
[127.0.0.1:10011].
```

Activate the replicas

The new replica needs to be activated. To do this, the collective `addReplica` command is used. The command takes as parameters the host:port of the replica to activate. The command also takes the host, port, user, and password of the original controller (see Example 5-46).

Example 5-46 Using the collective addReplica command to activate the replica

```
$ collective addReplica localhost:10011 --host=localhost --port=9444 --user=admin
--password=adminPwd
```

```
Adding the endpoint to the replica set...
```

```
SSL trust has not been established with the target server.
```

```
Certificate chain information:
```

```
Certificate [0]
```

```
Subject DN: CN=localhost, OU=controller1, O=ibm, C=us
```

```
Issuer DN: OU=controllerRoot, O=b222ec67-3a88-4bd9-9c6d-57ad19d0187e,
DC=com.ibm.ws.collective
```

```
Serial Number: 8,338,143,633,065
```

```
Expires: 8/19/20 11:13 AM
```

```
SHA-1 digest: 1B:CE:8C:7F:37:A6:5B:A3:06:DF:B9:C7:42:2D:5E:26:22:55:58:C7
```

```
MD5 digest: BF:58:7B:7E:B1:A2:40:F0:A4:1C:86:D0:C2:CA:5E:32
```

```
Certificate [1]
```

```
Subject DN: OU=controllerRoot, O=b222ec67-3a88-4bd9-9c6d-57ad19d0187e,
DC=com.ibm.ws.collective
```

```
Issuer DN: OU=controllerRoot, O=b222ec67-3a88-4bd9-9c6d-57ad19d0187e,
DC=com.ibm.ws.collective
```

```
Serial Number: 8,333,307,608,526
```

```
Expires: 8/14/40 11:13 AM
```

```
SHA-1 digest: 2E:8A:2B:38:31:18:9A:CC:C8:55:6E:AD:C8:C8:E5:DC:22:8A:DC:42
```

```
MD5 digest: FE:DF:00:F3:53:C5:79:E5:80:0E:4D:6A:54:04:47:0E
```

```
Do you want to accept the above certificate chain? (y/n) y
```

```
Successfully added replica endpoint localhost:10011 to the replica set.
```

The messages.log of the two controllers should show that the replicas are communicating with each other. The log for the controller1 server should contain the entries shown in Example 5-47, verifying that the replica set contains two servers.

Example 5-47 Controller1 messages.log file showing the replica set communication

```
CWWKX6013I: The collective controller state is {S_PROPOSING}, last proposed
command is 94, the last accepted command is 94, the last executed command is 94
and the log is 94.
CWWKX6009I: The collective controller successfully connected to replica
127.0.0.1:10011. Current active replica set is [127.0.0.1:10010]. The configured
replica set is [127.0.0.1:10010]. The connected standby replicas are
[127.0.0.1:10011].
CWWKX6013I: The collective controller state is {S_PROPOSING}, last proposed
command is 97, the last accepted command is 97, the last executed command is 97
and the log is 97.
CWWKX6015I: A request to change the active collective controller replica set was
received and is now processing. The current active replica set is
{127.0.0.1:10010}. The requested new active replica set is
{127.0.0.1:10010,127.0.0.1:10011}.
CWWKX6016I: The active collective controller replica set changed successfully. The
current active replica set is {127.0.0.1:10010,127.0.0.1:10011}. The previous
active replica set was {127.0.0.1:10010}.
```

To further verify communication, check the controller2 messages.log file. It should contain the entries shown in Example 5-48.

Example 5-48 Controller2 messages.log showing the replica set communication

```
CWWKX6016I: The active collective controller replica set changed successfully. The
current active replica set is {127.0.0.1:10011,127.0.0.1:10010}. The previous
active replica set was {127.0.0.1:10010}.
CWWKX6014I: This collective controller replica finished synchronizing the data
with the other replicas. The log is 100.
CWWKX6011I: The collective controller is ready, and can accept requests. The
leader is 127.0.0.1:10010. Current active replica set is [127.0.0.1:10011,
127.0.0.1:10010]. The configured replica set is [127.0.0.1:10011,
127.0.0.1:10010].
```

Note: To support high availability, there must always be an odd number of controllers within a replica set. A third controller should be added following the instructions in this section, modifying port numbers to prevent port conflicts.

5.4.7 Setting up a Liberty server cluster

A Liberty profile server can be configured into a server cluster for more efficient management of servers that are hosting the same applications. The cluster provides high availability and scalability for the applications. To enable a cluster, the clusterMember-1.0 feature needs to be available in the Liberty installation. If this feature is not installed, it can be installed from the repository as documented in 3.6.1, “Installing assets by using the installUtility command” on page 48.

The additional cluster members are added to the Liberty topology, as shown in Figure 5-22.

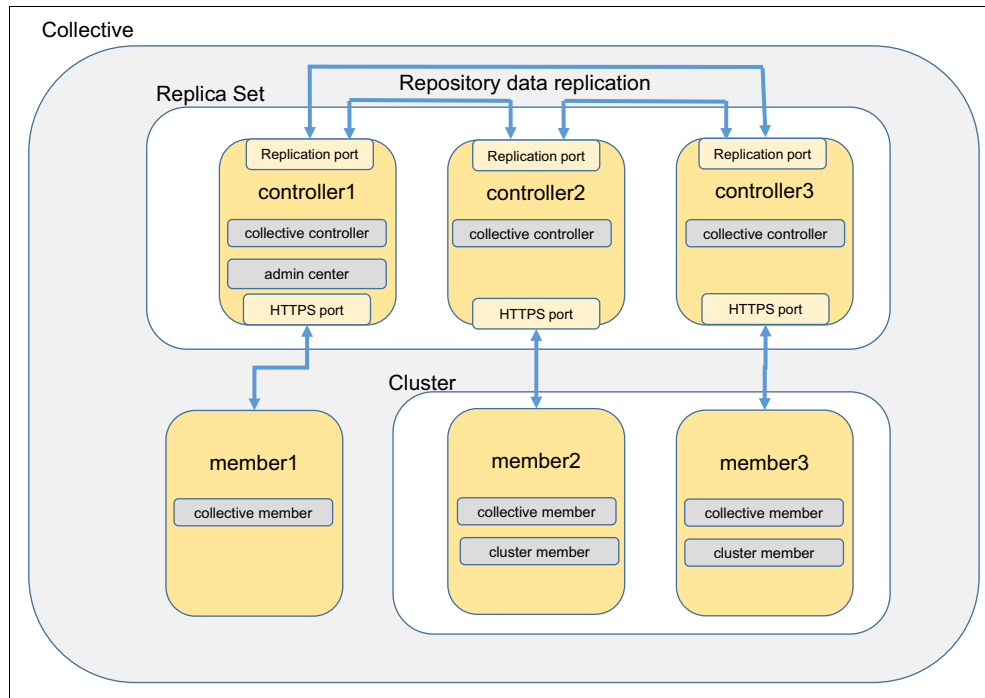


Figure 5-22 Adding clusters to the Liberty profile collective

The following sections show how to create a cluster of two servers, package the servers for deployment, and deploy the cluster to the collective. To create and deploy a cluster, the following procedure is used:

- ▶ Create the cluster member servers.
- ▶ Create the deployment package for the cluster member.
- ▶ Deploy the cluster member package to the collective.

Create the cluster member servers

Create the cluster member server, `clusterMember1`, by using the **server create** command as shown in Example 5-49.

Example 5-49 Create the `clusterMember1` server

```
$ server create clusterMember1
server clusterMember1 created.
$
```

Repeat the operation for the second cluster member, `clusterMember2`.

Edit the `server.xml` for both of the new servers to include the cluster member feature. As the servers are all running on the same host, the ports for the server need to be changed for each of the new servers (ports 9084 and 9447 are used for `clusterMember1` and ports 9085 and 9448 are used for `clusterMember2`). The **clusterMember** element is used to specify the cluster name.

The updated `server.xml` is shown in Example 5-50 on page 112. Repeat the creation process and configuration updates for `clusterMember2` with the correct ports.

Example 5-50 Update the cluster members server.xml to include the cluster member feature

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

  <!-- Enable features -->
  <featureManager>
    <feature>webProfile-7.0</feature>
    <feature>clusterMember-1.0</feature>

  </featureManager>

  <!-- To access this server from a remote client add a host attribute to the
  following element, e.g. host="*" -->
  <httpEndpoint id="defaultHttpEndpoint"
    hosts="*"
    httpPort="9084"
    httpsPort="9447" />

  <clusterMember name="MyCluster" />
</server>
```

Create the cluster member deployment package

The servers can be packaged up for deployment by using the `server package` command. Specify the `--include=minify` information so that the packages contain only the minimal set of binary files and the user files, as shown in Example 5-51.

Example 5-51 Package up the cluster members for deployment

```
$ server package clusterMember1
--archive=/home/itsouser/clusterMember1-deploy-package-zip --include=minify
```

Repeat the process for the `clusterMember2` server.

Deploy the cluster member package to the collective

To deploy the packaged cluster member to the collective, the `deployMembers` script documented in 5.4.5, “Adding members to the Liberty profile collective” on page 97 is used. The command to deploy the package is shown in Example 5-52.

Example 5-52 Using the deployMembers script to deploy the cluster to the collective

```
$ jython deployMembers.py
--zipFile=/home/itsouser/clusterMember1-deployment-package.zip
--installDir=/home/itsouser/remoteHost --installHost=localhost --rpcUser=itsouser
--rpcUserPassword=itsouserPwd
--truststore=WLP_INSTALL_DIR/wlp/usr/servers/controller1/resources/security/trust.
jks --truststorePassword=keystorePwd //host=localhost --port=9444 --user=admin
--password=adminPwd
```

The output from the `deployMembers` script should be similar to the output shown in Example 5-53. The output shows that the package has been deployed to the remote locations, unpacked, the cluster member joined to the collective, and finally that the cluster member has been started.

Example 5-53 Output from the `deployPackage` command

```
Connecting to the server...
Successfully connected to the server "localhost:9444"
Assigning host context for: localhost
The host has already been registered, calling updateHost instead.
The host localhost connection information is configured.
Loading and expanding cluster-deployment-package.zip on target machine location
/home/itsouser/remoteHost
```

```
Member clusterMember1 join the collective
Uploading needed security files to clusterMember1
Starting member clusterMember1
Server clusterMember1 started with process ID 25189
```

The `clusterMember1` files are copied to the `/home/itsouser/remoteHost`. The log files can be found in the `SERVER_CONFIG_DIR` logs directory of this new server, `clusterMember1`. The `messages.log` should show that the `clusterMember1` server start was complete, as shown in Example 5-35 on page 100. The log should also show that the cluster information was published to the collective repository on the `controller1` server.

Example 5-54 Output of the `messages.log` of the `clusterMember1` server

```
CWWKX8114I: The server's paths were successfully published to the collective
repository.
CWWKX8112I: The server's host information was successfully published to the
collective repository.
```

The deployment should be repeated for the `clusterMember2` server.

Using scripts to work with clusters

Sample scripts are available on the WASdev website for managing clusters. The scripts can be accessed through the following website:

<https://developer.ibm.com/wasdev/downloads/#filter/sortby=wlpInformation.featuredW eight;sortorder=desc>

Scripts are available to start and stop clusters, list clusters, and list their cluster members. Scripts that are available can also generate a plug-in for the cluster, for example, to enable an HTTP server to route requests across members in a cluster. The plug-in is required for the dynamic routing capability that is explained in more detail in 10.1, “Introduction to Intelligent Management” on page 174.



Accessing databases

When an application or WebSphere component requires access to a relational database, that database must be defined to WebSphere as a data source. Two basic definitions are required:

- ▶ A Java Database Connectivity (JDBC) provider definition describes a vendor-provided JDBC driver. It includes the type of database access that it provides and the location of the files that provide the implementation.
- ▶ A data source definition defines which JDBC provider to use, the name, and location of the database, and other connection properties.

This chapter provides information about the various considerations for accessing databases from WAS Liberty.

This chapter covers the following topics:

- ▶ JDBC resources
- ▶ Steps to define access to a database
- ▶ Configuring data sources in Liberty
- ▶ Configuring connection pooling properties in Liberty
- ▶ Accessing MongoDB databases
- ▶ Logging data source activity
- ▶ Using the timed operations feature to monitor database operations

6.1 JDBC resources

The JDBC application programming interface (API) provides a programming interface for data access of relational databases from the Java programming language. Liberty supports the JDBC API with the `jdbc-4.1` feature. The following sections explain how to create and configure data source objects for use by JDBC applications. This is the only method to connect to a database if you intend to use connection pooling and distributed transactions.

The following database platforms are supported for Liberty:

- ▶ IBM DB2®
- ▶ Oracle
- ▶ Sybase
- ▶ IBM Informix®
- ▶ Microsoft SQL Server
- ▶ Apache Derby (test and development only)
- ▶ MySQL
- ▶ Sybase
- ▶ SolidDB
- ▶ Third-party vendor JDBC data source using SQL99 standards

Liberty supports the use of MongoDB and CouchDB Java driver.

For a current detailed list of the databases supported, see the requirements for Liberty at the following website:

<http://www.ibm.com/support/docview.wss?uid=swg27038218#libcore-855>

6.1.1 JDBC providers and data sources

A *data source* represents a real-world source of data, such as a relational database. When a data source object is registered with a Java Naming and Directory Interface (JNDI) naming service, an application can retrieve it from the naming service and use it to make a connection to the associated database.

Information about the data source and how to locate it, such as its name, the server on which it resides, its port number, and so on, is stored in the form of *properties* on the `DataSource` object. Storing this information in this manner makes an application more portable. Portability is increased because it does not need to hardcode a driver name, which often includes the name of a particular vendor. It also makes maintaining the code easier. For example, if the data source is moved to a different server, all that needs to be done is to update the relevant property in the data source. None of the code using that data source needs to be managed.

To increase application performance and reduce workload on the database, connections to it are typically pooled. In other words, when the application closes the connection, the connection is returned to a connection pool, rather than being destroyed.

Data source *classes* and JDBC *drivers* are implemented by the data source vendor. By configuring a JDBC provider, you provide information about the set of classes that are used to implement the data source and the database driver. Also, you provide the environment settings for the `DataSource` object. A driver can be written purely in the Java programming language or in a mixture of the Java programming language and the Java Native Interface (JNI) native methods.

6.1.2 WebSphere support for data sources

The following programming model is used for accessing a data source:

1. An application retrieves a DataSource object from the JNDI naming space.
2. After the DataSource object is obtained, the application code calls the `getConnection()` method on the data source to get a Connection object. The connection is obtained from a pool of connections.
3. After the connection is acquired, the application sends SQL queries or updates to the database.

6.2 Steps to define access to a database

The following steps are involved in defining access to a database:

1. Verify that connection to the database server is supported by WebSphere Application Server.
2. Ensure that the database is created and can be accessed by the systems that use it.
3. Ensure that the JDBC provider classes are available on the systems that access the database. If you are not sure which classes are required, consult the documentation from the provider.
4. Configure a JDBC provider in the server configuration. The JDBC provider gives the class path of the data source implementation class and the supporting classes for database connectivity. This is vendor-specific.
5. Configure a data source. The JDBC data source encapsulates the database-specific connection settings. You can configure many data sources that use the same JDBC provider.

6.3 Configuring data sources in Liberty

A data source associated with different JDBC providers can be configured for database connectivity in Liberty. The JDBC providers supply the driver implementation classes that are required for JDBC connectivity with your specific vendor database.

Data sources are provided by JDBC drivers and come in the following varieties:

▶ `javax.sql.DataSource`

This is the basic form of a data source. It does not provide interoperability that enhances connection pooling and cannot participate as a two-phase capable resource in transactions involving multiple resources.

▶ `javax.sql.ConnectionPoolDataSource`

This type of data source is enabled for connection pooling. It cannot participate as a two-phase capable resource in transactions involving multiple resources.

▶ `javax.sql.XADataSource`

This type of data source is both enabled for connection pooling and can participate as a two-phase capable resource in transactions involving multiple resources.

Use the following procedure to configure a data source:

1. Liberty needs to be informed where to find the JDBC driver. In the server.xml file, define a shared library pointing to the location of your JDBC driver JAR or compressed files.

JAR files: The JAR files that are used for accessing your database are not provided as part of the Liberty run time. This example uses a DB2 database, which can be downloaded from the following website:

<http://www.ibm.com/support/docview.wss?uid=swg21363866>

Library placement: For easier Liberty server packaging, it is recommended to place driver JAR files in the Liberty shared resources folder (WLP_HOME\usr\shared\resources). This folder is referred to by default in the server variable `${shared.resource.dir}`. Create the DB2 subdirectory in that folder and place the JAR files in that directory.

Example 6-1 defines a shared library for DB2 driver jar files.

Example 6-1 Shared library for DB2

```
<library id="DB2JCC4Lib">
  <fileset dir="${shared.resource.dir}/DB2" includes="db2jcc4.jar
db2jcc_license_cisuz.jar.jar"/>
</library>
```

2. Define a data source using the JDBC driver. Example 6-2 defines a data source for DB2 JDBC driver with default data source type. The terms used in Example 6-2 are defined in the following list:

- The `<dataSource>` configuration element defines a data source.
- The `<jdbcDriver>` element identifies a JDBC driver, and its attribute `libraryRef` identifies the JDBC driver JAR files and native files.
- The `<properties.db2.jcc>` is the data source properties for the IBM Data Server Driver for JDBC and SQLJ for DB2. It is the child of the complex type “dataSource.”

Example 6-2 Data source definition for DB2 JDBC driver

```
<dataSource id="db2" jndiName="jdbc/db2">
  <jdbcDriver libraryRef="DB2JCC4Lib"/>
  <properties.db2.jcc databaseName="SAMPLEDB" serverName="localhost"
portNumber="50000"/>
</dataSource>
```

Example 6-3 defines a data source for the DB2 JDBC driver with `XADataSource` type.

Example 6-3 Data source for DB2 JDBC driver with XADataSource type

```
<dataSource id="db2xa" jndiName="jdbc/db2xa" type="javax.sql.XADataSource">
  <jdbcDriver libraryRef="DB2JCC4Lib"/>
  <properties.db2.jcc databaseName="SAMPLEDB" serverName="localhost"
portNumber="50000"/>
</dataSource>
```

6.3.1 Configuring third-party data sources

This section provides examples of configuring data source elements for commonly used databases.

JAR files: The JAR files that are used for accessing your database are not provided as part of the Liberty run time. This example uses an Oracle database, which can be downloaded from the following website:

<http://www.oracle.com/technetwork/database/features/jdbc/jdbc-drivers-12c-download-1958347.html>

Library placement: For easier Liberty server packaging, it is recommended to put driver JAR files in the Liberty shared resources folder (`WLP_HOME\usr\shared\resources`). This folder is referred to by default by the server variable `${shared.resource.dir}`. Create the oracle subdirectory in that folder and place the JAR files in that directory.

Example 6-4 configures a shared library and a data source for an Oracle database.

Example 6-4 Data source for Oracle database

```
<dataSource id="oracle" jndiName="jdbc/oracle">
  <jdbcDriver libraryRef="OracleLib"/>
  <properties.oracle URL="jdbc:oracle:thin:@//localhost:1521/SAMPLEDB"/>
</dataSource>

<library id="OracleLib">
  <fileset dir="${shared.resource.dir}/oracle" includes="*.jar"/>
</library>
```

JAR files: The JAR files that are used for accessing your database are not provided as part of the Liberty run time. This example uses a Derby database. JAR files that are provided as part of your database run time can be downloaded from the following website:

https://db.apache.org/derby/derby_downloads.html

Library placement: For easier Liberty server packaging, it is recommended to place driver JAR files in the Liberty shared resources folder (`WLP_HOME\usr\shared\resources`). This folder is referred to by default by the server variable `${shared.resource.dir}`. Create the Derby subdirectory in that folder and put the JAR files in that directory.

Example 6-5 configures a shared library and data source for an embedded Derby database.

Example 6-5 Data source for Derby database

```
<dataSource id="derbyEmbedded" jndiName="jdbc/derbyEmbedded">
  <jdbcDriver libraryRef="DerbyLib"/>
  <properties.derby.embedded databaseName="C:/databases/SAMPLEDB"
createDatabase="create"/>
</dataSource>

<library id="DerbyLib">
  <fileset dir="${shared.resource.dir}/derby"/>
</library>
```

Example 6-6 configures a shared library and data source for a JDBC driver that is not known to Liberty. The JDBC driver is at `C:/Drivers/SampleJDBC/sampleDriver.jar` and provides an implementation of `javax.sql.XADataSource` named `com.ibm.sample.SampleXADataSource`. The JDBC driver also provides vendor-specific data source properties, such as database name, host name, and port.

Example 6-6 Data source for a JDBC driver unknown to Liberty

```
<dataSource id="sample" jndiName="jdbc/sample" type="javax.sql.XADataSource">
  <jdbcDriver libraryRef="SampleJDBCLib"
    javax.sql.XADataSource="com.ibm.sample.SampleXADataSource"/>
  <properties databaseName="SAMPLEDB" hostName="localhost" port="12345"/>
</dataSource>

<library id=SampleJDBCLib">
  <fileset dir="C:/Drivers/SampleJDBC/" includes="sampleDriver.jar"/>
</library>
```

More information about configuring database connectivity in Liberty is provided in the IBM Knowledge Center at the following website:

http://www.ibm.com/support/knowledgecenter/SSEQTP_8.5.5/com.ibm.websphere.wlp.doc/ae/twlp_dep_configuring_ds.html?cp=SSEQTP_8.5.5%2F1-3-11-0-3-2-19-0-0

6.3.2 Application-defined data sources in Liberty

In Liberty, data sources to databases can be defined within the application through annotations or in the deployment descriptor configuration file. Configure a shared library in the `server.xml` configuration file pointing to the location of the JDBC driver jars. Then, using either annotations or in the deployment descriptor file, the data source can be defined in the application. To define a data source, use the following steps:

1. Configure a shared library in the `server.xml` configuration file pointing to the location of the JDBC driver jars.
2. Configure the application's classloader in the `server.xml` configuration file with a `commonLibraryRef` pointing to the shared library.
3. Use either annotations or the deployment descriptor file to define the data source in the application.

Example 6-7 defines a data source in an application using annotations.

Example 6-7 Defining a data source using annotations

```
@DataSourceDefinition(
    name          = "java:comp/env/jdbc/db2",
    className     = "com.ibm.db2.jcc.DB2DataSource",
    databaseName  = "SAMPLEDB",
    serverName    = "localhost",
    portNumber    = 50000,
    properties    = { "driverType=4" },
    user          = "user1"
    password      = "pwd1"
)

public class MyServlet extends HttpServlet {
```



```

    @Resource(lookup="java:comp/env/jdbc/db2")
    DataSource ds;
}

```

Example 6-8 defines a data source in the application using the deployment descriptor in the `web.xml` configuration file.

Example 6-8 Defines a data source using deployment descriptor

```

<data-source>
  <name>java:comp/env/jdbc/db2</name>
  <class-name>com.ibm.db2.jcc.DB2DataSource</class-name>
  <server-name>localhost</server-name>
  <port-number>50000</port-number>
  <database-name>SAMPLEDB</database-name>
  <user>user1</user>
  <password>pwd1</password>
  <property><name>driverType</name><value>4</value></property>
</data-source>

```

More information about application-defined data sources is provided in the IBM Knowledge Center at the following website:

http://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/rwlp_ds_appdefined.html

6.3.3 Runtime data source configuration update in Liberty

In Liberty, a data source is configured by specifying the attributes of the `dataSource` element in the `server.xml` configuration file. Many attributes for data source are updated dynamically at run time. Table 6-1 describes each attribute of the `dataSource` element and shows how the configuration change is applied at run time.

Table 6-1 How configuration update is applied

Attribute name	How the configuration update is applied
<code>beginTranForResultSetScrollingAPIs</code>	The update is effective immediately.
<code>beginTranForVendorAPIs</code>	The update is effective immediately.
<code>commitOrRollbackOnCleanup</code>	The update is effective immediately.
<code>connectionManagerRef</code>	All connections and the connection pool are destroyed. The data source is then managed by the new connection manager.
<code>connectionSharing</code>	The update is applied with each first connection handle in a transaction.
<code>isolationLevel</code>	The update is applied with new connection requests. Current connections retain their isolation level.
<code>jdbcDriverRef</code>	All connections and the connection pool are destroyed. The new JDBC driver is then used.
<code>jndiName</code>	All connections and the connection pool are destroyed. The new JNDI name is then used.

Attribute name	How the configuration update is applied
propertiesRef	If the data source is Derby Embedded, all connections and the connection pool are destroyed before new properties go into effect. For other JDBC drivers, the new properties go into effect with new connection requests.
queryTimeout	The update is effective immediately.
statementCacheSize	The statement cache is resized upon next use.
supplementalJDBCTrace	All connections and the connection pool are destroyed. The new setting is then used.
syncQueryTimeoutWithTransactionTimeout	The update is effective immediately.
transactional	The update is applied to new connections and existing connections not in use from the connection pool.
type	All connections and the connection pool are destroyed. The new setting is then used.

6.4 Configuring connection pooling properties in Liberty

Performance of an application that connects to a database can be greatly affected by the availability of connections to the database and how those connections affect the performance of the database itself. There are no simple rules that tell you how to configure the connection pool properties. Your configuration is highly dependent on application, network, and database characteristics. You must coordinate the values that you specify in Liberty closely with the database administrator.

Remember to include all resources in capacity planning. If 10 applications all connect to a database using separate connection pools of 10 maximum connections, this means that there is a theoretical possibility of 100 concurrent connections to the database. Ensure that the database server has sufficient memory and processing capacity to support this requirement.

Connection pooling for the data sources in Liberty are configured by defining a connection manager for the data source. Example 6-9 defines a connectionManager element in the server.xml file to define the connection pool properties for a data source.

Example 6-9 Defines a connectionManager element

```
<dataSource id="db2" jndiName="jdbc/db2" connectionSharing="MatchCurrentState"
isolationLevel="TRANSACTION_READ_COMMITTED" statementCacheSize="20">
  <connectionManager maxPoolSize="20" minPoolSize="5" connectionTimeout="10s"
agedTimeout="30m"/>
  <jdbcDriver libraryRef="DB2JCC4Lib"/>
  <properties.db2.jcc databaseName="SAMPLEDB" serverName="localhost"
portNumber="50000" currentLockTimeout="30s" />
</dataSource>
```

Note: The server uses default values for any connection management settings that are not defined on the connection manager element. If a connection manager is not defined at all for a data source, the server uses default values for all of the settings.

Runtime updates can be made to the data source connection pooling configuration without restarting the Liberty server. Table 6-2 shows the available attributes and how the configuration update is applied.

Table 6-2 Runtime attributes and how configuration update is applied

Attribute name	How the configuration update is applied
agedTimeout	The update is effective immediately.
connectionTimeout	The update is effective immediately.
maxIdleTime	The update is effective immediately.
maxNumberOfMCsAllowableInThread	The update is effective immediately.
maxPoolSize	The update is effective immediately.
minPoolSize	The update is effective immediately.
numConnectionsPerThreadLocal	The update is effective immediately.
reapTime	The update is effective immediately.
purgePolicy	The update is effective immediately.
numConnectionsPerThreadLocal	The update is effective immediately.

6.5 Accessing MongoDB databases

Liberty provides configuration support for MongoDB. MongoDB is a scalable, high-performance, open source NoSQL database. For access to a MongoDB instance, Liberty applications use the MongoDB Java driver that you configure for the server.

To enable an application to use MongoDB, configure a shared library for the MongoDB Java driver and a library reference to the shared library in the `server.xml` file of Liberty. An application can access MongoDB directly from the application or through the `mongodb-2.0` feature and MongoDB instance configurations in the `server.xml` file.

6.5.1 Configuring Liberty to access MongoDB APIs directly

To configure Liberty to access MongoDB APIs directly, use the following steps:

1. Install the MongoDB Java driver in a location that your application and the Liberty run time can access.

JAR files: The JAR files that are used for accessing your database are not provided as part of the Liberty run time. This example uses a Mongo database, which can be downloaded from the following website:

<http://docs.mongodb.org/ecosystem/drivers/java>

Library placement: For easier Liberty server packaging, it is recommended to place driver JAR files in the Liberty shared resources folder (WLP_HOME\usr\shared\resources). This folder is referred to by default by the server variable `${shared.resource.dir}`. Create the Mongo subdirectory in that folder and place the JAR files in that directory.

2. Configure a shared library for the MongoDB driver (.jar in the server.xml) as shown in Example 6-10.

Example 6-10 Shared Library Reference for MongoDB driver

```
<library id="MongoLib">
  <file name="${shared.resource.dir}/mongo/mongo-java-driver-2.13.1.jar" />
</library>
```

3. Configure the library reference for the shared library in an application element in the server.xml file as described in Example 6-11.

Example 6-11 Adding shared library reference on application definition

```
<application name="mongodemo" location="MongoDemo.war">
  <classloader commonLibraryRef="MongoLib" />
</application>
```

The application can now access the MongoDB APIs directly.

To learn more about configuring a Liberty server to access MongoDB APIs directly, see the following IBM Knowledge Center website:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/twlp_mongodb_create.html?cp=SSAW57_8.5.5%2F3-3-11-0-3-2-20-0-1

6.5.2 Configuring Liberty to access MongoDB using runtime injection engine

To use the runtime injection engine, you must have completed the configuration to access MongoDB APIs directly, and then complete the following steps:

1. Add the `mongodb-2.0` feature in `server.xml` file. Enable the `jndi-1.0` feature if you want to use JNDI to look up resources. The JNDI feature is not required if you use only resource injection. Example 6-12 on page 125 shows enabling both features.

Note: You must use the MongoDB Version 2.13.1 Java driver. At the writing of this book, this is the only compatible version.

Example 6-12 Enabling mongodb-2.0 and jndi-1.0 feature in server.xml

```
<featureManager>
  <feature>mongodb-2.0</feature>
  <feature>jndi-1.0</feature>
</featureManager>
```

2. Configure a mongoDB element that has a reference to the shared library (previously created in Example 6-10 on page 124), as shown in Example 6-13.

Example 6-13 Setting a mongoDB element in server.xml

```
<mongo id="mongo" libraryRef="MongoLib" />
```

3. Continue configuring that same mongoDB element as described in Example 6-14.

Example 6-14 Configuring a mongoDB element in server.xml

```
<mongoDB jndiName="mongo/testdb" mongoRef="mongo" databaseName="db-test" />
```

Configuring a JNDI name enables an application or the Liberty run time to find the MongoDB instance. The configuration enables both JNDI lookup and resource injection to MongoDB on the Liberty.

An example of server.xml using the features enabled in the previous steps is provided in Example 6-15.

Example 6-15 server.xml with mongoDB configured

```
<server>
  <featureManager>
    <feature>mongodb-2.0</feature>
    <feature>jndi-1.0</feature>
    <feature>servlet-3.0</feature>
  </featureManager>

  <library id="MongoLib">
    <file name="\${shared.resource.dir}/mongo/mongo-java-driver-2.13.1.jar" />
  </library>

  <application name="mongodemo" location="MongoDemo.war">
    <classloader commonLibraryRef="MongoLib" />
  </application>

  <mongo id="mongo" libraryRef="MongoLib" />
  <mongoDB jndiName="mongo/testdb" mongoRef="mongo" databaseName="db-test" />

</server>
```

To learn more about configuring Liberty to access MongoDB using the runtime injection engine, see the following IBM Knowledge Center website:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/twlp_mongodb_create.html?cp=SSAW57_8.5.5%2F3-3-11-0-3-2-20-0-1

6.5.3 Connecting to a distributed set of MongoDB instances

Accessing the data stored in a distributed set of MongoDB instances is almost the same procedure as connecting to a single MongoDB instance. Instead of configuring a single MongoDB server, you can pass a collection of host names and ports that are either MongoDB replica set members or shared mongoDB servers.

If the host:port combinations are replica set members, the client finds all members and uses the master by default. If the combinations are shared mongoDB servers, the client sends all requests to the closest member with the lowest ping time. If the closest member is down, the client automatically fails over to the next server.

The full configuration of Example 6-15 on page 125 with a distributed set of mongoDB instances running in the same machine listening on ports 9991, 9992, and 9993 is provided in Example 6-16.

Example 6-16 Configuration with a distributed set of mongoDB instances

```
<server>
  <featureManager>
    <feature>mongodb-2.0</feature>
    <feature>servlet-3.0</feature>
  </featureManager>

  <library id="MongoLib">
    <file name="\${shared.resource.dir}/mongo/mongo-java-driver-2.13.1.jar" />
  </library>

  <application name="mongodemo" location="MongoDemo.war">
    <classloader commonLibraryRef="MongoLib" />
  </application>

  <mongo id="mongo1" libraryRef="MongoLib"
hostNames="localhost,localhost,localhost" ports="9991,9992,9993"/>

  <mongoDB jndiName="mongo/testdb" mongoRef="mongo" databaseName="db-test" />

</server>
```

To learn more about connecting to a distributed set of MongoDB, see the following IBM Knowledge Center website:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/twlp_mongodb_create.html?cp=SSAW57_8.5.5%2F3-3-11-0-3-2-20-0-1

6.5.4 Configuring secure container-managed MongoDB connections

To use container-managed security, the Mongo configuration element must specify a user and a password. Only one user is allowed for each Mongo configuration. All MongoDB instances use the specified user and password. For example, all MongoDB instances that reference mongo1, in the following example, use mUserName for user and 123 for password. See Example 6-17 on page 127.

Example 6-17 Specifying user name and password

```
<mongo id="mongo1" libraryRef="MongoLib" user="mUserName" password="123"/>
<mongoDB jndiName="mongo/testdb" mongoRef="mongo1" databaseName="db-test-1"/>
<mongoDB jndiName="mongo/testdb2" mongoRef="mongo1" databaseName="db-test-2"/>
```

To learn more about the secure MongoDB connections, see the following IBM Knowledge Center website:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/twlp_mongodb_create.html?cp=SSAW57_8.5.5%2F3-3-11-0-3-2-20-0-1

6.6 Data access with CouchDB

Data access applications that use CouchDB can run on Liberty. For access to a CouchDB instance, applications use the ektor Java API and a connection instance that is configured for the NoSQL database.

Example 6-18 shows how to define ektor dependency.

Example 6-18 Defining ektor dependency in server.xml

```
<dependency>
  <groupId>org.ektorp</groupId>
  <artifactId>org.ektorp</artifactId>
  <version>1.4.1</version>
</dependency>
```

To enable an application to use CouchDB, you must configure a shared library for the CouchDB Java driver and a library reference to the shared library in the `server.xml` file of Liberty. An application can access CouchDB either directly from the application, or through the `couchdb-1.0` feature. Applications can access CouchDB instance configurations in the `server.xml` file.

JAR files: The JAR files that are used for accessing your database are not provided as part of the CouchDB run time. This example uses a CouchDB database, which can be downloaded from the following website:

<http://couchdb.apache.org>

Library placement: For easier Liberty server packaging, it is recommended to place driver JAR files in the Liberty shared resources folder (`WLP_HOME\usr\shared\resources`). This folder is referred to by default by the server variable `${shared.resource.dir}`. Create the `couch` subdirectory in that folder and place the JAR files in that directory.

To enable an application to use CouchDB, use the following steps:

1. Configure a shared library. Example 6-19 shows how to configure a shared library for the ektor driver files in the `server.xml` file of the Liberty server.

Example 6-19 Configuring shared library for ektor driver files

```
<library id="couchLibrary">
<fileset dir="${shared.resource.dir}/couch" includes="*.jar"/>
</library>
```

2. Add a CouchDB configuration that has a reference to the shared library just created in Example 6-19 on page 127. The next example, Example 6-20, shows how to add CouchDB data source.

Example 6-20 Configure CouchDB Datasource feature in server.xml

```
<couchdb id="couchdb" jndiName="couchdb/connector"
libraryRef="couchLibrary" url="http://localhost:5984"
username="admin" password="password"/>
```

3. Enable the couchdb-1.0 feature to the server.xml file. See Example 6-21.

Example 6-21 Enabling CouchDB feature in server.xml

```
<featureManager>
  <feature>couchdb-1.0</feature>
  <feature>jndi-1.0</feature>
</featureManager>
```

4. Enable direct access to CouchDB from the application. Configure a library reference for the shared library in an application element in the server.xml file. See Example 6-22.

Example 6-22 Defining class in an application

```
<application ...>
  <classloader commonLibraryRef="couchLibrary"/>
</application>
```

To learn more about configuring a Liberty profile to access CouchDB, see the following IBM Knowledge Center website:

https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_couchdb_create.html

6.7 Logging data source activity

To enable JDBC tracing for Liberty, additional logging for activity on a data source, database and driver-specific custom trace setting needs to be enabled. If your JDBC driver does not provide its own custom tracing or logging facilities, or the facilities that it provides are minimal, you can use supplemental JDBC tracing from the application server. For more information about Enabling JDBC Tracing, see the related topic in the IBM Knowledge Center at the following website:

https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_dep_jdbc_trace.html

6.8 Using the timed operations feature to monitor database operations

When enabled, the timed operations feature generates a logged warning when JDBC calls are operating more slowly or quickly than expected.

Periodically, the timed operation feature creates a report in the application server log detailing which operations took the longest to execute. If you run the server dump command, the timed operation feature generates a report containing the information about all the operations it has tracked. You can use the information listed in these reports to decide if anything is running slower or faster than you expect or is acceptable.

To enable timed operations, add the `timedOperations-1.0` feature to the `server.xml` file, as shown in Example 6-23.

Example 6-23 Enabling timedOperations in server.xml

```
<featureManager>
  <feature>jdbc-4.1</feature>
  <feature>timedOperations-1.0</feature>
</featureManager>
```

The timed operations report contains the 10 longest JDBC timed operations. The frequency and enablement of this report is configurable in the `server.xml` file, with a default of once per day (24 hours). It is possible to disable the generation of the report to the logs, or change the frequency of the report (for example, to once every 12 hours). To do so, use the `timedOperation` element inside the `server.xml`, as shown in Example 6-24.

Example 6-24 Disabling the generation of the report and changing the frequency of the report

```
<featureManager>
  <feature>jdbc-4.1</feature>
  <feature>timedOperations-1.0</feature>
</featureManager>

<timedOperation enableReport="false" reportFrequency="12"/>
```

The configuration in Example 6-25 specifies that the report is generated each hour with the first 200 queries. Notice that the entry, `enableReport`, is omitted because the default value is true. The `reportFrequency` entry (optional) is using the time indicator 1h.

Example 6-25 Generating reports each hour with the first 200 queries

```
<featureManager>
  <feature>jdbc-4.1</feature>
  <feature>timedOperations-1.0</feature>
</featureManager>

<timedOperation reportFrequency="1h" maxNumberTimedOperations="200" />
```

Note: The value of `maxNumberTimedOperations` is an integer and the default value is 10000. The setting for `enableReport` is boolean and the default value is true. The `reportFrequency` is a string and its unit of time can be set.

To get detailed information about the timed operations that have an abnormal behavior, change the `traceSpecification` attribute to include the trace string `com.ibm.ws.timedoperations.*=FINE` in `server.xml`.

Example 6-26 shows a trace example in `server.xml`.

Example 6-26 Trace example in `server.xml`

```
<logging traceSpecification="com.ibm.ws.timedoperations.*=FINEST"/>
```

Trace can be included in the `bootstrap.properties` file, as shown in Example 6-27.

Example 6-27 Trace example in `bootstrap.properties`

```
"com.ibm.ws.timedoperations.*=FINEST"
```

Note: The values of trace level must be (from higher-detail level to lower-detail level) one of these types: FINEST, FINER, FINE, DETAIL, CONFIG, INFO, AUDIT, WARNING, SEVERE, or FATAL.

Example 6-28 shows a sample of an automatically generated report in the log.

Example 6-28 Timed operations report

```
[4/9/13 7:49:13:590 PDT] 00000018 id=
com.ibm.wsspi.timedoperations.TimedOperationService          I TRAS0092I: The
following operations took the longest time to run since the last report has been
generated:
Operation websphere.datasource.execute:jdbc/DataSource:select count(order) as
"ordert" from orderb o where o.account_accountid in (select accountid from
accountb a where a.profile_userid like 'uid:%') took 280ms to complete
```

For more information about the Timed Operations feature on Liberty, see the following IBM Knowledge Center website:

https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_timeop.html?lang=en



Messaging applications

Liberty supports asynchronous messaging as a method of communication through the Java Message Service (JMS) programming interface. There are three JMS messaging providers that are supported in Liberty:

- ▶ The Liberty embedded messaging engine
- ▶ The service integration bus, which is the default messaging provider of WAS Classic
- ▶ WebSphere MQ messaging provider, which uses the WebSphere MQ system as the provider

For details about elements, their attributes, and properties used in this chapter to configure messaging, refer to the following website:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/autodita/rwlp_metatype_4ic.html

This chapter includes the following topics:

- ▶ Liberty embedded JMS messaging provider
- ▶ Interoperating with the service integration bus messaging provider
- ▶ WebSphere MQ messaging provider
- ▶ Liberty application client container

7.1 Liberty messaging server configuration features

Liberty servers that host messaging applications require the appropriate features to be enabled. Those features and what they configure are noted in the following list:

- ▶ The `wasJmsServer-1.0` feature configures the Liberty server to support the JMS server run time. This feature provides the capabilities for connections, transactions, persistence, security, and so on.
- ▶ The `wasJmsClient-1.1` or `wasJmsClient-2.0` configures the Liberty server to support JMS client connectivity. This feature provides the resource adapter support that allows JMS clients to perform synchronous and asynchronous messaging activities.

Note: The `wasJmsClient-2.0` feature supersedes the `wasJmsClient-1.1` feature. The `wasJmsClient-2.0` feature is compliant with JMS 2.0 specifications and is supported only in JDK 7 or later. If you use JDK 6, you must use the `wasJmsClient-1.1` feature and the JMS 1.1 specification.

To enable JMS in a Liberty server, add the `wasJmsServer-1.0` or `wasJmsClient-x.x` features in the `server.xml` file, as shown in Example 7-1.

Example 7-1 Enabling JMS

```
<featureManager>
  <feature>wasJmsServer-1.0</feature>
  <feature>wasJmsClient-2.0</feature>
  <feature>jndi-1.0</feature>
</featureManager>
```

The Liberty server also supports the use of message-driven beans (MDBs). The `jmsMdb-3.1` or `jmsMdb-3.2` feature provides support for deploying and configuring the JMS resources that are required for the MDB to run within Liberty. This feature enables MDB to interact with either the embedded Liberty messaging or WebSphere MQ.

If you want to perform a Java Naming and Directory Interface (JNDI) lookup for JMS resources, you must also add the `jndi-1.0` feature.

7.2 Liberty embedded JMS messaging provider

For an application to send or receive messages from a destination hosted on a Liberty embedded messaging provider, the application must be connected with the messaging engine that keeps the queues or topics.

Liberty messaging allows you to use three types of JMS application connectivity:

- ▶ The JMS application is running on the same Liberty server as the Liberty messaging engine. See section 7.2.1, “Enabling JMS messaging for a single Liberty server” on page 133.
- ▶ The JMS application is running on a Liberty server and connects over TCP/IP to a Liberty messaging engine on a different server. See section 7.2.2, “Enabling JMS messaging between two Liberty servers” on page 136.
- ▶ The JMS application is running on a WAS Classic server and connects over TCP/IP to a Liberty messaging engine. See section 7.3.1, “Enabling service integration bus to connect to Liberty messaging” on page 138.

Figure 7-1 illustrates Liberty messaging connectivity.

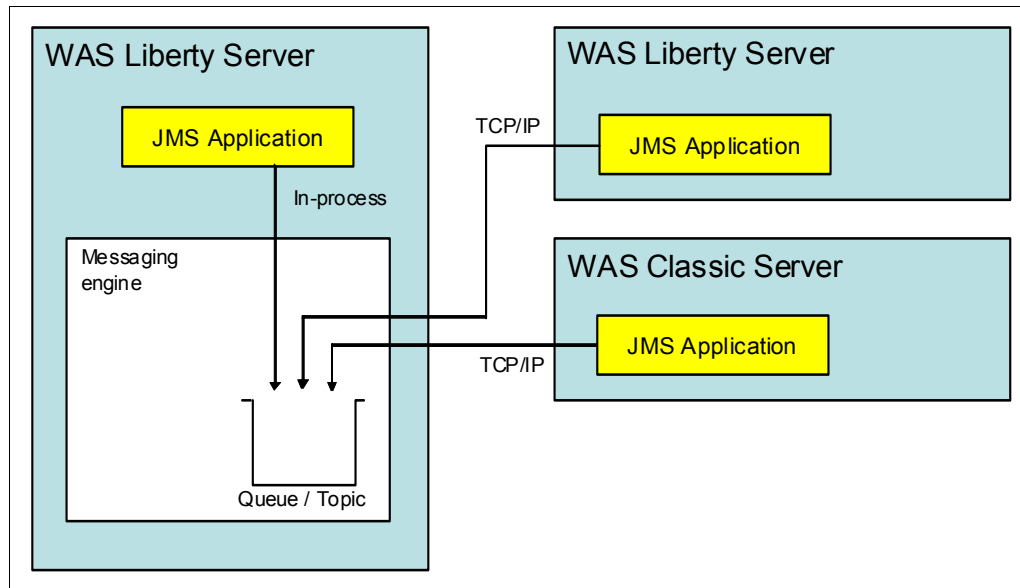


Figure 7-1 Liberty embedded messaging connectivity

7.2.1 Enabling JMS messaging for a single Liberty server

The steps in this section are applicable to a JMS application being deployed in the same Liberty server as is hosting the messaging engine.

Tip: If you have trouble configuring the server for JMS messaging, including using the proper elements, attributes, syntax, or other issues, a simple way to get started is to use the WebSphere developer tools. The tools provide an editor for `server.xml` that allows you to choose the elements and configure them.

To enable JMS messaging for a single Liberty server, use the following sections:

- ▶ Add the features
- ▶ Configuring point-to-point messaging (connection to a queue)
- ▶ Configuring publish and subscribe messaging (connection to a topic)

Add the features

To enable JMS messaging for this application, add the appropriate feature to enable the JMS client or server function.

In this example, the Liberty server hosts a JMS client application and receives requests from external JMS applications. Both the `wasJmsServer-1.0` and `wasJmsClient-x.x` features are added to the server configuration for this scenario. See Example 7-2.

Example 7-2 JMS client and server features in server.xml

```
<featureManager>
  <feature>wasJmsServer-1.0</feature>
  <feature>wasJmsClient-2.0</feature>
  <feature>jndi-1.0</feature>
</featureManager>
```

Configuring point-to-point messaging (connection to a queue)

To configure the messaging engine and the elements that the application uses, proceed with the following steps:

1. Use the `<messagingEngine>` element to create a messaging engine and queue (queue1). See Example 7-3.

Example 7-3 Messaging engine definition

```
<messagingEngine>
  <queue id="queue1" />
</messagingEngine>
```

2. (Optional) Enable the messaging engine to accept remote incoming messaging connections from TCP/IP. To do so, use the `<wasJmsEndpoint>` element. Use the following attributes to specify the host interfaces and ports to use for the endpoint. See Example 7-4:

- The default for the host is `localhost`. You can use `"*"` for all available network interfaces.
- The default for the JMS port is `7276`. Use the `wasJmsPort` attribute to specify a different port.
- The default for the JMS secure port is `7286`. Use the `wasJmsSSLPort` attribute to specify a different port.

Example 7-4 JMS endpoint definition

```
<wasJmsEndpoint
  host="*"
  wasJmsPort="7276" wasJmsSSLPort="7286" >
</wasJmsEndpoint>
```

3. Define a JMS queue connection factory for the Liberty messaging engine. See Example 7-5.

The connection factory is created with the `<jmsQueueConnectionFactory>` element and the following attributes:

- Use the `jndiName` attribute to provide the JNDI name for the application to use to identify the queue connection factory.
- (Optional) Specify the connection manager with the `<connectionManager>` element. Attributes for this element can be used to manage the connections, for example, connection pool sizes, timeout values, and purge policy.

Best practice: Nest the `connectionManager` (without any `id` attribute) under the connection factory, where it is guaranteed to pool connections only for that one connection factory. It is not advised to configure `connectionManager` as top level with an `id`, because this allows multiple connection factories and data sources to attempt to use the same pool, which is not supported.

- Specify the `properties.wasJms` attribute to indicate that any properties defined within the tag are interpreted as WebSphere messaging (Liberty or WAS Classic messaging).

Example 7-5 Queue connection factory definition

```
<jmsQueueConnectionFactory jndiName="jms/QueueCF">
  <connectionManager maxPoolSize="10" />
  <properties.wasJms />
</jmsQueueConnectionFactory>
```

```
</jmsQueueConnectionFactory>
```

4. Define a JMS queue with the `<jmsQueue>` element. See Example 7-6:
 - Specify the ID attribute to provide a way to reference the element.
 - Use the `jndiName` attribute to define the JNDI name that the application uses to look up the queue.
 - Specify the `properties.wasJms` attribute and the `queueName` tag to specify the queue to which this JMS definition refers. The queue named in the `queueName` tag refers to the queue defined in the messaging engine.

Example 7-6 JMS queue definition

```
<jmsQueue id="queue1" jndiName="jms/Queue">  
  <properties.wasJms queueName="queue1" />  
</jmsQueue>
```

Example 7-7 shows the completed definition of point-to-point messaging.

Example 7-7 Point-to-point messaging

```
<featureManager>  
  <feature>wasJmsServer-1.0</feature>  
  <feature>wasJmsClient-2.0</feature>  
  <feature>jndi-1.0</feature>  
</featureManager>  
  
<messagingEngine>  
  <queue id="queue1" />  
</messagingEngine>  
  
<wasJmsEndpoint host="*" wasJmsPort="7276">  
</wasJmsEndpoint>  
  
<jmsQueueConnectionFactory jndiName="jms/QueueCF">  
  <connectionManager maxPoolSize="10" />  
  <properties.wasJms />  
</jmsQueueConnectionFactory>  
  
<jmsQueue id="queue1" jndiName="jms/Queue">  
  <properties.wasJms queueName="queue1" />  
</jmsQueue>
```

Configuring publish and subscribe messaging (connection to a topic)

Setting up a topic space for publish and subscribe messaging is similar to the process detailed in “Configuring point-to-point messaging (connection to a queue)” on page 134. In this example, define the following options:

- ▶ Define a messaging engine and the topic space.
- ▶ (Optional) To enable the messaging engine to accept the remote incoming messaging connections from TCP/IP, define a JMS endpoint element with the host and port number.
- ▶ Define a JMS topic connection factory to connect to the messaging engine that has the topic definition.
 - (Optional) Define a connection manager.
- ▶ Define a JMS topic.

Example 7-8 shows a server configured for publish and subscribe messaging.

Example 7-8 Publish and subscribe messaging definition

```
<featureManager>
  <feature>wasJmsServer-1.0</feature>
  <feature>wasJmsClient-2.0</feature>
  <feature>jndi-1.0</feature>
</featureManager>

<messagingEngine>
  <topicSpace id="topicspace1" />
</messagingEngine>

<jmsTopicConnectionFactory jndiName="jms/TopicCF">
  <connectionManager maxPoolSize="10" />
  <properties.wasJms />
</jmsTopicConnectionFactory>

<jmsTopic jndiName="jms/Topic" id="topic1">
  <properties.wasJms topicName="anyTopic" topicSpace="topicspace1" />
</jmsTopic>
```

7.2.2 Enabling JMS messaging between two Liberty servers

This scenario describes a situation where the JMS client application is running on one Liberty server but using a messaging engine that is hosted in another Liberty server (the server).

To configure the Liberty server that is hosting the messaging engine, take the following actions:

- ▶ Enable wasJmsServer-1.0 feature and (optionally) the jndi-1.0 feature.
- ▶ Define a messaging engine and queue.
- ▶ By default, the messaging engine listens on port 7276 (unsecured) and 7286 (secured). To bind the messaging engine to different ports, specify the ports on the wasJmsEndpoint element.

Example 7-9 shows a Liberty server configuration for the JMS server that is hosting the messaging engine.

Example 7-9 Server hosting the messaging engine

```
<featureManager>
  <feature>wasJmsServer-1.0</feature>
  <feature>jndi-1.0</feature>
</featureManager>

<wasJmsEndpoint host="*"
  wasJmsPort="9011" wasJmsSSLPort="9100" />

<messagingEngine>
  <queue id="queue1" />
</messagingEngine>
```

To configure the Liberty server that is hosting the JMS client application, take the following actions:

- ▶ Enable the `wasJmsClient-x.x` feature and (optionally) the `jndi-1.0` feature.
- ▶ Configure a JMS queue connection factory for the Liberty messaging engine.
 - Specify `properties.wasJms` attribute to indicate that any properties defined within the tag are interpreted as WebSphere messaging.

Add the `remoteServerAddress` property to define the TCP/IP connection to the server with the messaging engine. The following format is shown in Example 7-10.

Example 7-10 Remote server address definition

```
remoteServerAddress="JMS_server_host:wasJmsPort:bootstrap_transport_chain"
```

The terms used in Example 7-10 are defined in the following list:

- `JMS_server_host` is the host name of the messaging engine server.
- `wasJmsPort` is one of the following addresses of the messaging engine hosting the remote end of the link:

If security is not enabled, use the value for `wasJmsPort` specified on the `wasJmsEndpoint` element on the server side.

If security is enabled, use the value for `wasJmsSSLPort` specified on the `wasJmsEndpoint` element on the server side.

- When connecting to another Liberty server, use the value `BootstrapBasicMessaging`, or for an SSL connection, use `BootstrapSecureMessaging` for the bootstrap transport chain value.

- (Optional) Define a connection manager.

- ▶ Define a JMS queue.

Example 7-11 shows a definition of Liberty server that is hosting the client application (but no messaging engine).

Example 7-11 Server not hosting any messaging engine

```
<featureManager>
  <feature>wasJmsClient-2.0</feature>
  <feature>jndi-1.0</feature>
</featureManager>

<jmsQueueConnectionFactory jndiName="jms/QueueCF">
  <connectionManager maxPoolSize="10" />
  <properties.wasJms
    remoteServerAddress="localhost:9011:BootstrapBasicMessaging" />
</jmsQueueConnectionFactory>

<jmsQueue id="queue1" jndiName="jms/Queue">
  <properties.wasJms queueName="queue1" />
</jmsQueue>
```

7.3 Interoperating with the service integration bus messaging provider

A *service integration bus* is a group of one or more WAS Classic servers that, as bus members, cooperate to provide asynchronous messaging services. The Liberty embedded messaging can interoperate with the service integration bus. This interoperation means that JMS clients on a Liberty server can send and receive messages to a destination on the WAS Classic service integration bus. Also, JMS clients on WAS Classic can send messages to the Liberty server.

For more information about the service integration bus, see the WebSphere Application Server V8.5.5 IBM Knowledge Center at this website:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/a/cjj0000_.html

7.3.1 Enabling service integration bus to connect to Liberty messaging

JMS applications that are deployed in WAS Classic server can be enabled to connect to Liberty messaging. Enabling requires you to configure the bus name and the ProviderEndPoint in WAS Classic server to specify the host and port where the Liberty messaging engine is running.

To start this process, in the WAS Classic administrative console, create a connection factory that defines the endpoint for the Liberty messaging engine. Use the following steps:

1. From the left menu, click **Resources** → **JMS** → **Connection factories**.
2. In the Connection factories window, click **New**.
3. Select the **Default messaging provider** option and then click **OK**.

Note: The JMS resources that are pointing to the Liberty messaging engine must always specify the bus name as `defaultBus` (case sensitive).

Figure 7-2 on page 139 shows how to create a connection factory by using the administrative console.

Administration

Scope

Provider

* Name

* JNDI name

Description

Category

Connection

* Bus name

Target

Target type

Target significance

Target inbound transport chain

Provider endpoints

Connection proximity

Figure 7-2 Creating Connection Factory

7.3.2 Enabling Liberty server to connect to a bus for point-to-point messaging

To set up the Liberty server to connect to a service integration bus for point-to-point messaging, take the following actions:

- ▶ Enable `wasJmsClient-x.x` feature and (optionally) the `jndi-1.0` feature.
- ▶ Configure a JMS queue connection factory for the Liberty messaging engine.
 - Specify the `properties.wasJms` attribute to indicate that any properties defined within the tag are interpreted as WebSphere messaging.

- Add the `remoteServerAddress` property to define the TCP/IP connection to the server with the messaging engine. The following format is shown in Example 7-12.

Example 7-12 Remote server address definition

```
remoteServerAddress="JMS_server_host:wasJmsPort:bootstrap_transport_chain"
```

The terms used in Example 7-12 are defined in the following list:

- `JMS_server_host` is the host name of the messaging engine server.
- `wasJmsPort` is one of the following addresses of the server hosting the messaging engine at the remote end of the link:

If security is not enabled, use the value for `SIB_ENDPOINT_ADDRESS`, which is 7276 by default.

For secure connections, use the value for `SIB_ENDPOINT_SECURE_ADDRESS`, which is 7286 by default.

- When connecting to another Liberty server, use the value `BootstrapBasicMessaging`, or for an SSL connection, use `BootstrapSecureMessaging` for the bootstrap transport chain value.

For details about these settings for the service integration bus, see the discussion on provider endpoints at the following website:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/SIBJMSConnectionFactory_DetailForm.html

- ▶ Define a JMS queue that names the queue on the service integration bus.

Example 7-13 shows a definition of the Liberty server for connection.

Example 7-13 Enabling Liberty to connect to a bus for point-to-point

```
<featureManager>
  <feature>wasJmsClient-2.0</feature>
  <feature>jndi-1.0</feature>
</featureManager>

<jmsQueueConnectionFactory jndiName="jms/QueueCF">
  <properties.wasJms busName="SampleBus"
    remoteServerAddress="localhost:7276:BootstrapBasicMessaging"
    targetTransportChain="InboundBasicMessaging" />
</jmsQueueConnectionFactory>

<jmsQueue jndiName="jms/Queue">
  <properties.wasJms queueName="queue1" />
</jmsQueue>
```

7.3.3 Enabling Liberty server to connect to a bus for publish and subscribe

To set up the Liberty server to connect to a bus in WAS Classic for publish and subscribe domain, take the following actions:

- ▶ Enable the `wasJmsClient-x.x` feature and optionally, the `jndi-1.0` feature.
- ▶ Define a JMS topic connection factory to define the connection to the messaging engine on the service integration bus.

- ▶ Specify the `properties.wasJms` attribute with the bus name and the remote server address property to define the connection to the messaging engine on the service integration bus.
- ▶ If you need durable topic subscriptions on all connections created by using this connection factory, specify `clientID` attribute in `properties.wasJms`.
- ▶ Define a JMS topic that identifies the topic on the bus.

Example 7-14 shows a definition of the Liberty server for bus connection.

Example 7-14 Enabling Liberty to connect to a bus for publish and subscribe

```
<featureManager>
  <feature>wasJmsClient-2.0</feature>
  <feature>jndi-1.0</feature>
</featureManager>

<jmsTopicConnectionFactory jndiName="jms/TopicCF">
  <properties.wasJms busName="SampleBus"
    remoteServerAddress="localhost:7276:BootStrapBasicMessaging"
    clientID="defaultID" />
</jmsTopicConnectionFactory>

<jmsTopic jndiName="jms/Topic">
  <properties.wasJms topicName="Topic1" />
</jmsTopic>
```

7.3.4 Enabling Liberty server to connect to a bus for message-driven beans

The Liberty server supports the use of message-driven beans (MDBs) as asynchronous message consumers. Incoming messages are passed automatically to the `onMessage()` method of an MDB that is deployed as a listener for the destination. The MDB processes the message. In this example, the MDBs are running on the Liberty server and the queue (where the messages are received) is on the service integration bus.

To configure a Liberty server for MDBs that connect to a service integration bus, take the following actions:

- ▶ Enable `wasJmsClient-1.1` and `jmsMdb-3.1` features, or enable `wasJmsClient-2.0` and `jmsMdb-3.2` features. You can optionally enable the `jndi-1.0` feature.
- ▶ Activation specifications are used to configure inbound message delivery to message-driven beans (MDBs) running in the Liberty server. The specification contains the information needed to receive messages. A JMS activation specification is associated with an MDB during application deployment.

The JMS activation identifies the endpoint on the service integration bus where the messages arrive.

- ▶ Define the JMS queue and specify the queue name.

Example 7-15 shows a definition of the Liberty server ready to connect to a bus for message-driven beans.

Example 7-15 Enabling Liberty to connect to a bus for message-driven beans

```
<featureManager>
  <feature>wasJmsClient-2.0</feature>
  <feature>mdb-3.2</feature>
  <feature>jndi-1.0</feature>
</featureManager>

<jmsActivationSpec id="JMSApp/SampleMDB">
  <properties.wasJms
    destinationRef="JMSQueue"
    remoteServerAddress="localhost:7276:BootStrapBasicMessaging"
    busName="SampleBus" />
</jmsActivationSpec>

<jmsQueue jndiName="jms/TriggerQ" id="JMSQueue">
  <properties.wasJms queueName="Q1" />
</jmsQueue>
```

7.4 WebSphere MQ messaging provider

The WebSphere MQ messaging provider allows the use of WebSphere MQ system as an external provider of JMS messaging resources. WebSphere MQ is both JMS 1.1 and JMS 2.0 compliant.

The WebSphere MQ messaging provider support in Liberty has the following restrictions:

- ▶ The WebSphere MQ classes for Java (often called the *Base Java*) are not included in the WebSphere MQ Liberty messaging feature. (Base Java is included in the resource adapter for other application servers but is not recommended for the Base Java APIs in the Java Enterprise Edition environments).
- ▶ The WebSphere MQ resource adapter has a transport type of BINDINGS_THEN_CLIENT. This transport type is not supported by the WebSphere MQ Liberty messaging feature.
- ▶ The Advanced Messaging Security (AMS) feature is not included in the WebSphere MQ Liberty messaging feature.

7.4.1 Enabling Liberty to connect WebSphere MQ

To set up the Liberty server to connect to a WebSphere MQ messaging provider, take the following actions:

- ▶ Enable the feature `wmqJmsClient-1.1` or `wmqJmsClient-2.0` in the Feature Manager.
- ▶ Define a JMS connection factory to configure the connection to WebSphere MQ. Use `properties.wmqJms` to indicate that the properties are for a WebSphere MQ provider. Configure the host name and the name, port, and channel of the queue manager.
(Optional) Define a connection manager.

- ▶ Define a variable that specifies the location of the WebSphere MQ Resource Adapter (`wmqJmsClient.rar.location`).
- ▶ Define a JMS queue to provide the queue manager and queue name.

Example 7-16 shows a definition of the Liberty server for connection with WebSphere MQ.

Example 7-16 Enabling Liberty to connect to WebSphere MQ

```

<featureManager>
  <feature>wmqJmsClient-2.0</feature>
  <feature>jndi-1.0</feature>
</featureManager>

<jmsConnectionFactory jndiName="jms/wmqCF">
  <connectionManager maxPoolSize="2" />
  <properties.wmqJms
    transportType="CLIENT"
    hostName="WebSphereMQhost"
    port="1414"
    channel="SYSTEM.DEF.SVRCONN"
    queueManager="QM01" />
</jmsConnectionFactory>

<variable name="wmqJmsClient.rar.location"
  value="/path/to/wmq/rar/wmq.jmsra.rar" />

<jmsQueue id="jms/queue1" jndiName="jms/wmqQ1">
  <properties.wmqJms
    baseQueueName="queue1"
    baseQueueManagerName="QM01" />
</jmsQueue>

```

For the JMS applications to connect using either the shared memories or in BINDING mode to WebSphere MQ, both the Liberty server and WebSphere MQ must be deployed on the same server. To allow JMS applications to connect in BINDING mode, use the `<nativeLibraryPath>` element in the `server.xml` file. Use that path to specify the location of the WebSphere MQ native libraries, as shown in Example 7-17.

Example 7-17 Enable JMS application in BINDING mode

```

<wmqJmsClient nativeLibraryPath="/opt/mqm/java/lib64"/>

```

7.4.2 Deploying message-driven beans to connect to WebSphere MQ

You can connect to WebSphere MQ using the message-driven beans (MDB). To do so use the features `jmsMdb-x.x` and `wmqJmsClient-x.x` in the `server.xml` file.

To set up the Liberty server to connect to WebSphere MQ using the MDB, take the following actions:

- ▶ Enable the `wmqJmsClient-1.1` and the `jmsMdb-3.1` features, or enable the `wmqJmsClient-2.0` and the `jmsMdb-3.2` features.
- ▶ Define a variable that specifies the location of the WebSphere MQ Resource Adapter (`wmqJmsClient.rar.location`).

- Define a JMS activation specification. Use the `properties.wmqJms` attributes to configure the transport type, host name, channel, port, and queueManager.

Important: The ID value on the `jmsActivationSpec` element must be in the format of application name/bean name or module name/bean name using the following definitions:

Application name	The name of the application that is deployed (for example, <code>JMSSample</code>). The application name applies only if the bean is packaged within an EAR file. The application defaults to the base name of the EAR file with no file name extension unless specified by the <code>application.xml</code> deployment descriptor.
Module name	The name of the module in which the bean is packaged. In a stand-alone <code>ejb-jar</code> file or WAR file, the <code><module-name></code> defaults to the base name of the module with any file name extension removed. In an EAR file, the <code><module-name></code> defaults to the path name of the module with any file name extension removed, but with any directory names included. The default <code><module-name></code> can be overridden by using the <code>module-name</code> element of <code>ejb-jar.xml</code> (for <code>ejb-jar</code> files) or <code>web.xml</code> (for WAR files).
Bean name	The <code>ejb-name</code> of the enterprise bean. For enterprise beans defined through annotation, the bean name defaults to the unqualified name of the session bean class, unless specified in the contents of the <code>name()</code> attribute of the <code>MessageDriven</code> annotation. For enterprise beans defined through <code>ejb-jar.xml</code> , it is specified in the <code><ejb-name></code> deployment descriptor element.

- Define a JMS queue. Use the `properties.wmqJms` attributes to configure the queue manager and queue name.

Example 7-18 shows a definition of a Liberty server to connect to WebSphere MQ by using the message-driven beans.

Example 7-18 Liberty server to connect to WebSphere MQ by using the MDB

```

<featureManager>
  <feature>mdb-3.2</feature>
  <feature>wmqJmsClient-2.0</feature>
  <feature>jndi-1.0</feature>
</featureManager>

<variable name="wmqJmsClient.rar.location"
  value="/path/to/wmq/rar/wmq.jmsra.rar" />

  <jmsActivationSpec id="JMSSample/JMSSampleMDB">
    <properties.wmqJms destinationRef="jndi/MDBQ"
      transportType="CLIENT"
      queueManager="MQ01"
      hostName="WebSpmerMQhost"
      channel="SYSTEM.DEF.SVRCONN"
      port="1414" />
  </jmsActivationSpec>

  <jmsQueue id="jndi/MDBQ" jndiName="jndi/MDBQ">
    <properties.wmqJms baseQueueName="MQ01" baseQueueManagerName="queue1" />
  </jmsQueue>

```

7.5 Liberty application client container

Liberty now supports running Java applications in the application client container. See 4.4.3, “Application client commands” on page 65. You can configure messaging applications that use the `wasJmsClient-2.0` feature to run on the client container.

7.5.1 Defining the server

To configure the Liberty server that is hosting the messaging engine, define the `wasJmsServer-1.0` feature and a messaging engine with queue and topic space to the `server.xml`. Detailed actions are described in 7.2.2, “Enabling JMS messaging between two Liberty servers” on page 136. Example 7-19 shows an example definition of a Liberty server definition.

Example 7-19 Server hosting the messaging engine

```
<featureManager>
  <feature>wasJmsServer-1.0</feature>
</featureManager>

<messagingEngine>
  <queue id="queue1" />
  <topicSpace id="topicspace1" />
</messagingEngine>

<wasJmsEndpoint host="*" wasJmsPort="17276" wasJmsSSLPort="17286">
</wasJmsEndpoint>
```

7.5.2 Creating and configuring the client container

To configure the Liberty client that is running the JMS client application, complete the following steps:

1. Create a client container by using the `client create` command. See Example 7-20.

Example 7-20 Create a client container

```
Liberty_Home/bin/client create client_name
```

2. Add the `javaeeClient-7.0` feature to the client configuration file.

To enable JMS Client in the client container, add the `javaeeClient-7.0` feature in the `Liberty_Home/usr/clients/client_name/client.xml` file. The `javaeeClient-7.0` feature enables the `wasJmsClient-2.0` feature and so on.

3. Add a connection factory and a queue or a topic definition to the client configuration file.

To connect the queue or the topic in the Liberty messaging engine defined in the server, configure `jmsConnectionFactory` and `jmsQueue` or `jmsTopic` to the `client.xml`.

Example 7-21 on page 146 is an example setting for using point-to-point messaging from the client application.

Example 7-21 Point-to-point messaging client definition

```
<featureManager>
  <feature>javaeeClient-7.0</feature>
</featureManager>

<jmsConnectionFactory jndiName="jms/cf">
  <properties.wasJms
    remoteServerAddress="localhost:17276:BootstrapBasicMessaging" />
</jmsConnectionFactory>

<jmsQueue jndiName="jms/queue1">
  <properties.wasJms queueName="queue1" />
</jmsQueue>
```

Example 7-22 is an example setting for using publish and subscribe messaging from the client application.

Example 7-22 Publish and subscribe messaging client definition

```
<featureManager>
  <feature>javaeeClient-7.0</feature>
</featureManager>

<jmsConnectionFactory jndiName="jms/cf">
  <properties.wasJms
    remoteServerAddress="localhost:17276:BootstrapBasicMessaging" />
</jmsConnectionFactory>

<jmsTopic jndiName="jms/topic1">
  <properties.wasJms topicName="topic1" topicSpace="topicspace1" />
</jmsTopic>
```

7.5.3 Deploying the JMS client application to the client container

You can deploy your client application by using either of the following steps:

- ▶ Place your client application EAR file under the *Liberty_Home/usr/clients/client_name/apps* directory and configure your application to the *client.xml* (as shown in Example 7-23).

Example 7-23 Client application definition

```
<application id="CLIENT_APP" name="CLIENT_APP" type="ear"
  location="ITSOJMS.ear" />
```

- ▶ Put the application under the *Liberty_Home/usr/clients/client_name/dropins* directory.

7.5.4 Starting the server and running the client

After completing the previous sections, the JMS client application is ready to run on the Liberty application client container. You can now start the server and run the client by using the following syntax:

```
Liberty_Home/bin/server start server_name
Liberty_Home/bin/client run client_name
```



Monitoring the Liberty server environment

Monitoring support is provided for Liberty; however, Liberty does not deliver with dedicated tools to monitor run time (such as the Tivoli Performance Viewer that is available in the WAS Classic administrative console). In Liberty, monitoring the user runtime components consists of two steps: Enabling the monitoring feature and using a standard tool for Java runtimes to view the monitored data.

This chapter provides information about monitoring the Liberty application server environment. It includes the following topics:

- ▶ Introduction to performance monitoring
- ▶ Monitoring Liberty using the monitor feature
- ▶ Monitoring Liberty using JConsole
- ▶ Monitoring Liberty by using the IBM Monitoring and Diagnostics Tools for Java - Health Center
- ▶ Monitoring Liberty using other tools
- ▶ Tuning Liberty

8.1 Introduction to performance monitoring

Performance monitoring means different things to different people. For some, it is a fast response time for users. For others, it is the volume of work that can be processed within a time period. For others still, it is how rapidly a system can recover from a failure.

Performance monitoring and tuning is essential in any enterprise, helping to ensure maximum returns for the IT investment. Aside from aiding users in getting the best response time, it also helps to determine the maximum load that the application can safely support. Performance problems in any environment can result in escalated support costs, loss of customer confidence, loss of revenue, and loss of credibility.

It is also important to understand that performance monitoring and tuning is an iterative process. You need to make a small adjustment, then measure the impact, then perform analysis, make another adjustment, and so on. Due to the vast differences in the applications the customers build, there are no global solutions that work well in every environment. Improving performance is a process of learning and testing.

8.2 Monitoring Liberty using the monitor feature

The `monitor-1.0` feature allows you to track information about the Liberty server run time. Adding this feature starts the monitoring functions. The monitoring feature in Liberty is different from the Performance Monitoring Infrastructure (PMI) in WAS Classic. The monitoring feature in Liberty collects performance data at run time, and the data is available as attributes on MXBean Java objects.

Table 8-1 shows the Liberty runtime components that are monitored and their associated MXBean.

Table 8-1 Monitored runtime components and their MXBean

Runtime component	MXBean
JVM	WebSphere:type=JvmStats
Web applications	WebSphere:type=ServletStats,name=*
The thread pool	WebSphere:type=ThreadPoolStats,name=Default Executor
Java API for XML Web Services (JAX-WS) endpoints	org.apache.cxf:type=WebServiceStats,service=*,port=*
Session management	WebSphere:type=SessionStats,name=*
Connection pool	WebSphere:type=ConnectionPool,name=*

See the following IBM Knowledge Center for detailed information about monitoring Liberty using MXBeans:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/twlp_mon.html

8.3 Monitoring Liberty using JConsole

You can use IBM Java™ Monitoring and Management Console (JConsole) to connect to a Java virtual machine (JVM), then look at the performance data that is collected by using each attribute of the MXBean. JConsole is a graphical tool that allows you to monitor and manage the behavior of Java applications. When JConsole connects to a Java application, it reports information about the application. The details include memory usage, the running threads, and the loaded classes. This data allows you to monitor the behavior of your application and the JVM. This information is useful in understanding performance problems, memory usage issues, hangs, and deadlocks.

JConsole is shipped as part of the IBM software development kit (SDK). To run JConsole against the Liberty server, start JConsole from the command line, as shown in the following syntax:

```
<SDK_install>\bin>jconsole.exe
```

When JConsole launches, a welcome window displays. Connect to the Liberty server Java process. To monitor a local process, the JConsole process must run under the same operating system user ID and using the same Java runtime as the server. In this case, you can choose the local server process. For more information about connecting to JConsole, see Chapter 5, “Administering the WebSphere Liberty profile” on page 71.

The JConsole loads all runtime information into the tool, including the characteristic of the Java runtime, heap size, and CPU usage, as illustrated in Figure 8-1.

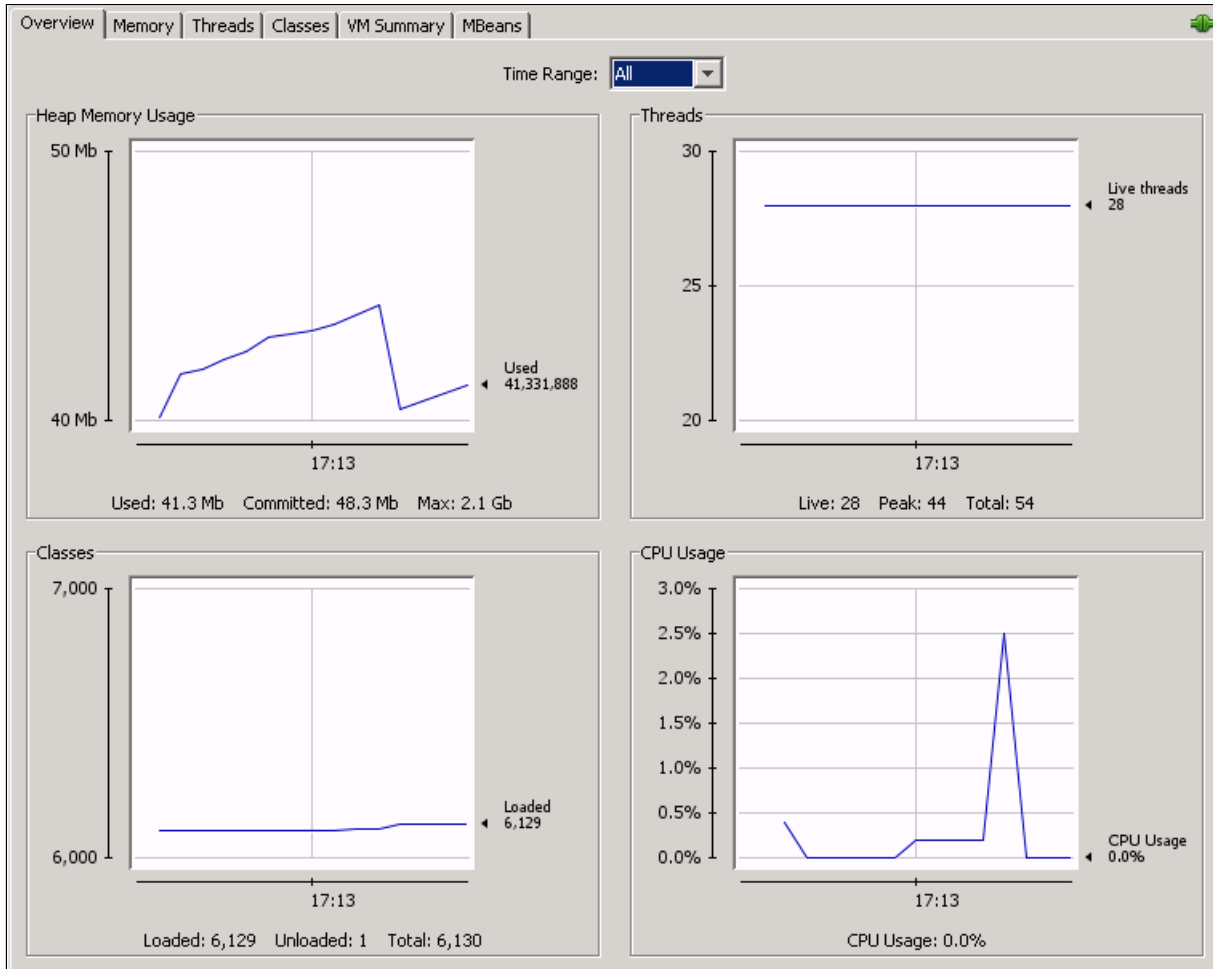


Figure 8-1 Monitoring key runtime characteristics of the Liberty server

Using JConsole, you can trigger MBeans operations that are part of Liberty. MBeans are available on the MBeans tab. Figure 8-2 on page 151 presents an example of issuing a restart operation on the ITSOWebCustomerCredit application.

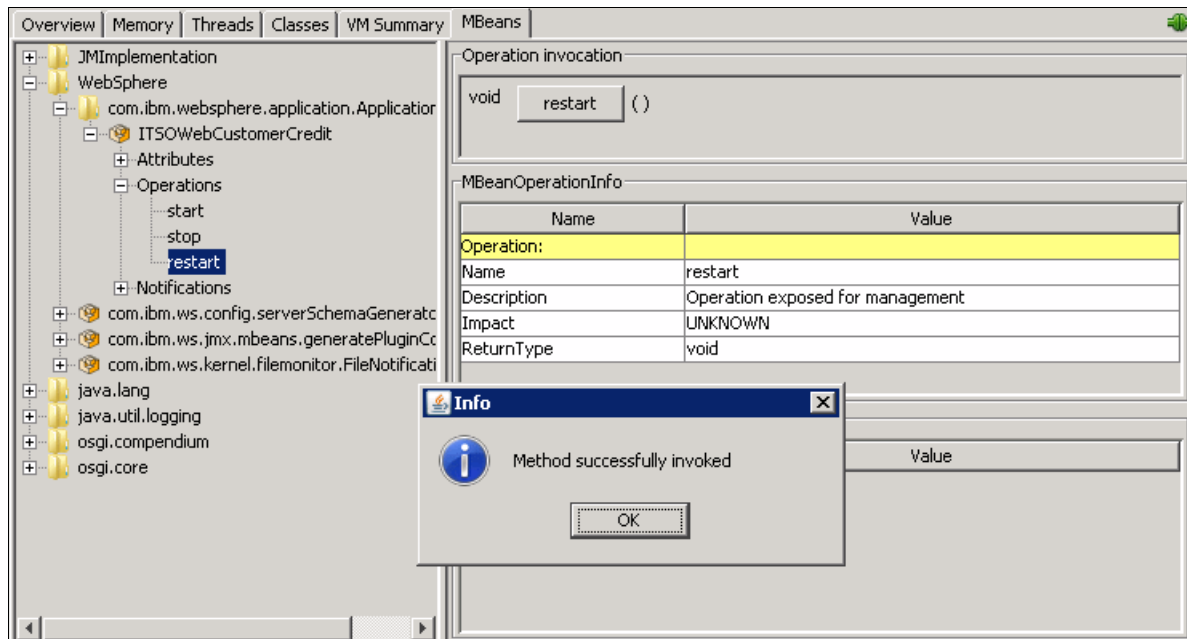


Figure 8-2 Restarting an application deployed on a Liberty server using JConsole

If you enabled the monitoring infrastructure in Liberty, you can also access additional MXBeans, such as JvmStats, as illustrated in Figure 8-3.

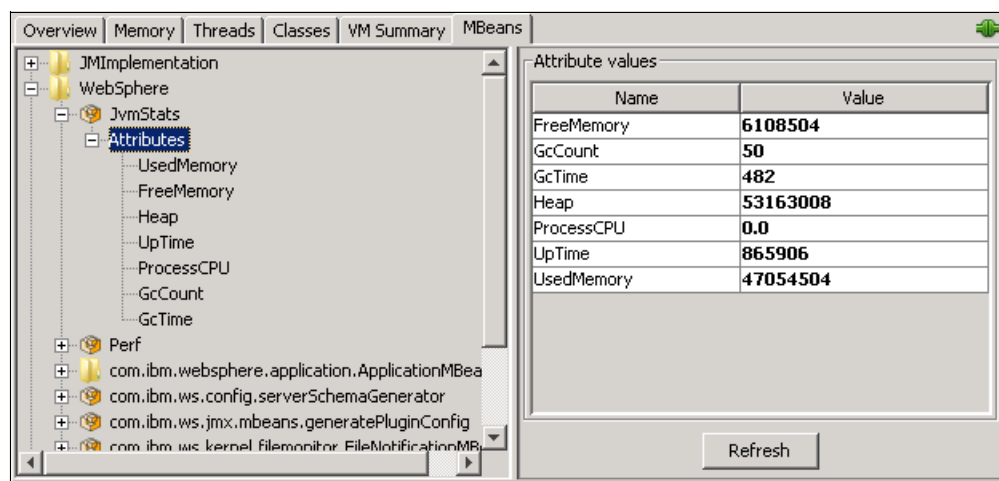


Figure 8-3 Accessing the Liberty monitoring infrastructure JvmStats MBean

8.3.1 Monitoring the Liberty run time remotely using a REST connector

Liberty provides a Representational State Transfer (REST) connector to establish a secured Java Management Extensions (JMX) connection to the Liberty server by using Secure Sockets Layer (SSL). The secured JMX connection is enabled with the feature `restConnector-1.0`.

Note: An application deployed on a Liberty server has unrestricted access to the MBeanServer directory.

Figure 8-4 shows how a remote JConsole can connect to a Liberty server using the restConnector.

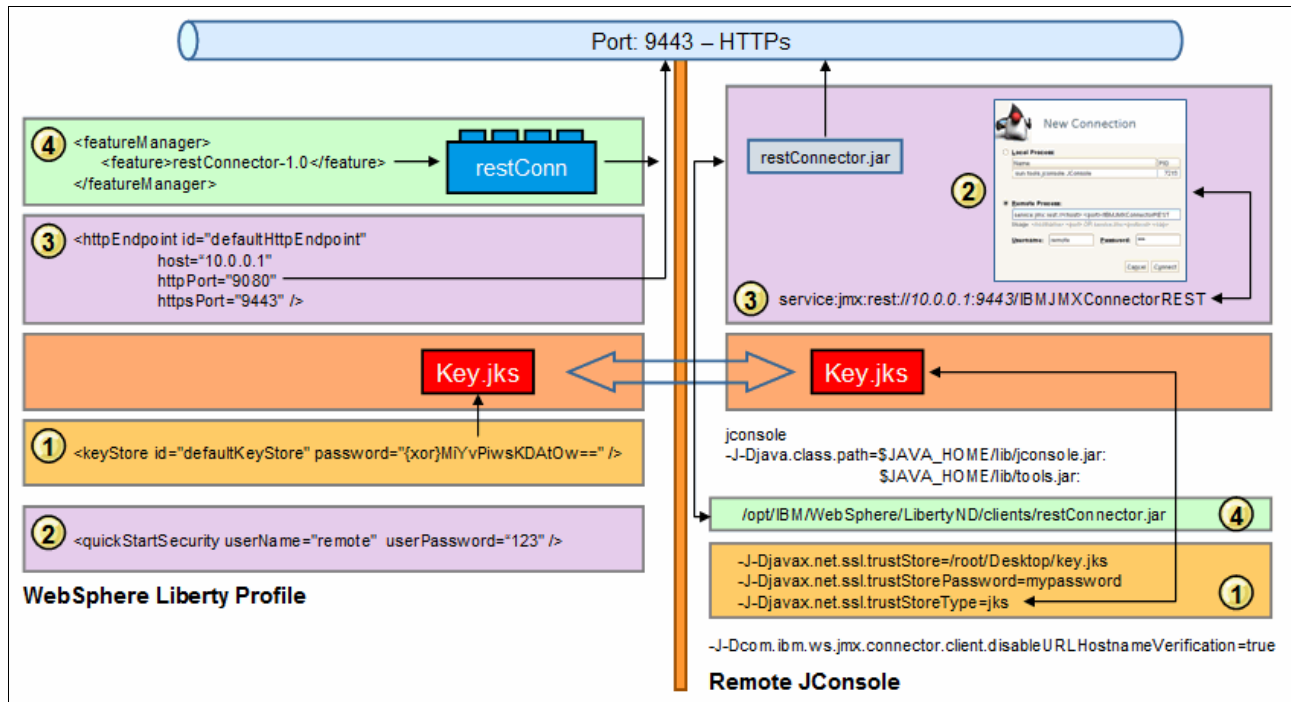


Figure 8-4 Diagram of a remote JConsole connection using restConnector

For more information about a simple method to configure and access the REST connector on a Liberty server, see Chapter 5, “Administering the WebSphere Liberty profile” on page 71.

8.4 Monitoring Liberty by using the IBM Monitoring and Diagnostics Tools for Java - Health Center

You can access the status of a running Java application by using the IBM Monitoring and Diagnostic Tools for Java - Health Center. The Health Center, available at no charge, is a lightweight diagnostic tool and API that monitors active JVMs for Java with minimal performance overhead. The Health Center suggests live tuning recommendations for Garbage Collection, profiles methods that include call stacks, and highlights contended locks.

Health Center provides a wealth of knowledge about server performance, including:

- ▶ Memory usage
- ▶ Garbage collection statistics
- ▶ Method-level profiling
- ▶ Threading
- ▶ Java Class loading
- ▶ Lock contention analysis

Health Center monitors several application areas, using the information to provide recommendations and analysis that help you improve the performance and efficiency of your application. Health Center can save the data that is obtained from monitoring an application and load it again for analysis later.

Health Center is included as a tool in IBM Support Assistant. Ensure that you install IBM Support Assistant on a different machine than the server machine; otherwise, the Health Center uses resources from the server process, and your results might not be accurate. You can also install and use the Health Center within an Eclipse client. The Health Center works only for IBM Java.

The Health Center includes an agent and a client, which you install separately. The Health Center agent collects data from a running application in your environment. The agent uses a small amount of processor time and memory and must be manually installed in an IBM JVM. The Health Center client connects to the agent and interprets the data that is obtained by the agent and provides recommendations to improve the performance of the monitored application.

Figure 8-5 shows the architecture for the Health Center.

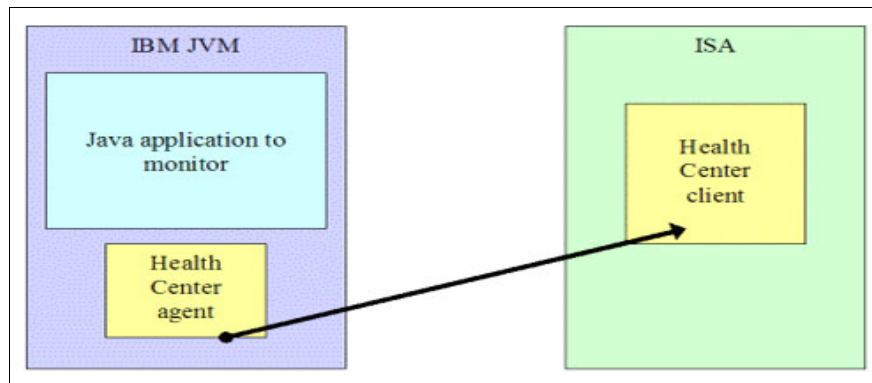


Figure 8-5 Health Center architecture

Using the Health Center

To use the Health Center to monitor Liberty, complete the following steps:

1. Download and install the IBM Support Assistant. The Health Center is a tool within the IBM Support Assistant.

You can also download the Health Center from the Liberty Repository at WASdev.net or from the Eclipse Marketplace at the following site:

<https://marketplace.eclipse.org/content/ibm-monitoring-and-diagnostic-tools-health-center>

2. Install the Health Center agent. For IBM SDK, Java Technology Edition, obtain updated Health Center agents. The IBM SDK contains a Health Center agent, but later versions of the agent that contain new function might be available. For the IBM SDK, Java Technology Edition, install the agent by extracting the downloaded package into the installation directory of the IBM SDK that runs your Java application. The installation directory is the parent directory of the `jre` directory.

Look at `JRE` in `jre/lib/ext` and verify that `healthcenter.jar` exists. If you do not have the agent installed, click the “Enabling an application monitoring” link in the connection wizard to install an agent.

3. Configure the Health Center agent. You can run the Health Center agent with the default settings, or you can configure various aspects, such as the port to use to communicate with the Health Center client, or which connection mode to use. You usually configure the agent by setting properties in a properties file. For Java applications, you can also set properties from a command line when you start the agent or attach the agent to a running application. You can set Health Center agent properties in the following ways:

- By setting system properties in the Health Center properties file, `healthcenter.properties`. This file is in the `jre/lib` directory of the JVM that contains the agent.
 - On the command line when you start the agent as system properties that you set by using the `-D` option of the Java command. For example, `-Dcom.ibm.java.diagnostics.healthcenter.agent.port=1999`.
 - On the command line when you start the agent as part of the `-Xhealthcenter` option of the Java command, when you start the agent and the application to be monitored at the same time.
4. Start the Health Center. The connection wizard starts and you enter the host name and port for your Liberty server. Then, click **Finish** to complete the connection.
 5. Examine the monitored items and information from the Health Center. The Health Center client is split into subsystems, each representing a component of the JVM.

Figure 8-6 shows the status of the monitored items.

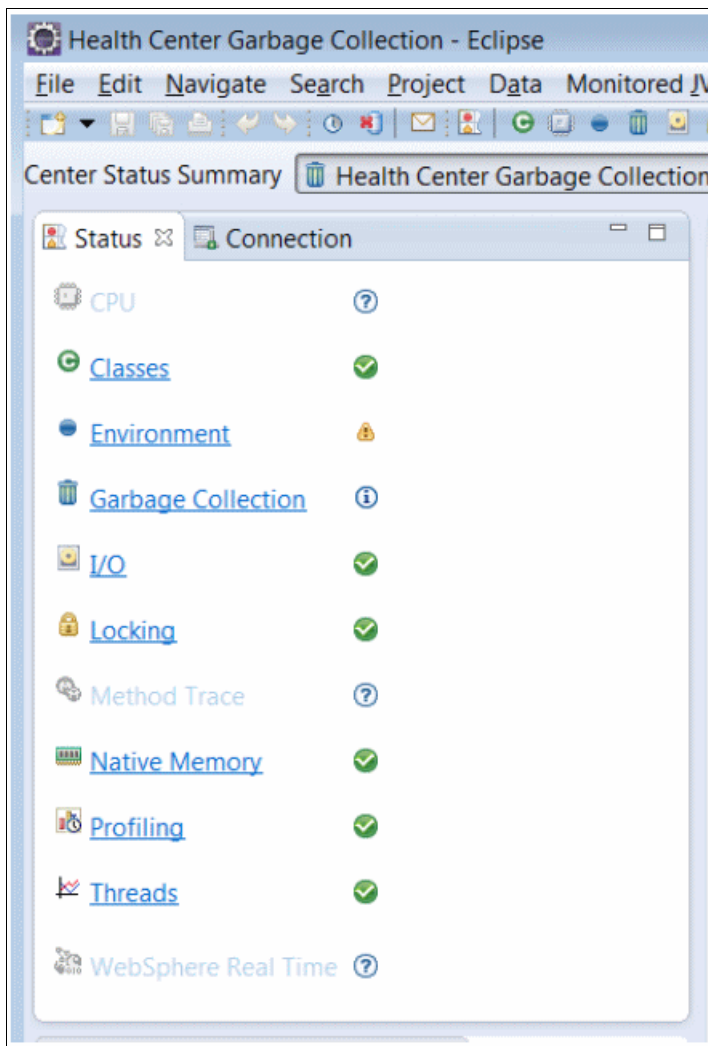


Figure 8-6 Health Center monitoring details

The first step in any Java application performance analysis is to study the garbage collection statistics. Click the **Garbage Collection** link to examine details.

There are two key items to review first for entry-level analysis:

- ▶ The Analysis and Recommendations section

The Analysis and Recommendations section in the lower-left corner provides useful tips and information that is based on built-in intelligence in Health Center. These tips can indicate garbage collection policy and heap size recommendations, observations about memory leaks or `System.gc()` calls, and more.

- ▶ The Summary panel

The Summary panel at the lower-right of the window contains data for the most important statistics to be concerned with.

You can examine other details, such as:

- ▶ Classes, where you can examine the data for Java classes. It displays the density of class loading over time, which classes were loaded, and when.
- ▶ Locking, which shows information about contention on inflated locks.
- ▶ Native Memory, where you can examine native memory usage, JVM native memory, and a breakdown of memory sizes, such as free physical, process virtual, and others.
- ▶ Profiling, which shows you the methods that are taking the most time.
- ▶ Threads, where you can examine the details of the live threads in your environment, such as number, current threads, and thread stack.

The status pane displayed in Figure 8-6 on page 154 shows a few areas that are currently unavailable. CPU utilization is a current restriction. Method Trace and WebSphere Real Time require additional configuration for statuses to appear for monitoring.

For more information about the Health Center, go to the following page:

<http://www.ibm.com/developerworks/java/jdk/tools/healthcenter>

8.5 Monitoring Liberty using other tools

There are a variety of analysis tools that can help you resolve issues when working with a Liberty server.

IBM Support Assistant

The IBM Support Assistant Team Server 5 provides a framework for IBM software products to deliver customized self-help information to the different tools within it. You can customize your IBM Support Assistant client by using the built-in Update capability to find and install new product features or support tools.

The IBM Support Assistant is a self-help problem determination and monitoring application, available at no charge. IBM Support Assistant 5 provides desktop tools, report generators, and web tools to take advantage of the server-based run time.

IBM Support Assistant provides a growing collection of tools that can be used in a Liberty environment. These tools include:

- ▶ Garbage Collection and Memory Visualizer (GCMV)

This tool helps with analyzing garbage collection behavior. You can use the GCMV to help visualize trends in native memory and Java heap growth in your server. This is useful to determine if you have a memory leak.

► **Memory Analyzer**

This tool is for analyzing Java heap memory by using heap dumps and system core files, in addition to Sun HPROF binary dumps. The Memory Analyzer tool provides memory leak detection and footprint analysis. The tool can also provide insight into where your application might be wasting memory, and can show the contents of method call stacks in addition to displaying the contents of any of the structures in your heap.

► **Thread and Monitor Dump Analyzer (TMDA)**

This tool is used to monitor thread dumps by presenting a list of the threads that existed at the time the thread dump was triggered. The tool looks for hangs, bottlenecks, and deadlocks. With this tool, you can visualize which threads are affected by slowdowns or held locks in the JVM, including information about the stack of each thread.

► **Interactive Diagnostic Data Explorer**

This tool displays a visual representation of your dump files. It works with system core files, IBM heap dump files, and IBM javacore files. The tool provides an editor in which you can run commands to find and view objects within your dumps and system cores. This can be useful when you are doing in-depth analysis of memory structures in your applications.

You can download the IBM Support Assistant Team Server 5 from the following page:

<http://www.ibm.com/software/support/isa/teamserver.html>

These tools can also be obtained from the Liberty Repository at WASdev.net or from the Eclipse Marketplace.

Admin Center web UI

The Admin Center provides monitoring information about a Liberty server or application by using the Explorer tool. In the Explorer tool, you can track used heap memory, loaded classes, active JVM threads, CPU usage, and other metrics, depending on the resource. The Monitor view shows the metrics graphically in chart form.

When monitoring a server, the charts include:

- Used heap memory
- Loaded classes
- Active JVM threads
- CPU usage
- Active Liberty threads

The Active Liberty threads chart is not visible in the Monitor view by default. You can add the chart by using the Edit Charts icon.

To use the Admin Center to monitor your environment, follow these steps:

1. Enable the `adminCenter-1.0`, `websocket-1.1`, and `monitor-1.0` features in `server.xml` as described in Example 8-1.

Example 8-1 Enabling the Admin Center

```
<featureManager>
  <feature>adminCenter-1.0</feature>
  <feature>websocket-1.1</feature>
  <feature>monitor-1.0</feature>
</featureManager>
```

The websocket feature provides a live view of the topology to the Admin Center. If the websocket feature is not enabled, the Admin Center periodically and frequently polls for changes.

The monitor feature provides more charts in the Monitor view based on your selection of either server or application, and the charts have more configuration options. For example, charts for web applications with multiple servlets, servers with active sessions, or servers with data sources display a drop-down list from which you can select resources to show in the chart.

2. Open the Admin Center and in the Toolbox, click the Explorer tool.
3. Select the server or the application that you want to monitor. Then, click **Monitor** in the navigation menu on the left.

You can scroll through the page and examine the charts for the data that is being monitored. For example, Figure 8-7 shows the chart for Used Heap Memory.

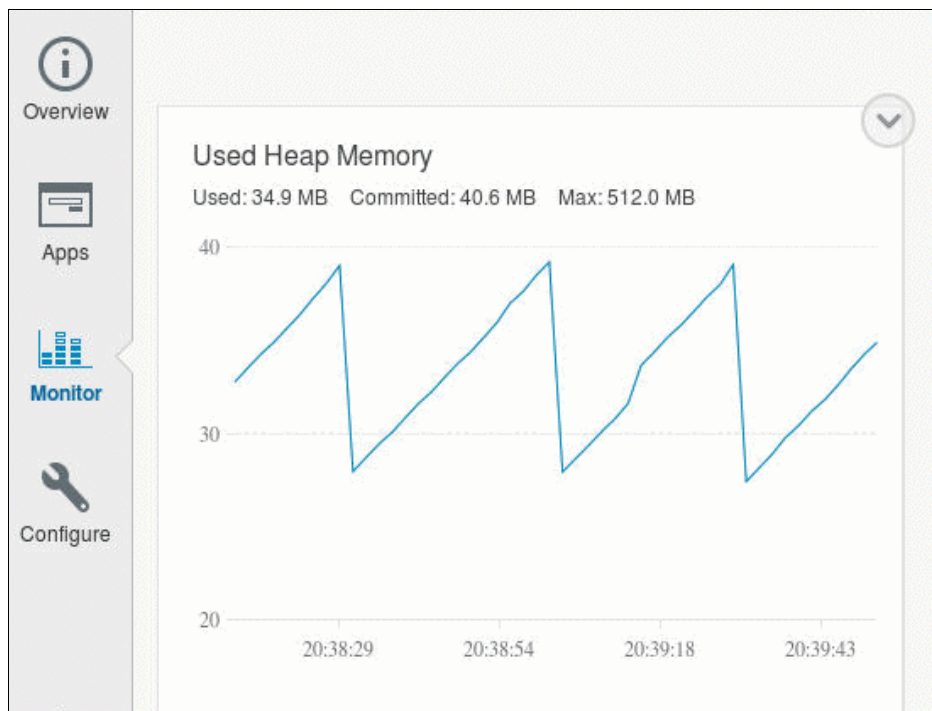


Figure 8-7 Monitoring used heap memory in graphical form

In addition to examining the data in graphical form, you can view just the raw data. To examine, click the Actions icon in the upper-right corner of any chart, and select **View chart data**. Figure 8-8 on page 158 shows an example of chart data collected for used heap memory.

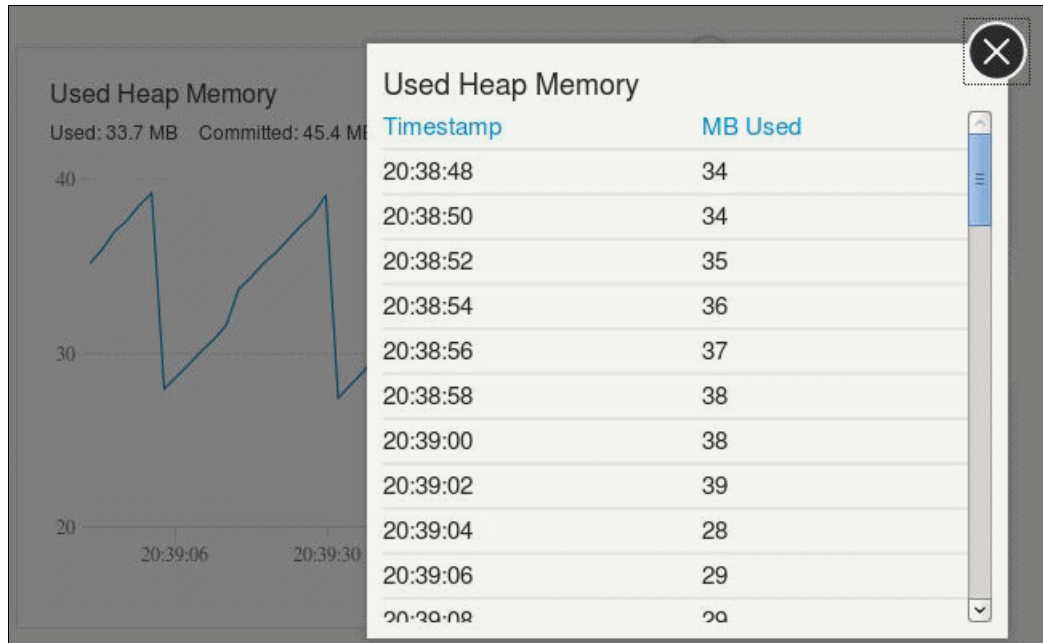


Figure 8-8 Used Heap Memory chart raw data

- By default, with the adminCenter-1.0 feature, you can obtain basic JVM metrics on all servers. You can obtain additional JVM server metrics when you enable the monitor-1.0 feature.

To examine more charts, click the Edit Charts icon in the upper right of the page. You can see the current charts and add any additional charts. For example, Figure 8-9 shows the options when adding a chart where you can add Active Liberty Threads. From Edit Charts, you can also remove any charts that you do not want to display. The metrics for threads are available on this server because the monitor-1.0 feature is also enabled.

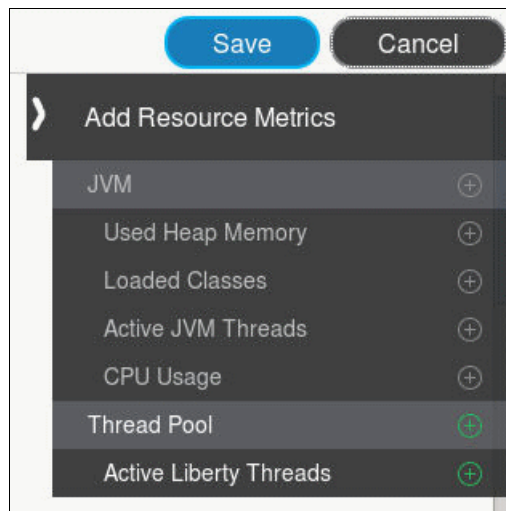


Figure 8-9 Edit charts in the admin center

Other tools

There are additional external monitoring tools that can be used to monitor the Liberty server.

IBM application performance management solutions help you manage the performance and availability of your applications. The application performance management products include:

- ▶ IBM Application Performance Management employs user experience monitoring, transaction tracking, and resource monitoring of application components to help identify, isolate, and resolve problems more quickly.
- ▶ IBM Tivoli Composite Application Manager for Application Diagnostics helps you to view the health of web applications and servers so that you can resolve performance issues faster, reduce downtime costs, and improve customer satisfaction.
- ▶ IBM Monitoring provides resource monitoring of infrastructure, application components, and cloud workloads to help speed slow transactions, resolve capacity issues, and prevent outages.
- ▶ IBM Application Diagnostics help you to gain code-level visibility into your applications and the health of your application servers to find performance bottlenecks in application code.
- ▶ The IBM CA Introscope tool can be used to proactively monitor complex Java and composite web applications and other emerging technology environments. It detects problems before they affect users and allows you to resolve issues quickly. For more information, see the following site:

<http://www.ibm.com/partnerworld/gsd/solutiondetails.do?solution=23517&expand=true>

- ▶ The AppDynamics tool, made available by AppDynamics, Inc., can be used to monitor the performance of applications across cloud computing environments. For more information, see the following site:

<http://www.appdynamics.com>

These are just a few of the tools that can be used to monitor the Liberty run time. There are many more third-party monitoring tools that can be used to help you monitor and ultimately tune your environment.

8.6 Tuning Liberty

You can tune parameters and attributes of Liberty for better performance. Liberty supports different attributes in the `server.xml` file to influence application performance.

To achieve better performance, the first place to start is to tune the JVM. Tuning the JVM is the most important tuning step, whether you are configuring a development or production environment. For a production environment, setting the minimum heap size and maximum heap size to the same value can provide the best performance by avoiding heap expansion and contraction.

The transport channel services are the next place to look to tune parameters. The transport channel services manage client connections, I/O processing for HTTP, thread pools, and connection pools. There are numerous attributes that you can tune to improve runtime performance, scalability, or both.

Other areas for tuning include the default executor and response time of servlets.

For more information about tuning Liberty, see the *WebSphere Application Server Performance Cookbook* at the following website:

<https://publib.boulder.ibm.com/httserv/cookbook>



Problem determination tools

Liberty includes logging and tracing capabilities similar to the full profile. Binary logging capabilities are available in V8.5.5. This chapter provides information about finding and viewing logs, taking traces, and taking dumps for use by IBM support.

This chapter includes the following topics:

- ▶ Text log and trace
- ▶ Binary log and trace
- ▶ Creating a dump of a Liberty server
- ▶ Event logging
- ▶ Request timing

9.1 Text log and trace

Liberty servers provide logging and tracing capabilities to help you monitor operations and determine the cause of problems. If you do not configure specific attributes for logging and tracing, the server environment uses a set of defaults. You can modify the default settings by specifying logging properties in the `server.xml` or `bootstrap.properties` file. Setting the properties in the `bootstrap.properties` file allow you to initiate tracing for server start.

In the following discussion, the `server.xml` file parameters are used to indicate how to configure the settings. For information about the equivalent `bootstrap.properties` settings, see the following website:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/rwlp_logging.html?cp=SSAW57_8.5.5%2F3-17-0-0

9.1.1 Configuring the server for logging

By default, log entries are written to two text files (`console.log` and `messages.log`) in the following location:

```
${wlp.install.dir}/usr/servers/server_name/logs
```

This location can be changed by using the **logDirectory** attribute in the `server.xml`.

By default, the `console.log` file contains audit-level messages. The **consoleLogLevel** attribute can be used to change this level. The valid values are INFO, AUDIT, WARNING, ERROR, and OFF.

The `messages.log` file contains all messages that are written or captured by the logging component. This log also contains time stamps and the issuing thread ID. The name of this file can be changed with the **messageFileName** attribute.

The logging can be controlled through the server configuration. Log files can be set to a maximum file size using the **maxFileSize** attribute. When that size is reached, the log *rolls over* to a new log file. The **maxFiles** attribute determines how many of each log file are kept. By default, a size limitation on the log file is not enforced.

Example 9-1 shows an example of specifying the logging properties in the `server.xml` file.

Example 9-1 Logging properties in a server.xml file

```
<logging logDirectory="/serverlogs/testserver1" />
```

For documentation of messages, see the IBM Knowledge Center at the following website:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/rwlp_messages.html?cp=SSAW57_8.5.5%2F3-17-0-7

9.1.2 Enabling tracing

Tracing can be enabled for a server. The trace entries are configured by using the **traceSpecification** attribute. The default trace level is `*=info`.

Trace entries are written to the `trace.log` file. The `trace.log` file is only created if additional or detailed trace is enabled in the `server.xml` file. The name of the file can be changed with the **traceFileName** attribute.

The server trace can expose sensitive data when tracing untyped data, such as bytes received over a network connection. The `suppressSensitiveTrace` attribute, when set to true, prevents potentially sensitive information from being exposed in log and trace files. The default value is false.

The `traceFormat` attribute controls the format of the trace log. The default format for Liberty is ENHANCED. You can also use BASIC and ADVANCED formats as in the full profile.

Example 9-2 shows how to specify trace settings to trace an application in the `server.xml` file.

Example 9-2 Settings in the `server.xml` file to trace an application

```
<logging traceSpecification="*=audit:com.myco.mypackage.*=debug"
traceFileName="trace.log"
maxFileSize="20"
maxFiles="10"
traceFormat="BASIC"/>
```

9.1.3 Using the WebSphere developer tools to configure logging and trace

The logging and trace settings can also be configured from the WebSphere developer tools. Use the following steps to complete that process:

1. In the Servers view, double-click **Server Configuration** to open the `server.xml` file.
2. Click **Add** to add a new element in the configuration. Select **Logging**.
3. With Logging selected in the configuration list, use the Logging Details panel to configure the settings (shown in Figure 9-1).

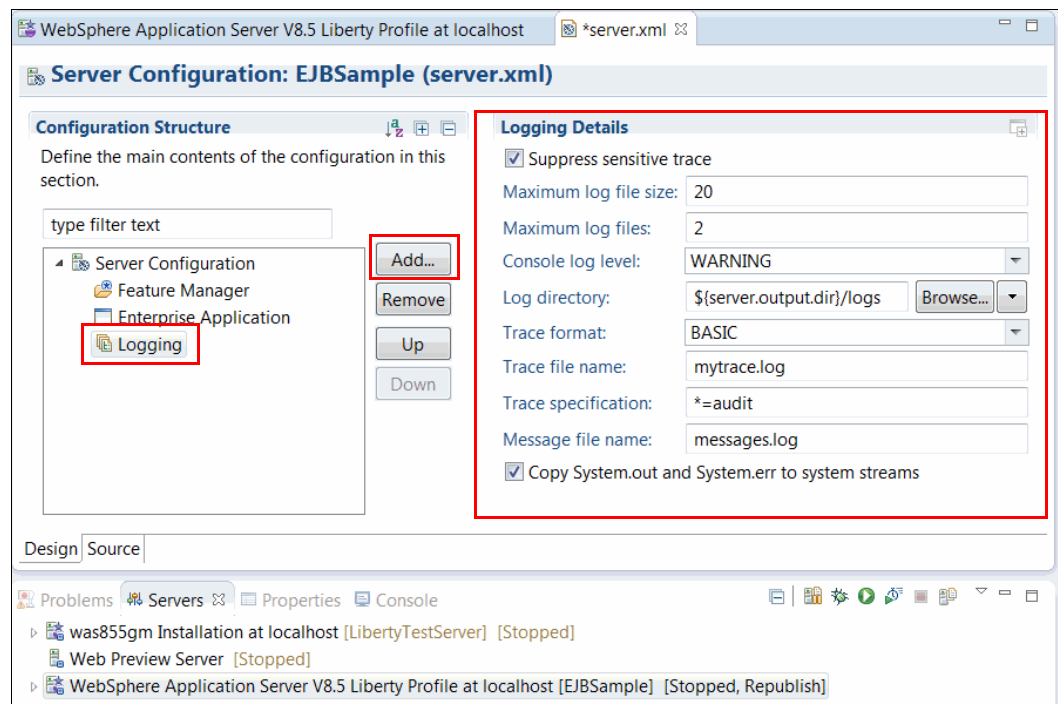


Figure 9-1 Configuring logging and tracing with the WebSphere developer tools

9.2 Binary log and trace

Binary logging is a high performance log and trace facility based on the full profile High Performance Extensible Logging (HPEL) technology.

Binary logging provides a convenient mechanism for storing and accessing log, trace, `System.err`, and `System.out` information produced by the application server or your applications. It is an alternative to the default log and trace facility, which provides the JVM logs and diagnostic trace files commonly named `messages.log` and `trace.log`.

9.2.1 Log data repository

The log data repository is a storage facility for log records. Log data is typically intended to be reviewed by administrators. The log data repository includes information from the following sources:

- ▶ Applications logging
- ▶ Server logging in `System.out` or `System.err`
- ▶ OSGi logging service at level `LOG_INFO` or higher (including `LOG_INFO`, `LOG_WARNING`, and `LOG_ERROR`)
- ▶ `java.util.logging` at level `Detail` or higher (including `Detail`, `Config`, `Info`, `Audit`, `Warning`, `Severe`, `Fatal`, and any custom levels at level `Detail` or higher)

9.2.2 Trace data repository

The trace data repository is a storage facility for trace records. Trace data is typically intended for use by application programmers or by the WebSphere Application Server support team. Trace data includes information from the following sources:

- ▶ Applications logging
- ▶ OSGi logging service at level `LOG_DEBUG`
- ▶ Server write to `java.util.logging` at levels below level `Detail` (including `Fine`, `Finer`, `Finest`, and any custom levels below level `Detail`).

9.2.3 Log and trace performance

Binary logging has a better performance than the default log and trace facility. One result is when using Binary logging the application server can run with trace enabled causing less impact to performance than tracing the same components using the default log and trace framework. Another result is that applications that frequently write to the logs can run faster when using binary logging.

Log and trace events are each stored in only one place

Log events, `System.out`, and `System.err` are stored in the log data repository. Trace events are stored in the trace data repository. Storing each type of event in only one location ensures that performance is not wasted on redundant data storage (shown in Figure 9-2 on page 165).

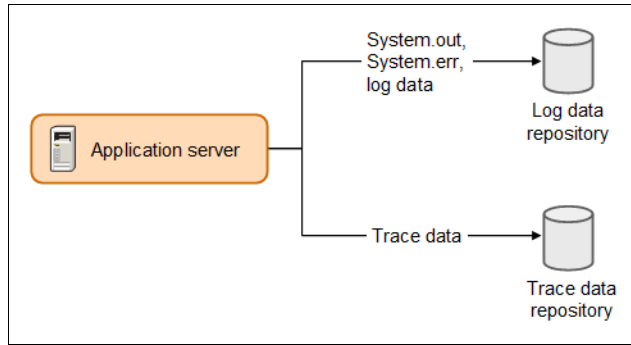


Figure 9-2 Location of where binary logging saves logs

Note: The console log should use `consoleLogLevel=OFF` where logging performance is important because all log and trace is already stored in the logdata and tracedata repositories. The console log is still used for anything the console writes to native `stderr` and `stdout`.

To learn more about configuring the binary logging service, see the IBM Knowledge Center at the following website:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.d/oc/ae/twlp_confHPEL.html?cp=SSAW57_8.5.5%2F1-0-2-10-1

9.2.4 Configuring binary logging

To enable binary logging to capture activity during server start, include the `binaryLogging-1.0` feature in `bootstrap.properties`, as shown in Example 9-3. This enables logging while the server configuration files are processed.

Example 9-3 Configuration of the binary logging using the `bootstrap.properties`

```
# Enable Binary Logging HPEL
websphere.log.provider=binaryLogging-1.0
```

If you only need to enable logging for activity that occurs after start, enable binary logging in `server.xml` as shown in Example 9-4. In this example, the log content is set to expire after 96 hours and the trace content is configured to retain a maximum of 1024 MB of data.

Example 9-4 Configuration of the binary logging parameters using the `server.xml`

```
<!-- Enable Binary Logging in server.xml -->
<server description="new server">

  <logging>
    <binaryLog purgeMinTime="96"/>
    <binaryTrace purgeMaxSize="1024"/>
  </logging>

</server>
```

Options, like time expiration and size of binary database, can be set in `server.xml` or in the `bootstrap.properties`.

Note: Enabling the binaryLogging requires a restart of Liberty.

Depending on what is to be monitored, use the following parameters in Table 9-1 to configure binary logging. If you prefer to see the detailed logs on run time, make a change after the server load, for this configuration and use the `server.xml`. It is easier to manage and does not require a reboot for the changes to take effect. If you prefer to see detailed logs include a code that runs during the server load, such as a feature module, `bootstrap.properties` is the correct choice. Because `bootstrap.properties` is read once before the `server.xml`, when `server.xml` starts to be read, binary logging parameters are already defined.

A table with the definitions to `server.xml` and `bootstrap.properties` is provided in Table 9-1.

Table 9-1 Binary logging attributes of server.xml and the equivalent properties of bootstrap.properties

Logging subelement	Attribute	Equivalent bootstrap.properties property
binaryLog	purgeMaxSize	com.ibm.hpel.log.purgeMaxSize
	purgeMinTime	com.ibm.hpel.log.purgeMinTime
	fileSwitchTime	com.ibm.hpel.log.fileSwitchTime
	bufferingEnabled	com.ibm.hpel.log.bufferingEnabled
	outOfSpaceAction	com.ibm.hpel.log.outOfSpaceAction
binaryTrace	purgeMaxSize	com.ibm.hpel.trace.purgeMaxSize
	purgeMinTime	com.ibm.hpel.trace.purgeMinTime
	fileSwitchTime	com.ibm.hpel.trace.fileSwitchTime
	bufferingEnabled	com.ibm.hpel.trace.bufferingEnabled
	outOfSpaceAction	com.ibm.hpel.trace.outOfSpaceAction

Example 9-4 on page 165, if defined in `bootstrap.properties`, appears as noted here in Example 9-5.

Example 9-5 Example of binary logging configuration in bootstrap.properties

```
# Enable Binary Logging HPEL in bootstrap.properties with options
websphere.log.provider=binaryLogging-1.0

com.ibm.hpel.log.purgeMinTime=96
com.ibm.hpel.trace.purgeMaxSize=1024
```

Note: After binary logging is enabled, the only text file that continues to receive updates is `console.log`. If you want to trail your logs or trace, you can use the **binaryLog** command-line tool with the `--monitor` option. For example, consider using `com.ibm.websphere.logging.hpel` API to read the log data and trace data repositories.

To learn more about configuring the binary logging parameters, see the IBM Knowledge Center at the following website:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/twlp_confHPEL.html?cp=SSAW57_8.5.5%2F1-0-2-10-1

9.2.5 Using the WebSphere developer tools to configure binary logging and trace

The logging and trace settings can be configured from the WebSphere developer tools. Use the following steps to complete that process:

1. In the Servers view, double-click **Server Configuration** to open the `server.xml` file.
2. Click **Add** to add an element in the configuration. Select **Logging**.
3. Select the new Logging in the list and click **Add** again. Select **Binary Log** or **Binary Trace**, as shown in Figure 9-3 and click **OK**.

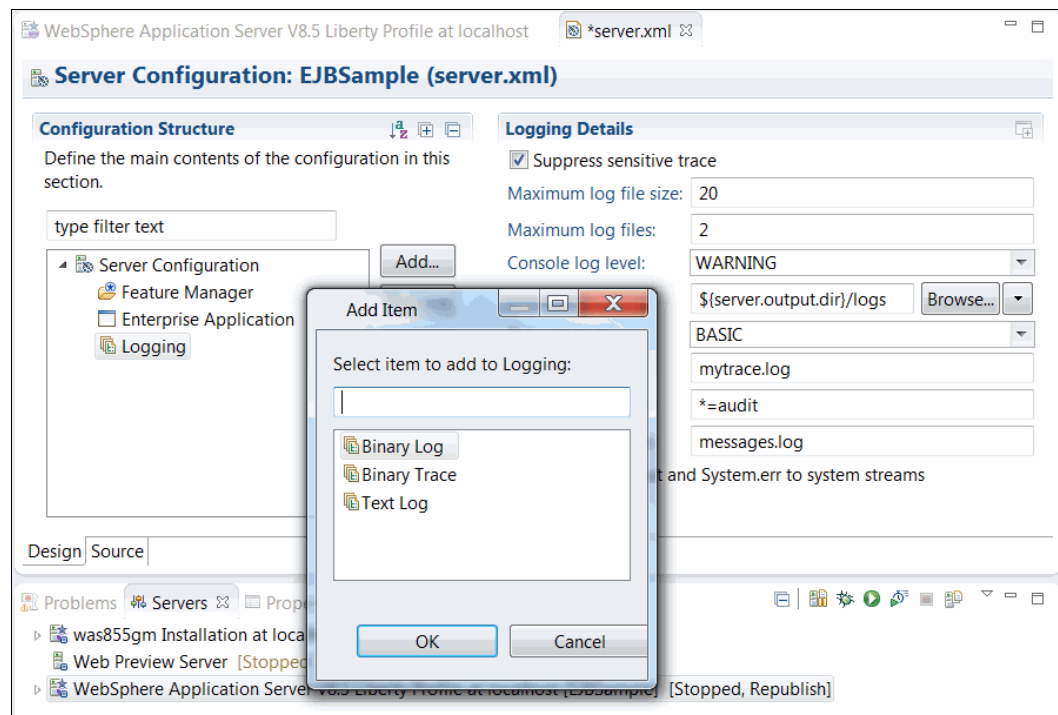


Figure 9-3 Configuring binary logging and tracing with the WebSphere developer tools

4. With **Binary Log** selected in the configuration list, use the Binary Log Details panel to configure the settings, as shown in Figure 9-4.

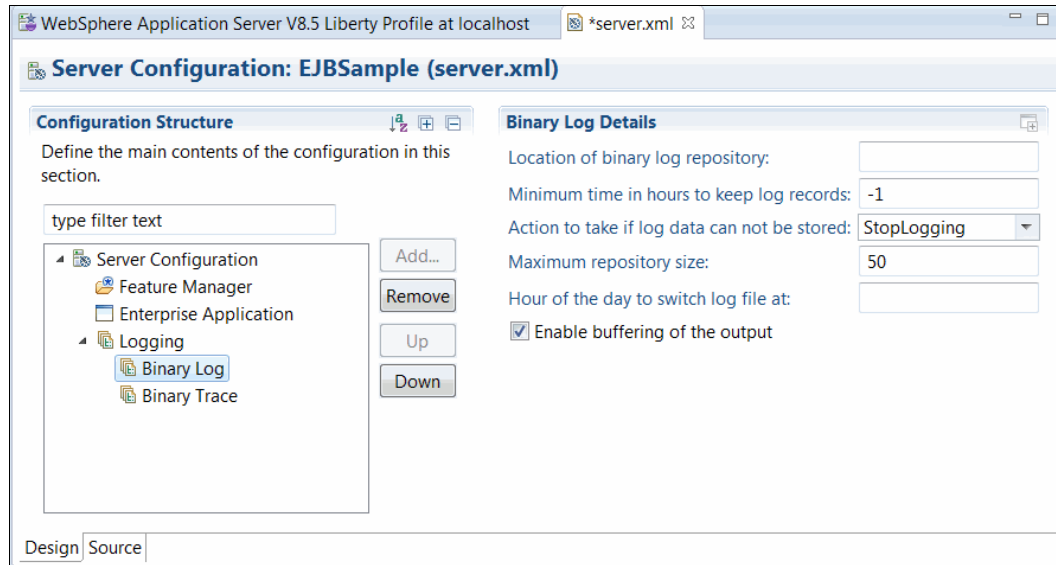


Figure 9-4 Configuring binary logging

5. Select **Binary Trace** and configure the settings.

9.2.6 Reading logs with the binaryLog command

Use the **binaryLog** command to view or copy the contents of a binary logging repository, or list the available server process instances in the repository. The **binaryLog** command is equivalent to the **logViewer** command in the profile `bin` directory of the full profile application server.

The binary log and trace facility writes to a repository in a binary format. You can view, query, and filter the repository by using the **binaryLog** command. The **binaryLog** command provides options for quickly converting repository contents into text in various formats, such as basic and advanced formats. The command also provides options to make acquiring the data you need from the logs easier. For example, it allows you to filter what log records you want by level, logger name, date, and time.

Example 9-6 shows the command syntax formula

Example 9-6 Command syntax formula

```
binaryLog action {serverName | repositoryPath} [options]
```

The following list defines the terms used in syntax formula that is shown in Example 9-6:

- `serverName`: Specify the name of a Liberty server with a repository from which to read.

- ▶ repositoryPath: Specify the path to a repository from which to read. This is typically the directory that contains the logdata and tracedata directories.
If serverName or repositoryPath are not specified on the command line, the task is performed against the default server instance, defaultServer, if it exists.
- ▶ binaryLog action: The value of options is different based on the value of action. Following are some action parameters that can be engaged:
 - view: Read a repository, optionally filter it, and create a human-readable version.
 - copy: Read a repository, optionally filter it, and write the contents to a new repository.
 - listInstances: List the server process instances in the repository.

A server instance is the collection of all log or trace records written from the time a server is started until it is stopped.

The **binaryLog** command outputs are placed in /bin directory inside the Liberty server root.

Examples

The following list notes several examples and the syntax used to enact that listed option:

- ▶ Display all events in the defaultServer repository between July 19th, 2013 and August 2nd, 2013:
binaryLog view --minDate=07/19/13 --maxDate=08/02/13
- ▶ Display new events from server myServer, whose specified level is WARNING or higher, by using the advanced format as the server writes them to the log repository:
binaryLog view myServer --monitor --minLevel=WARNING --format=advanced
- ▶ View log messages from a repository at /apps/server1/logs; include only those that were written to the error stream of a specific repository:
binaryLog view /apps/server1/logs --includeLogger=SystemErr
- ▶ View events from the defaultServer repository that occurred before June 14th, 2015 4:28 PM eastern daylight time:
binaryLog view --maxDate="06/14/15 16:28:00:000 EDT"
- ▶ Write events from the defaultServer repository that contains a 'thread' extension with value 'Default Executor-thread-4':
binaryLog view --includeExtension=thread="Default Executor-thread-4"
--format=advanced
- ▶ View the list of server instances in the defaultServer repository:
binaryLog listInstances

A list of instances is shown in Example 9-7.

Example 9-7 Return of the binaryLog listInstances command

Using D:\wlp\usr\servers\defaultServer\logs as repository directory.

Instance ID	Start Date
1358809441761	1/21/15 18:04:01:761 EST
1358864476191	1/22/15 9:21:16:191 EST
1358869523192	1/22/15 10:45:23:192 EST
1358871281166	1/22/15 11:14:41:166 EST
1358879829000	1/22/15 13:37:09:000 EST
1358892222067	1/22/15 17:03:42:067 EST

- ▶ View events from the defaultServer by using one of the instance IDs from Example 9-7 on page 169:

```
binaryLog view --includeInstance=1358871281166
```

- ▶ Copy events from the defaultServer, whose specified level is WARNING or higher, from the latest server instance to a new repository at /opt/wpl/toSupport directory:

```
binaryLog copy defaultServer /opt/wpl/toSupport --minLevel=warning  
--includeInstance=latest
```

Tip: The **binaryLog** tool is able to filter log and trace data most efficiently when used with the following filter options:

- ▶ --minDate
- ▶ --maxDate
- ▶ --includeThread
- ▶ --minLevel
- ▶ --maxLevel

To learn more about configuring the **binaryLog** command, see the IBM Knowledge Center at the following website:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/rwlp_logviewer.html?cp=SSAW57_8.5.5%2F1-0-2-10-0

9.3 Creating a dump of a Liberty server

To capture the state information of a Liberty server, use the **dump** command. It can be useful for problem diagnosis of a Liberty server. The file that the **dump** command generates contains server configuration, log information, and details about the deployed applications. Also, it can be used against a running or stopped server; however, if the server is running, the following additional information is gathered:

- ▶ State of each OSGi bundle in the server
- ▶ Wiring information for each OSGi bundle in the server
- ▶ Component list managed by the Service Component Runtime (SCR)
- ▶ Detailed information about each component from SCR

Example 9-8 shows the running of a **dump** command.

Example 9-8 Running the server dump command

```
server dump server1 --archive=/opt/wpl/dump/server1_dump.zip
```

You can also create a server dump of the Liberty server by using the WebSphere developer tools from the menu for the server in the Servers view.

9.4 Event logging

As part of the monitoring and diagnostic capabilities, Liberty generates events at various components of Java Platform, Enterprise Edition to track the requests. The eventLogging-1.0 feature logs such events when the application requests are running. By using this feature, the user can track the requests that are running in Liberty. Each request is associated with a unique correlator called the request ID and the context information that helps the user to understand the request-specific data.

The event logging feature is controlled through the server configuration. This feature is configured in the `server.xml` file, and the default attributes for event logging can also be overridden using the `server.xml`, shown in Example 9-9.

Example 9-9 server.xml

```
<featureManager>
  <feature>eventLogging-1.0</feature>
</featureManager>
<eventLogging minDuration="10ms" />
```

To learn more about overriding the default attributes, see the IBM Knowledge Center at the following website:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.d oc/ae/rwlp_feature_eventLogging-1.0.html?lang=en

Parsing the event log entries in the `messages.log` file

The event logs capture the information of the events in the following format, shown in Example 9-10.

Example 9-10 Messages.log file

```
[Log mode] [Request Identifier] # [Event Type] # [Context Information] #
[Duration] (optional)
```

The following list defines the terms used in Example 9-10:

- ▶ `Log mode` indicates whether the log was recorded at the entry to the event or the exit from the event. `BEGIN` refers to the entry to the event and `END` refers to the exit from the event.
- ▶ `Request identifier` is a unique string that is assigned to each request. This can be used for filtering events that belong to a particular request.
- ▶ `Event type` provides information about the event source. The event type can be used for filtering events of a specific type.
- ▶ `Context information` of the event provides details relevant to the event type. The information varies depending on the event type. Context information can contain multiple sections and are separated by | (spacelsspace).
- ▶ `Duration` indicates the time that is taken by the event. The duration appears only in the exit event entries.

9.5 Request timing

The requestTiming-1.0 feature provides diagnostic information when the duration of any request exceeds the configured threshold, shown in Example 9-11.

Example 9-11 requestTiming-1.0 feature

```
<featureManager</>
  <feature/>requestTiming-1.0</feature>
</featureManager>
```

The request timing feature can track the duration of every request that is coming into the system. You can configure the feature to watch for slow and hung requests, as shown in Example 9-12.

Example 9-12 Configure requestTiming feature

```
<requestTiming
  includeContextInfo="true"
  slowRequestThreshold="10s"
  hungRequestThreshold="600s"
  sampleRate="1"
/>
```

The following list defines the terms that are shown in Example 9-12:

- ▶ `includeContextInfo` indicates if the context information details are included in the log output. The default is `true`.
- ▶ `slowRequestThreshold` is the duration of time that a request can run before being considered slow. The default is 10 seconds. You can set to 0 to disable slow request checking.
- ▶ `hungRequestThreshold` is the duration of time that a request can run before being considered hung. The default is 10 minutes. You can set to 0 to disable hung request checking.
- ▶ `sampleRate` is the rate at which the sampling should happen for the slow request tracking. The default is every request (1).



Intelligent Management

In this chapter, you are introduced to the Intelligent Management capabilities of Liberty.

The following topics are covered:

- ▶ Introduction to Intelligent Management
- ▶ Dynamic routing
- ▶ Auto scaling
- ▶ Maintenance mode
- ▶ Health management

10.1 Introduction to Intelligent Management

Intelligent Management provides a virtualized infrastructure that redefines the traditional concepts of Java Platform, Enterprise Edition resources and applications, and their relationships. This application infrastructure virtualization allows the product to automate operations in an optimal manner, increasing the quality of service. Intelligent Management extends the quality of service that is provided by your middleware environment. In short, you experience the benefits of an autonomic middleware environment, which is self-configuring, self-protecting, and self-optimizing.

Intelligent Management for Liberty includes the following primary features:

- ▶ **Dynamic routing**
Routes HTTP requests automatically to the active Liberty servers in a collective.
- ▶ **Auto scaling**
Starts, stops, or creates Liberty servers in clusters automatically, based on scaling policies
- ▶ **Health management**
Allows you to specify conditions to watch for and diagnostic actions to automatically take when the conditions are observed. You can monitor the status of your application servers, sense problem areas, and then respond to these problem areas before an outage occurs.
- ▶ **Maintenance mode**
Allows you to prevent the disruption of client requests by routing client traffic that is targeted for a server that is in maintenance mode to another server.

It is important to note that the Intelligent Management features in Liberty V8.5.5.7 are only a subset of the Intelligent Management features in WAS Classic V8.5.5.

10.2 Dynamic routing

Routing of web requests to servers in a Liberty collective is done by using a web server with the WebSphere plug-in. With static routing, the information used to route requests is read from a plug-in configuration file. The routing information in the file contains the endpoint information of the servers in the collective. The routing information for each server is generated by invoking an administrative MBean method on each server. For multiple servers, the routing information for the WebSphere plug-in must be merged. The WebSphere Plug-in detects when a server or application is unavailable, when communication errors occur with the server or application. This topology becomes more complicated as servers or applications are added. The routing information in the file must be regenerated for each change. The changes must then be merged into the configuration file that is provided to the WebSphere plug-in.

The dynamicRouting-1.0 feature enables routing of HTTP requests to members of Liberty collectives without having to regenerate the WebSphere plug-in configuration file when the environment changes. When servers, cluster members, applications, or virtual hosts are added, removed, started, stopped, or modified, the new information is dynamically delivered to the WebSphere plug-in. Requests are routed based on up-to-date information.

Figure 10-1 on page 175 shows an example of a dynamic routing topology.

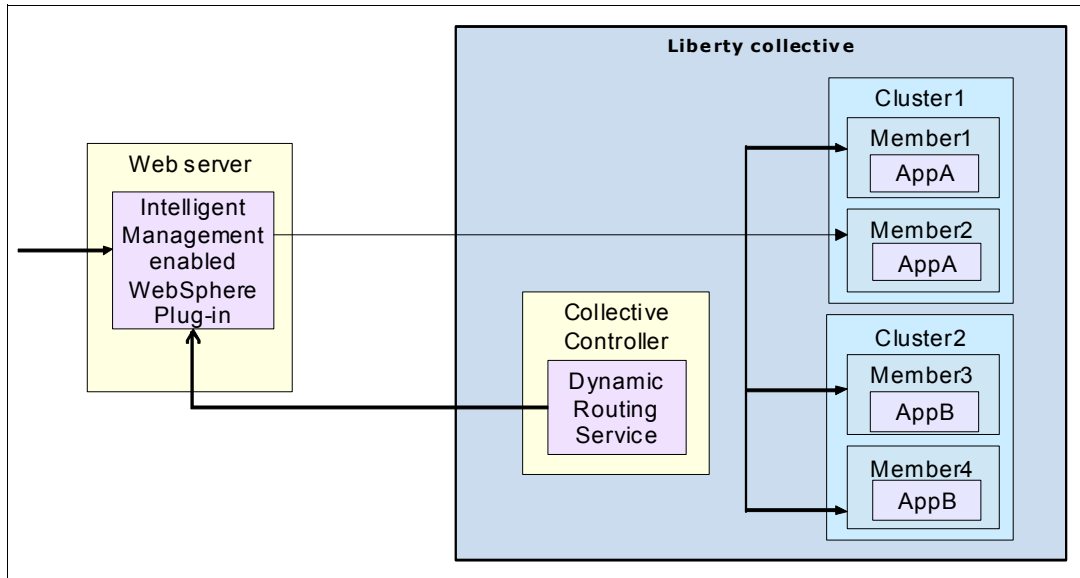


Figure 10-1 Dynamic routing topology

Collective members publish member-specific routing information to the collective controller. The controller uses the published information to create and maintain the routing information for the collective. The controller delivers the routing information to the plug-in as the routing configuration or state changes. As new controllers with the dynamic routing feature enabled are added to the collective, the new controller information is also delivered to the plug-in. The plug-in can use the new controllers to fail over to if the original controller becomes unavailable.

The dynamic routing service maintains the current routing information for all of the applications in the collective. The Intelligent Management-enabled WebSphere plug-in connects to the dynamic routing service and the service delivers up-to-date routing information to the plug-in. As the servers and applications in the collective change, the new routing information is delivered to the plug-in. The plug-in then routes requests successfully into the changed topology. The dynamic routing service also delivers server and application start and stop events to the plug-in.

The main benefit of dynamic routing is that routing information is maintained so that web requests are routed successfully as the routing topology changes. The plug-in does not have to use communication errors to determine whether an application or server is available. Also, the plug-in routing configuration file does not need to be maintained manually. This reduces the chances of error in the environment and saves administrative time.

10.2.1 Configuring dynamic routing

To use the dynamic routing feature, you need to configure the `dynamicRouting-1.0` feature in the `server.xml` file on the controller. The `dynamicRouting-1.0` feature provides the dynamic routing service.

To configure dynamic routing in your environment, use the following steps:

- ▶ Install and configure the Web Server Plug-in for WebSphere Application Server
- ▶ Configure a collective
- ▶ Configure the dynamic routing feature
- ▶ Generate the keystore and plug-in configuration files
- ▶ Configure the web server host
- ▶ Start the web server

For more details about configuring a dynamic routing topology in your environment, use the following steps:

1. Install and configure the Web Server Plug-in for WebSphere Application Server

The first step is to install a web server that is supported by the Web Server Plug-in for WebSphere Application Server, such as the IBM HTTP Server. Then, download and install the IBM Installation Manager. To download and get more information about the IBM Installation Manager, see the following page:

<http://www.ibm.com/support/docview.wss?uid=swg27025142>

You can use the Installation Manager to access online product repositories to install the Web Server Plug-in for WebSphere Application Server and the needed interim fix for the dynamic routing feature. The interim fix needed is APAR number PI27023.

2. Configure a collective

For details about configuring a collective, controller, and cluster, see Chapter 5, “Administering the WebSphere Liberty profile” on page 71.

3. Configure the dynamic routing feature

Add the `dynamicRouting-1.0` feature to the `server.xml` file on the controller as shown in Example 10-1. Ensure that you update the `server.xml` on each controller that you have in your topology.

Example 10-1 Adding the dynamicRouting feature

```
<featureManager>
  <feature>collectiveController-1.0</feature>
  <feature>dynamicRouting-1.0</feature>
</featureManager>
```

After the feature is added, start the controller.

4. Generate the keystore and plug-in configuration files

To generate the keystore and plug-in configuration files, you use the `dynamicRouting setup` command. The `--host` and `--port` arguments identify the collective controller that can process the command. The `--user` and `--password` arguments are the administrative user ID and password for authenticating with the controller. If you do not provide the password value on the command line, you are prompted to enter it when running the command. You also need to include details for the `--pluginInstallRoot` and `--webServerNames` arguments.

Example 10-2 shows an example of generating the keystore and plug-in configuration files.

Example 10-2 Generating the keystore and plug-in configuration files

```
$ ./dynamicRouting setup --host=1exbz181072.1ex.dst.ibm.com --port=9449
--user=liberty --password= --keyStorePassword=liberty
--pluginInstallRoot=/opt/IBM/WebSphere/Plugins --webServerNames=webserverITS01
Enter password --password:
```

Generating WebSphere plug-in configuration file for web server webserverITS01

SSL trust has not been established with the target server.

Certificate chain information:

Certificate [0]

Subject DN: CN=1exbz181072.1ex.dst.ibm.com, OU=controllerITS01, O=ibm, C=us

Issuer DN: OU=controllerRoot, O=b63a4192-885d-4c1b-b3de-6b4b4b0a5dc2,
DC=com.ibm.ws.collective
Serial Number: 1,041,316,133,498,593
Expires: 8/24/20 3:35 PM
SHA-1 digest: 07:63:DA:7F:D3:4C:20:B8:E3:32:A7:31:5C:34:4B:B9:71:EF:28:53
MD5 digest: 9F:EC:27:81:CC:C1:A0:5A:32:57:FB:29:F4:95:4A:DA

Certificate [1]

Subject DN: OU=controllerRoot, O=b63a4192-885d-4c1b-b3de-6b4b4b0a5dc2,
DC=com.ibm.ws.collective
Issuer DN: OU=controllerRoot, O=b63a4192-885d-4c1b-b3de-6b4b4b0a5dc2,
DC=com.ibm.ws.collective
Serial Number: 1,041,305,392,743,106
Expires: 8/19/40 3:34 PM
SHA-1 digest: 06:3F:FC:23:66:B1:32:3D:12:E0:3B:86:D0:D9:CA:14:C6:B4:4E:9B
MD5 digest: 17:32:B6:6E:DC:9F:55:FB:EC:7D:86:C1:C7:51:17:0B

Do you want to accept the above certificate chain? (y/n) y
Successfully completed MBean request to the controller.
Successfully generated WebSphere plug-in configuration file plugin-cfg.xml
Generating keystore for web server webserverITS01
Successfully completed MBean request to the controller.
Successfully generated keystore plugin-key.jks.

Generated WebSphere plug-in configuration file plugin-cfg.xml
for web server webserverITS01.
Also generated keystore file plugin-key.jks that enables secure
communication between the Dynamic Routing service and
clients. The file contains personal certificate issued
to DN CN=liberty,OU=client,O=ibm,C=us. Ensure the liberty user exists in the
user registry and has a role assigned.

If you are using quick start security, add the following line to
the controller server.xml file and update the password:

```
<quickStartSecurity user="liberty" password=""/>
```

If you are using basic registry, add the following lines to
the controller server.xml file and update the password:

```
<basicRegistry id="basic" realm="ibm/api">  
  <user name="liberty" password=""/>  
</basicRegistry>
```

```
<administrator-role>  
  <user>liberty</user>  
</administrator-role>
```

Copy the WebSphere plug-in configuration file to the directory specified
in the WebSpherePluginConfig directive in the IBM HTTP Server httpd.conf
file. Copy keystore file plugin-key.jks to a directory on the
web server host, and run "gskcmd" to convert the keystore to CMS format and
to set personal certificate as the default.

For example:

```
gskcmd -keydb -convert -pw <<password>> -db /tmp/plugin-key.jks -old_format jks  
-target /tmp/plugin-key.kdb -new_format cms -stash
```

```
gskcmd -cert -setdefault -pw <<password>> -db /tmp/plugin-key.kdb -label
default
```

```
Copy resulting /tmp/plugin-key.kdb, .sth, .rdb files to the directory
/opt/IBM/WebSphere/Plugins/config/webserverITS01/
$
```

You can see that the output of the command is the plug-in configuration file, `plugin-cfg.xml`, and a keystore containing a personal and signer certificates, `plugin-key.jks`. Both of the files are generated in the directory from where you ran the **dynamicRouting setup** command.

The details of the `plugin-cfg.xml` file are shown in Example 10-3. Note the `<IntelligentManagement>` element and details in the stanza. Included is information about the connector, which indicates the controller details. If you have multiple controllers, there will be a connector element for each controller.

Example 10-3 The plugin-cfg.xml configuration file

```
<?xml version="1.0" encoding="UTF-8"?><!--HTTP server plugin config file for web
serverITS01 generated on 2015.08.27 at 11:34:32 EDT-->

<Config ASDisableNagle="false" AcceptAllContent="false" AppServerPortPreference=
"HostHeader" ChunkedResponse="false" FIPSEnable="false" IISDisableNagle="false"
IISPluginPriority="High" IgnoreDNSFailures="false" RefreshInterval="60" Response
ChunkSize="64" SSLConsolidate="false" TrustedProxyEnable="false" VHostMatchingCo
mpat="false">
  <Log LogLevel="Error" Name="/opt/IBM/WebSphere/Plugins/logs/webserverITS01/ht
tp_plugin.log"/>
  <Property Name="ESIEnable" Value="true"/>
  <Property Name="ESIMaxCacheSize" Value="1024"/>
  <Property Name="ESIInvalidationMonitor" Value="false"/>
  <Property Name="ESIEnableToPassCookies" Value="false"/>
  <Property Name="PluginInstallRoot" Value="/opt/IBM/WebSphere/Plugins/" />
<!-- Configuration generated using httpEndpointRef=defaultHttpEndpoint-->
<!-- The default_host contained only aliases for endpoint defaultHttpEndpoint.
The generated VirtualHostGroup will contain only configured web server
ports:
  webserverPort=80
  webserverSecurePort=443 -->
  <Property Name="Keyfile" Value="/opt/IBM/WebSphere/Plugins/config/webserverIT
S01/plugin-key.kdb"/>
  <Property Name="Stashfile" Value="/opt/IBM/WebSphere/Plugins/config/webserver
ITS01/plugin-key.sth"/>
  <IntelligentManagement>
    <Property name="webserverName" value="webserverITS01"/>
    <ConnectorCluster enabled="true" maxRetries="-1" name="default" retryInter
val="60">
      <Property name="uri" value="/ibm/api/dynamicRouting"/>
      <Connector host="lexbz181072.lex.dst.ibm.com" port="9449" protocol="https">
        <Property name="keyring" value="/opt/IBM/WebSphere/Plugins/config/we
bserverITS01/plugin-key.kdb"/>
      </Connector>
    </ConnectorCluster>
  </IntelligentManagement>
</Config>
```

5. Configure the web server host

The last part of this configuration is to make all of the necessary files available to the WebSphere Plug-in the web server host. First, copy the generated `plugin-cfg.xml` and `plugin-key.jks` files to a temporary directory on the web server host.

Next, set up secure communication between the plug-in and the dynamic routing service. The keystore generated by the **dynamicRouting setup** command, `plugin-key.jks`, is not in a format that can be used by the WebSphere Plug-in. The generated file must be converted to a format that can be used by the plug-in, the CMS format.

The **gskcmd** command is used to convert the keystore. The command is provided with the web server installation. Example 10-4 shows the **gskcmd** command.

Example 10-4 Example of the gskcmd

```
$ ./gskcmd -keydb -convert -db /tmp/IHS/plugin-key.jks -old_format jks -target /tmp/IHS/plugin-key.jks -new_format cms -stash -pw liberty -new_pw liberty
$
```

Copy both the `plugin-cfg.xml` file and all of the converted keystore files to the plug-in config directory. Finally, copy the `plugin-cfg.xml` to the directory specified in the `WebSpherePluginConfig` directive in the IBM HTTP Server (IHS) `httpd.conf` file.

6. Start the web server

Start the web server and begin dynamically routing to the applications installed in the Liberty collective.

10.3 Auto scaling

Auto scaling provides an autonomic scaling capability of Liberty servers. Auto scaling provides JVM elasticity to clusters. With JVM elasticity, auto scaling features dynamically adjust the number of running Liberty servers in a cluster based on workload. As workload goes up, auto scaling starts cluster members. When workload goes down, auto scaling stops cluster members.

Auto scaling also provides Liberty elasticity to clusters. With Liberty elasticity, auto scaling features can install Liberty software onto a registered host and create a new server. The number of available servers grows when application demand is high and shrinks when application demand is low. Because support for Liberty elasticity includes support for JVM elasticity, auto scaling can also start or stop servers based on workload.

To dynamically adjust the number of servers used to service your workload, auto scaling uses user-defined scaling policies with the current state of the cluster to determine whether a scaling action needs to occur. The machines that host the cluster members provide the data on the workload.

Auto scaling is a configurable feature that runs on top of the collective feature. When used in coordination with the dynamic routing feature, auto scaling can provide elasticity to clusters to support fluctuations in workload demands.

The auto scaling feature starts and stops servers or provisions new servers as needed to meet scaling policies. As the server state changes, dynamic routing ensures that requests are sent correctly to the available servers without requiring an administrator to update the routing configuration file. It offers a way to automate cluster monitoring, relieving you of the need to gauge how many servers are needed to support the application at any one time. In addition to automating this responsibility, auto scaling provides a concise and simple manner for managing cluster policy.

Figure 10-2 shows an example of the auto scaling and dynamic routing topology.

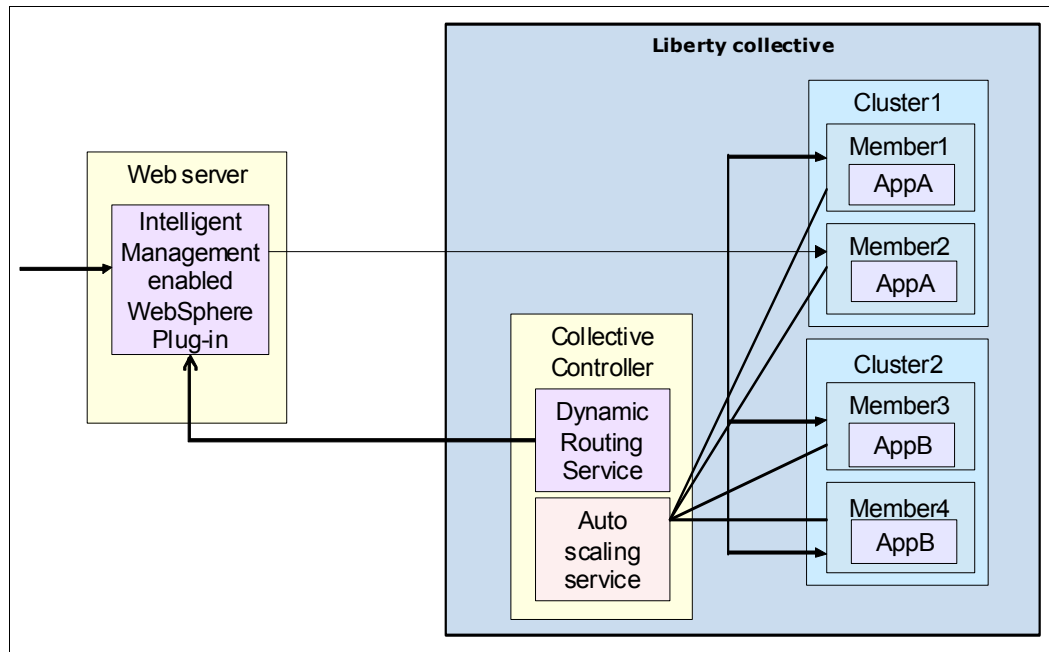


Figure 10-2 Auto scaling and dynamic routing topology

10.3.1 Auto scaling features

To set up an auto-scalable cluster, you need at least one collective controller with at least two member servers joined to the controller. Then, you need to configure the auto scaling features. There are two features that provide the auto scaling capabilities for a Liberty cluster. The features include:

- ▶ Scaling controller
- ▶ Scaling members

Scaling controller

An auto scaling cluster is a server cluster that can expand and contract depending on the workload in your environment. The scaling controller maintains the view of all the clusters in the topology and is in charge of starting and stopping cluster members and provisioning new servers. The scaling controller does this by analyzing performance metrics sent by cluster members and comparing them against defined scaling policies. When a metric violates a policy threshold, it decides whether to scale in or scale out.

Collective controllers are required because they provide administration functionality that leverages the ability of the collective controller to manage the scaling controller. If replica sets are being used, all collective controller members must be scaling controllers. However, only one of the running scaling controllers can make decisions. When using replica sets if a controller is stopped, another running scaling controller takes over for it.

Scaling members

Scaling members are collective members. The scaling member monitors the resources within the server and its host and when needed, it sends this information back to the scaling controller.

Scaling members are divided into two categories: host leaders and host followers. Host leaders are scaling members that are elected to talk to the scaling controller. Only one scaling member on a host is elected the host leader. All other members of the cluster then become host followers who simply monitor their resource usage and pass reports to the host leader. The host leader is the only scaling member that communicates with the scaling controller.

All scaling member servers must belong to a cluster because all scaling policy information is applied at the cluster level. A cluster has a unique name within a Liberty collective. All Liberty servers that specify the same cluster name within the same collective are members of the same cluster.

10.3.2 Scaling policies

Scaling policies allow you to control the scaling behaviors of clusters. Scaling policies are used by the scaling controller to determine when to start or stop members of a cluster. A cluster is scaled to meet a minimum or maximum number of servers per cluster or to meet resource demands of a cluster.

Scaling can also happen based on the resource consumption of a cluster. The scaling policies allow you to set the minimum and maximum thresholds for server resources, such as CPU, memory, and heap. The scaling member sends the metric resource data to the scaling controller. If a cluster violates a metric that is defined in a scaling policy, the scaling controller reacts accordingly, starting a member if a maximum is breached, or stopping a member if usage drops below a minimum.

Scaling policies are defined in two ways:

- ▶ By using a built-in scaling policy
- ▶ Defining the `<defaultScalingPolicy>` or `<scalingPolicy>` elements in the `server.xml` file on the scaling controller

The built-in scaling policy

By default, a built-in scaling policy is embedded in the scaling controller. The built-in scaling policy indicates:

- ▶ A minimum of two cluster members, if available, are kept active. The minimum number might not be met if some or all of the members are exceeding the metric thresholds.
- ▶ An additional cluster member is started when the average CPU, heap, or memory use of all active members exceeds 90%.
- ▶ A cluster member is stopped when the average CPU and heap use drops below 30%.

Defining scaling policies, the <defaultScalingPolicy>

You can override the built-in scaling policy if needed by defining scaling policies in the `server.xml` file on the scaling controller. To override the built-in policy, use the `<defaultScalingPolicy>` element. The `defaultScalingPolicy` indicates the default policy to use for any auto-scaled cluster that does not have a specific scaling policy defined.

Example 10-5 shows an example of using the `<defaultScalingPolicy>` element. The `<defaultScalingPolicy>` element specifies only the attributes that it intends to overwrite. All other attributes are inherited from the built-in scaling policy. In this example, the minimum and maximum number of scaling members for a cluster is set. The controller uses this information to ensure that the number of running servers falls within this bound.

Example 10-5 The defaultScalingPolicy

```
<scalingDefinitions>
  <defaultScalingPolicy enabled="true" min="2" max="3"/>
</scalingDefinitions>
```

The `enabled` value is set to `true`, which tells the scaling controller to use this policy. Setting the `enabled` value to `false` prevents the scaling controller from making scaling decisions.

The policy requires at least two members be running and no more than three should be running. If only one scaling member is started, the controller starts another member. If the maximum number is three and four members are started, the controller stops one member of the cluster to meet the maximum.

The thresholds for CPU, heap, and memory metrics are inherited from the built-in scaling policy. When the average CPU, heap, or memory use of all active members exceeds 90%, start a member. When the average CPU and heap use drops below 30%, stop a member.

Defining scaling policies, the <scalingPolicy>

You can also override the built-in policy by using the `<scalingPolicy>` element. The `<scalingPolicy>` element provides fine-grained control of each cluster's scaling policy. The thresholds defined in the policy are targeted at a specific cluster or clusters by using the `<bind>` element. In addition, the metric policy for a `<scalingPolicy>` element must be defined, or they will not exist. This allows granular control of which metrics are analyzed when making scaling decisions.

Example 10-6 shows a scaling policy defined in the `server.xml` file where the `<scalingPolicy>` element is used.

Example 10-6 The scalingPolicy

```
<scalingDefinitions>
  <scalingPolicy enabled="true" min="2" max="3">
    <bind cluster="ITSOCluster">
      <metric name="cpu" min="30" max="80" />
    </scalingPolicy>
</scalingDefinitions>
```

The policy is targeted to the `ITSOCluster` and indicates a CPU metric, with a minimum of 30% and maximum of 80% CPU usage. A member is started when the average members CPU resource use exceeds 80% and the member is stopped when the average is less than 30%. Heap and memory metric data are not used in the scaling decisions by the scaling controller. Also, the minimum number of members that should be running is two and the maximum is three members.

Figure 10-3 shows an example of a scaling policy in use in Java virtual machine (JVM) elasticity. In this example, a cluster member processes requests from a web server. Eventually, the traffic becomes heavy enough to cause it to violate the CPU threshold, triggering a message containing the metrics data to be sent to the scaling controller. The scaling controller, noticing that the member has violated the upper threshold, starts another cluster member to balance the load of requests. With another member started, the CPU of both members stabilizes to something reasonable.

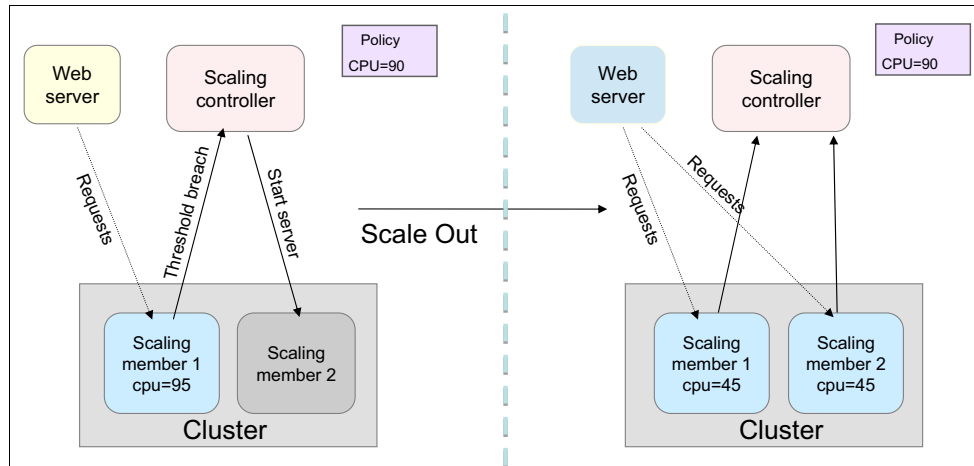


Figure 10-3 Scaling policies and JVM elasticity

Figure 10-4 shows an example of a policy in use in Liberty elasticity. In this example, two cluster members are processing requests from a web server. Again, the traffic becomes heavy enough to cause a violation in the CPU threshold, triggering a message containing the metrics data to be sent to the scaling controller. The scaling controller, noticing that the members have violated the upper threshold, starts another cluster member to balance the load of requests. With another member started, the CPU of all members stabilizes to something reasonable.

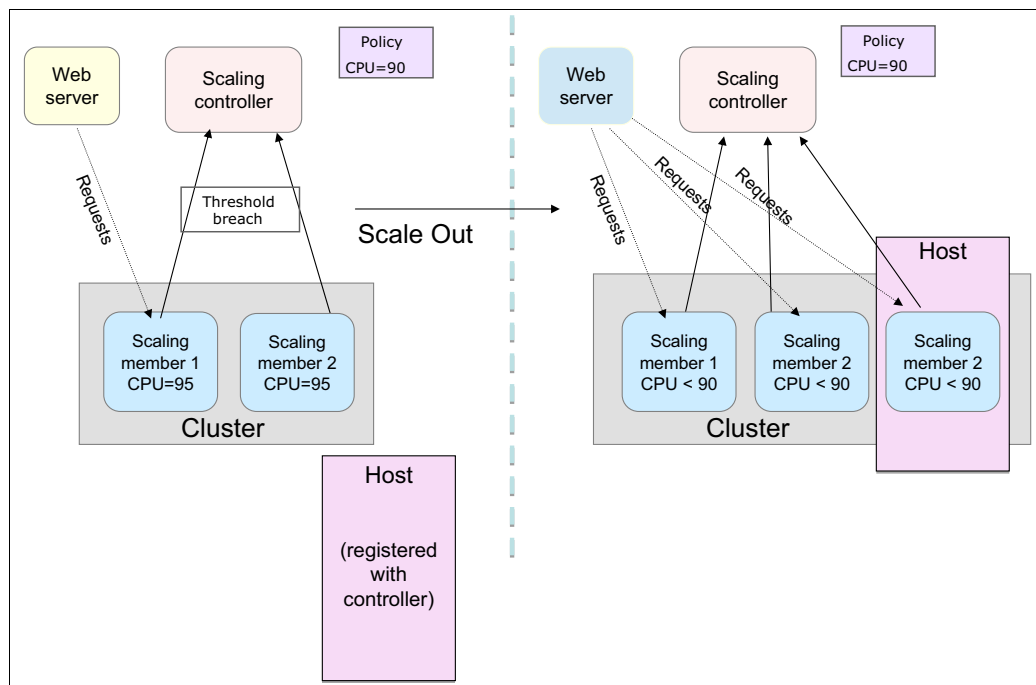


Figure 10-4 Scaling policies and Liberty elasticity

10.3.3 Configuring auto scaling for JVM elasticity

When you configure JVM elasticity, the scaling controller can start or stop Liberty servers based on resource use and scaling policies. There is no provisioning of new servers.

To configure auto scaling in your environment, use the following steps:

- ▶ Configure a controller, collective, replica set, and cluster
- ▶ Configure the scaling controller feature
- ▶ Configure the scaling member feature
- ▶ Define a policy
- ▶ Start the members, if needed, and enable more policies

For more details about configuring an auto scaling topology in your environment, use the following steps:

1. Configure a controller, collective, replica set, and cluster

Configure and start a controller and if needed, a replica set. Create a collective, add members, and create any needed clusters.

For details about configuring a controller, collective, replica set, and clusters, see Chapter 5, “Administering the WebSphere Liberty profile” on page 71.

2. Configure the scaling controller feature

Add the `scalingController-1.0` feature to the `server.xml` file on the controller, as shown in Example 10-7. Ensure that you update the `server.xml` on each controller that you have in your topology.

Example 10-7 Adding the scalingController-1.0 feature

```
<featureManager>
  <feature>collectiveController-1.0</feature>
  <feature>scalingController-1.0</feature>
</featureManager>

<scalingDefinitions>
  <defaultScalingPolicy enabled="false" min="2" max="4"/>
</scalingDefinitions>
```

Because Liberty configuration is dynamic, when you add the scaling controller feature, the default scaling policy takes effect and you might get unexpected results. For example, the default policy has a minimum of two servers. When you save the `server.xml` file, the scaling controller attempts to start two servers. For this reason, it is best to disable the default scaling policy while configuring auto scaling.

Example 10-8 shows an example of the messages that appear in the `messages.log` file for the controller if the scaling policy is enabled. If the scaling policy is `enabled=false`, the messages do not appear in the `messages.log` file.

Example 10-8 Details from the messages.log file

```
[ ] 00000032 com.ibm.ws.imf.apc.IMContainerGroupImpl I
CWWKV0405I: The scaling controller cannot meet the minimum instances for
cluster ITS0Cluster because too few scaling members are defined.
[ ] 00000138 com.ibm.ws.imf.apc.IMContainerGroupImpl I
CWWKV0404I: The scaling controller cannot increase the number of servers in
cluster ITS0Cluster because it cannot find a host with the capacity to add a
server.
```

After adding the scaling controller feature, examine the `messages.log` file to ensure that the feature is installed and activated. Look for the message that indicates the server is selected to be the primary scaling controller. See Example 10-9.

Example 10-9 Output from the `messages.log` file

```
[ ] 00000031 com.ibm.ws.scaling.controller.internal.ScalingControllerImpl I
CWWKV0100I: The ScalingController feature is activated.
[ ] 00000031 ws.collective.singleton.internal.SingletonServiceManagerImpl I
CWWKX1002I: Singleton service ScalingControllerSingletonService for scope
collective is created.
[ ] 00000031 om.ibm.ws.collective.singleton.internal.SingletonServiceImpl I
CWWKX1019I: The local singleton candidate is elected leader:
host=lexbz181072.lex.dst.ibm.com; userdir=/opt/IBM/WebSphere/Liberty/usr/;
server=controllerITS01; port=9449; service=ScalingControllerSingletonService;
scope=collective
[ ] 00000031 com.ibm.ws.scaling.controller.messaging.ControllerHaService I
CWWKV0102I: This server is elected to be the primary scaling controller.
[ ] 00000031 com.ibm.ws.scaling.manager.stack.internal.StackManagerImpl I
CWWKV0302I: The existing stacks are []
```

3. Configure the scaling member feature

To make a collective member a dynamic cluster member, add the `clusterMember-1.0` and `scalingMember-1.0` features to the `server.xml` file to all collective members that you want the scaling controller to control, as shown in Example 10-10.

Example 10-10 Adding the member features

```
<featureManager>
  <feature>collectiveMember-1.0</feature>
  <feature>clusterMember-1.0</feature>
  <feature>scalingMember-1.0</feature>
</featureManager>

<hostSingleton name="ScalingMemberSingletonService" port="30003" />
```

Each scaling member needs to define a `<hostSingleton>` element in the `server.xml`. The `<hostSingleton>` element needs to include a `hostSingleton` name, in this case `ScalingMemberSingletonService` and a port number, such as 30003. You can indicate any port number if the port number is unique on the host computer. All scaling members on the same host must use the same `hostSingleton` port. The port is used as a synchronization mechanism to elect the host leader.

If the server is not started when you add the features and the `hostSingleton` element, you must start it manually for the scaling controller to recognize the added features. Examine the `messages.log` file for the controller for a `CWWKV0121I` message. The message indicates that the controller recognizes the scaling member.

Only one scaling member per host communicates with the scaling controller. The first scaling member to connect to the `ScalingMemberSingletonService` is elected as the host leader. If the host leader stops, another scaling member takes over as the host leader by an election process that is arbitrated by the `ScalingMemberSingletonService`.

Examine the `messages.log` file on the scaling member to verify that the features are activated. You should see that the features are activated, the `ScalingMemberSingletonService` is elected the host leader, and the messenger connection goes to the scaling controller.

Example 10-11 shows the details from the messages.log file.

Example 10-11 Details from the messages.log file

```
[ ] 00000020 ws.collective.singleton.internal.SingletonServiceImpl I
CWWKX1002I: Singleton service ScalingMemberSingletonService for scope host is
created.
[ ] 00000021 om.ibm.ws.collective.singleton.internal.SingletonServiceImpl I
CWWKX1019I: The local singleton candidate is elected leader:
host=lexbz181072.lex.dst.ibm.com; userdir=/opt/IBM/WebSphere/Liberty/usr/;
server=serverITS01; port=9448; service=ScalingMemberSingletonService;
scope=host
[ ] 00000021 com.ibm.ws.scaling.member.internal.ScalingMemberImpl I
CWWKV0203I: Server host=lexbz181072.lex.dst.ibm.com;
userdir=/opt/IBM/WebSphere/Liberty/usr/; server=serverITS01; port=9448;
service=ScalingMemberSingletonService; scope=host is elected as the host
leader.
[ ] 00000020 com.ibm.ws.scaling.member.internal.ScalingMemberImpl I
CWWKV0200I: The ScalingMember feature is activated.
```

If you are using the Admin Center, you can examine the status of the scaling member. Figure 10-5 shows the details for serverITS01 and that the server is auto scaled as noted by the identifier Auto scaling policy. The server is also a member of the cluster ITSOCler.

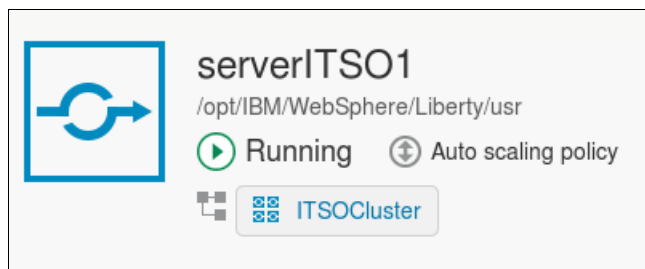


Figure 10-5 Auto scaling policy visual identifier in the Admin Center

4. Define a policy

In this example, all collective members are stopped before the policy is created. However, it is not a requirement to stop the collective members before you create a policy.

Example 10-12 shows an updated defaultScaling policy in the server.xml file on the controller. Ensure that you update the server.xml on each controller that you have in your topology.

Example 10-12 Updating the defaultScalingPolicy

```
<scalingDefinitions>
  <defaultScalingPolicy enabled="true" min="2" max="4"/>
</scalingDefinitions>
```

5. Start the members, if needed, and enable more policies

In the scenario used in this section, the defaultScalingPolicy was modified to enabled=true. This causes the scaling controller to immediately evaluate the policy. Because the policy indicates a minimum of two servers started, the scaling controller starts the scaling members serverISTO1 and serverISTO2. Example 10-13 on page 187 shows the details from the messages.log file.

Example 10-13 Output from the messages.log file

```
[ ] 0000051f com.ibm.ws.scaling.controller.internal.ScalingExecutorImpl I
CWVKV0112I: The scaling controller has successfully started server serverITS02
on host lexbz181072.lex.dst.ibm.com.
[ ] 0000053c ctive.repository.internal.metadata.AdminMetadataEventHandler I
CWWKX9068I: Administrative metadata for resource
lexbz181072.lex.dst.ibm.com,%2Fopt%2FIBM%2FWebSphere%2FLiberty%2Fusr,serverITS0
2,Sample1 was removed from the collective repository.
[ ] 0000053c ctive.repository.internal.metadata.AdminMetadataEventHandler I
CWWKX9068I: Administrative metadata for resource
lexbz181072.lex.dst.ibm.com,%2Fopt%2FIBM%2FWebSphere%2FLiberty%2Fusr,serverITS0
2,snoop was removed from the collective repository.
[ ] 00000523 com.ibm.ws.scaling.controller.internal.ScalingExecutorImpl I
CWVKV0112I: The scaling controller has successfully started server serverITS01
on host lexbz181072.lex.dst.ibm.com.
```

Next, if needed, you can create or modify scaling policies.

10.3.4 Configuring auto scaling for Liberty elasticity

When you configure Liberty elasticity, the scaling controller can provision new servers and start or stop Liberty servers based on resource use and scaling policies. When provisioning a new server, the new server is auto-deployed on available hosts that are registered with the collective controller.

With Liberty elasticity, it can deploy the Liberty runtime, Java runtime environment (JRE), and Liberty server packages.

To configure auto scaling for Liberty elasticity in your environment, use the following steps:

- ▶ Configure a collective to support JVM elasticity
- ▶ Configure the dynamic cluster members
- ▶ Create packages for deploying to new hosts
- ▶ Provide packages to the collective controller to deploy onto a host
- ▶ Register each target host with a scaling controller
- ▶ Configure a scaling policy

For more information about configuring auto scaling for Liberty elasticity, see the following web page:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/twlp_autoscale_configlibertyelast.html?lang=en

10.4 Maintenance mode

Periodic product maintenance is important to keep your system environment working correctly, and to avoid trouble caused by known issues. At some point, you might have a problem with a host or server and need to run diagnostic tests to troubleshoot a specific host or server. These situations can lead to the disruption of client requests to servers in your environment.

You can administratively put a host or a server into maintenance mode. In fact, you should set maintenance mode before you even begin to perform any diagnostic tests, maintenance, or tuning on a host or server.

You can put a server into maintenance mode when you need to perform server-level problem determination. When a server is placed into maintenance mode, you stop routing HTTP requests to the server. By default, the web server continues to send requests with affinity to a server in maintenance mode. However, you can break affinity to a server and cause requests with affinity to be routed to other servers. When a server that is a scaling member is in maintenance mode, the scaling controller cannot start or stop that server. Also, the server does not count toward the minimum or maximum running instance settings for the dynamic cluster.

When a host is placed into maintenance mode, it places all of the servers on the host into maintenance mode. The host maintenance mode allows you to put a host into maintenance mode when you need to apply operating system fixes or other fixes. When a host is in maintenance mode, the scaling controller cannot provision a new server on that host.

You can configure maintenance mode either by using the command line or by using the Admin Center. Placing a server into maintenance mode is a persistent change. The host or server remains in maintenance mode until you disable it.

10.4.1 Configuring maintenance by using the command line

When configuring maintenance mode by using the command line, there are three actions on the collective command that you can use:

- ▶ `enterMaintenanceMode`
- ▶ `exitMaintenanceMode`
- ▶ `getMaintenanceMode`

Example 10-14 shows the syntax for the `enterMaintenanceMode` command.

Example 10-14 Command syntax for `enterMaintenanceMode`

```
$ ./collective enterMaintenanceMode
  --host=controllerHostName
  --port=controllerHttpsPortNumber
  --user=adminUser
  --password=adminPassword
  --hostName=serverHostName
  [--usrDir=serverUserDirectory]
  [--server=serverName]
  [--break]
  [--force]
  [--autoAcceptCertificates]
```

The `--host` and `--port` arguments identify the collective controller that can process the command. The `--user` and `--password` arguments are the administrative user ID and password for authenticating with the controller. If you do not provide the password value on the command line, you are prompted to enter it when running the command. All of these arguments are required.

The `--hostName`, `--usrDir`, and `--server` arguments identify the server to be put into maintenance mode. If you want to put a host and its servers into maintenance mode, provide the `--hostName` argument only and omit the `--usrDir` and `--server` arguments.

Note: All the examples in this section use the fully qualified domain name for the host name. In a production environment, you should always use the fully qualified domain name.

A host can be registered with the collective under different names. It is important that the host name specified for the collective registerHost, updateHost, and unregisterHost be consistent with the host name used for the registered collective members. The defaultHostName attribute in the server.xml for the member controls the host name to which the server considers itself to belong. All collective commands must use the same host name as that used in the defaultHostName attribute in the server.xml file.

Configuring maintenance mode for a server

The server serverITS01 on the host lexbz181072.lex.dst.ibm.com is put into maintenance mode as shown in Example 10-15. Because the password argument is empty, you are prompted to enter the password. You are also prompted to accept the certificate.

Example 10-15 Output of enterMaintenanceMode for serverITS01

```
$ ./collective enterMaintenanceMode --host=lexbz181072.lex.dst.ibm.com --password
--port=9449 --user=liberty --hostName=lexbz181072.lex.dst.ibm.com --password
--usrDir=/opt/IBM/WebSphere/Liberty/usr --server=serverITS01
Enter password --password:
```

SSL trust has not been established with the target server.

Certificate chain information:

Certificate [0]

Subject DN: CN=lexbz181072.lex.dst.ibm.com, OU=controllerITS0, O=ibm, C=us
Issuer DN: OU=controllerRoot, O=88f53048-e416-4401-9bf5-fc23ab68fa84,
DC=com.ibm.ws.collective
Serial Number: 958,970,498,256,025
Expires: 8/23/20 4:42 PM
SHA-1 digest: AF:72:AF:3F:A1:0C:3A:DF:C1:42:54:3F:37:30:78:96:60:11:1D:15
MD5 digest: A0:85:04:D9:62:11:73:39:FA:D5:B7:2F:3F:05:30:D2

Certificate [1]

Subject DN: OU=controllerRoot, O=88f53048-e416-4401-9bf5-fc23ab68fa84,
DC=com.ibm.ws.collective
Issuer DN: OU=controllerRoot, O=88f53048-e416-4401-9bf5-fc23ab68fa84,
DC=com.ibm.ws.collective
Serial Number: 958,959,416,359,026
Expires: 8/18/40 4:42 PM
SHA-1 digest: 4F:2F:26:D7:7B:61:88:96:D7:FB:5F:AD:0D:5E:CF:96:AC:95:CD:68
MD5 digest: 18:5F:55:F5:29:59:46:DC:35:88:55:23:CD:CB:94:BA

Do you want to accept the above certificate chain? (y/n) y

Successfully set maintenance mode for serverITS01.

To see the details from the command, examine the messages.log file for the controller. In this example, controllerITS01. Example 10-16 on page 190 shows the messages in the messages.log file regarding the **enterMaintenanceMode** command for serverITS01.

The option to maintain session affinity to server is set to true, which is the default setting. This means that the web server continues to send requests with affinity to this server that is now in maintenance mode.

Example 10-16 Details from the messages.log file

```
[ ] 0000043c .ibm.ws.collective.command.internal.MaintenanceModeMBeanImpl I
CWWKX7225I: The collective controller is processing a request to place server
serverITS01 in user directory /opt/IBM/WebSphere/Liberty/usr on host
lexbz181072.lex.dst.ibm.com into maintenance mode. The option to maintain session
affinity is set to true. The option to bypass autoScaling violations is set to
false.
[ ] 0000043c .ibm.ws.collective.command.internal.MaintenanceModeMBeanImpl I
CWWKX7228I: Server serverITS01 in user directory /opt/IBM/WebSphere/Liberty/usr on
host lexbz181072.lex.dst.ibm.com has been placed into maintenance mode.
```

In Example 10-17, the `enterMaintenanceMode` command is used to put serverITS02 into maintenance mode. This time, the argument `--break` is used, which causes requests with affinity to be routed to other servers.

Example 10-17 Output of enterMaintenanceMode with the break option

```
$ ./collective enterMaintenanceMode --host=lexbz181072.lex.dst.ibm.com --password
--port=9449 --user=liberty --hostName=lexbz181072.lex.dst.ibm.com --password
--usrDir=/opt/IBM/WebSphere/Liberty/usr --server=serverITS02 --break
Enter password --password:
```

SSL trust has not been established with the target server.

Certificate chain information:

Certificate [0]

```
Subject DN: CN=lexbz181072.lex.dst.ibm.com, OU=controllerITS0, O=ibm, C=us
Issuer DN: OU=controllerRoot, O=88f53048-e416-4401-9bf5-fc23ab68fa84,
DC=com.ibm.ws.collective
Serial Number: 958,970,498,256,025
Expires: 8/23/20 4:42 PM
SHA-1 digest: AF:72:AF:3F:A1:0C:3A:DF:C1:42:54:3F:37:30:78:96:60:11:1D:15
MD5 digest: A0:85:04:D9:62:11:73:39:FA:D5:B7:2F:3F:05:30:D2
```

Certificate [1]

```
Subject DN: OU=controllerRoot, O=88f53048-e416-4401-9bf5-fc23ab68fa84,
DC=com.ibm.ws.collective
Issuer DN: OU=controllerRoot, O=88f53048-e416-4401-9bf5-fc23ab68fa84,
DC=com.ibm.ws.collective
Serial Number: 958,959,416,359,026
Expires: 8/18/40 4:42 PM
SHA-1 digest: 4F:2F:26:D7:7B:61:88:96:D7:FB:5F:AD:0D:5E:CF:96:AC:95:CD:68
MD5 digest: 18:5F:55:F5:29:59:46:DC:35:88:55:23:CD:CB:94:BA
```

Do you want to accept the above certificate chain? (y/n) y

Successfully set maintenance mode for serverITS02.

Example 10-18 on page 191 shows the messages in the `messages.log` file regarding the `enterMaintenanceMode` command for serverITS02 in the `messages.log` file where the `--break` argument is used, setting the value to maintain session affinity to false.

Example 10-18 Details from the messages.log file

```
[ ] 000004da .ibm.ws.collective.command.internal.MaintenanceModeMBeanImpl I
CWWKX7225I: The collective controller is processing a request to place server
serverITS02 in user directory /opt/IBM/WebSphere/Liberty/usr on host
lexbz181072.lex.dst.ibm.com into maintenance mode. The option to maintain session
affinity is set to false. The option to bypass autoScaling violations is set to
false.
[ ] 000004da .ibm.ws.collective.command.internal.MaintenanceModeMBeanImpl I
CWWKX7228I: Server serverITS02 in user directory /opt/IBM/WebSphere/Liberty/usr on
host lexbz181072.lex.dst.ibm.com has been placed into maintenance mode.
```

To determine the maintenance status of a specific server, use the **getMaintenanceMode** command as shown in Example 10-19. The command arguments are the same for this command. The argument **--autoAcceptCertificates** is used, which means to automatically trust Secure Sockets Layer (SSL) certificates during this command.

You can see that serverITS01 is in maintenance mode. No details about the **getMaintenanceMode** command appear in the messages.log on the collective controller.

Example 10-19 Syntax and output of the getMaintenanceMode command

```
$ ./collective getMaintenanceMode --host=lexbz181072.lex.dst.ibm.com --password
--port=9449 --user=liberty --hostName=lexbz181072.lex.dst.ibm.com
--server=serverITS01 --usrDir=/opt/IBM/WebSphere/Liberty/usr
--autoAcceptCertificates
Enter password --password:
```

Auto-accepting the certificate chain for target server.
Certificate subject DN: CN=lexbz181072.lex.dst.ibm.com, OU=controllerITS0, O=ibm,
C=us

serverITS01 is in maintenance mode.

To exit maintenance mode for a server, use the **exitMaintenanceMode** command. Example 10-20 shows an example of taking serverITS01 out of maintenanceMode.

Example 10-20 Syntax and output of exitMaintenanceMode command

```
$ ./collective exitMaintenanceMode --host=lexbz181072.lex.dst.ibm.com --password
--port=9449 --user=liberty --hostName=lexbz181072.lex.dst.ibm.com
--server=serverITS01 --usrDir=/opt/IBM/WebSphere/Liberty/usr
--autoAcceptCertificates
Enter password --password:
```

Auto-accepting the certificate chain for target server.
Certificate subject DN: CN=lexbz181072.lex.dst.ibm.com, OU=controllerITS0, O=ibm,
C=us

Successfully unset maintenance mode for serverITS01.

Example 10-21 shows the output of the `exitMaintenanceMode` command for the serverITS01 in the `messages.log` file.

Example 10-21 Output in the messages.log file

```
[ ] 00000475 .ibm.ws.collective.command.internal.MaintenanceModeMBeanImpl I
CWWKX7230I: The collective controller is processing a request to take server
serverITS01 in user directory /opt/IBM/WebSphere/Liberty/usr on host
lexbz181072.lex.dst.ibm.com out of maintenance mode.
[ ] 00000475 .ibm.ws.collective.command.internal.MaintenanceModeMBeanImpl I
CWWKX7232I: Server serverITS01 in user directory /opt/IBM/WebSphere/Liberty/usr on
host lexbz181072.lex.dst.ibm.com has been taken out of maintenance mode.
```

Configuring maintenance mode for a host

Example 10-22 shows the command to put the host `lexbz181072.lex.dst.ibm.com` into maintenance mode. In this example, the host and all servers on the host are put into maintenance mode.

Example 10-22 Putting a host into maintenance mode

```
$ ./collective enterMaintenanceMode --host=lexbz181072.lex.dst.ibm.com --password
--port=9449 --user=liberty --hostName=lexbz181072.lex.dst.ibm.com --password
Enter password --password:
```

```
SSL trust has not been established with the target server.
```

```
Certificate chain information:
```

```
Certificate [0]
```

```
Subject DN: CN=lexbz181072.lex.dst.ibm.com, OU=controllerITS0, O=ibm, C=us
Issuer DN: OU=controllerRoot, O=88f53048-e416-4401-9bf5-fc23ab68fa84,
DC=com.ibm.ws.collective
Serial Number: 958,970,498,256,025
Expires: 8/23/20 4:42 PM
SHA-1 digest: AF:72:AF:3F:A1:0C:3A:DF:C1:42:54:3F:37:30:78:96:60:11:1D:15
MD5 digest: A0:85:04:D9:62:11:73:39:FA:D5:B7:2F:3F:05:30:D2
```

```
Certificate [1]
```

```
Subject DN: OU=controllerRoot, O=88f53048-e416-4401-9bf5-fc23ab68fa84,
DC=com.ibm.ws.collective
Issuer DN: OU=controllerRoot, O=88f53048-e416-4401-9bf5-fc23ab68fa84,
DC=com.ibm.ws.collective
Serial Number: 958,959,416,359,026
Expires: 8/18/40 4:42 PM
SHA-1 digest: 4F:2F:26:D7:7B:61:88:96:D7:FB:5F:AD:0D:5E:CF:96:AC:95:CD:68
MD5 digest: 18:5F:55:F5:29:59:46:DC:35:88:55:23:CD:CB:94:BA
```

```
Do you want to accept the above certificate chain? (y/n) y
```

```
Successfully set maintenance mode for lexbz181072.lex.dst.ibm.com.
```

```
Successfully set maintenance mode for serverITS01.
```

```
Successfully set maintenance mode for serverITS02.
```

Example 10-23 on page 193 shows the output of the `enterMaintenanceMode` command for the host `lexbz181072.lex.dst.ibm.com` in the `messages.log` file.

Example 10-23 Output in the messages.log file

```
[ ] 00000485 .ibm.ws.collective.command.internal.MaintenanceModeMBeanImpl I
CWWKX7224I: The collective controller is processing a request to place host
lexbz181072.lex.dst.ibm.com into maintenance mode. The option to maintain session
affinity is set to true. The option to bypass autoScaling violations is set to
false.
[ ] 00000485 .ibm.ws.collective.command.internal.MaintenanceModeMBeanImpl I
CWWKX7227I: Host lexbz181072.lex.dst.ibm.com has been placed into maintenance
mode.
[ ] 00000485 .ibm.ws.collective.command.internal.MaintenanceModeMBeanImpl I
CWWKX7228I: Server serverITS01 in user directory /opt/IBM/WebSphere/Liberty/usr on
host lexbz181072.lex.dst.ibm.com has been placed into maintenance mode.
[ ] 00000485 .ibm.ws.collective.command.internal.MaintenanceModeMBeanImpl I
CWWKX7228I: Server serverITS02 in user directory /opt/IBM/WebSphere/Liberty/usr on
host lexbz181072.lex.dst.ibm.com has been placed into maintenance mode.
```

10.4.2 Configuring maintenance by using the Admin Center

To configure maintenance by using the Admin Center, complete the following steps.

1. Open the Admin Center and log in.
2. In the toolbox, open the Explore tool.
3. From the view, you can select either Servers or Hosts to configure. For example, to configure maintenance mode for a server, click **Servers**.

For example, in Figure 10-6 there are two servers, serverITS01 and serverITS02. To configure server maintenance mode for serverITS01, click the **Actions** icon → **Enable Maintenance Mode**.

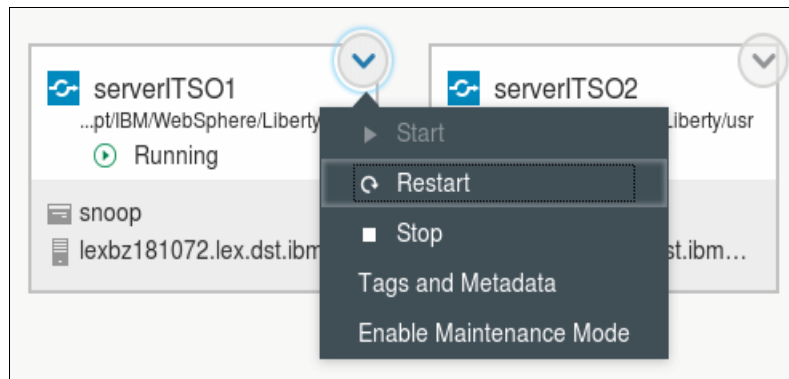


Figure 10-6 Enable server maintenance mode

In Figure 10-7 on page 194, you can see the “Enable maintenance mode” prompt. From here, you can choose the toggle option to “Break affinity with active sessions.” Otherwise, the default is for the web server to continue to send requests to the server. Finally, click **Enable**.

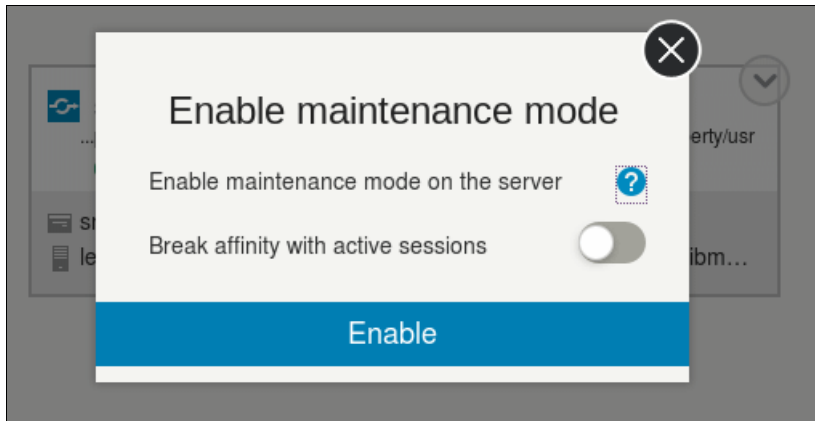


Figure 10-7 Enable maintenance mode prompt

In Figure 10-8, you can see that maintenance mode is enabled for serverITSO1 as indicated by an orange box around the server. The server, serverITSO2, does not have maintenance mode enabled.

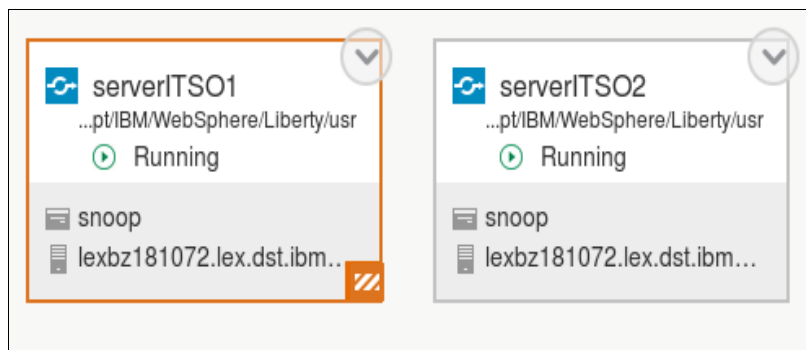


Figure 10-8 Maintenance mode for serverITSO1

To disable maintenance mode for a server, click the **Actions** icon → **Disable Maintenance Mode** as displayed Figure 10-9.

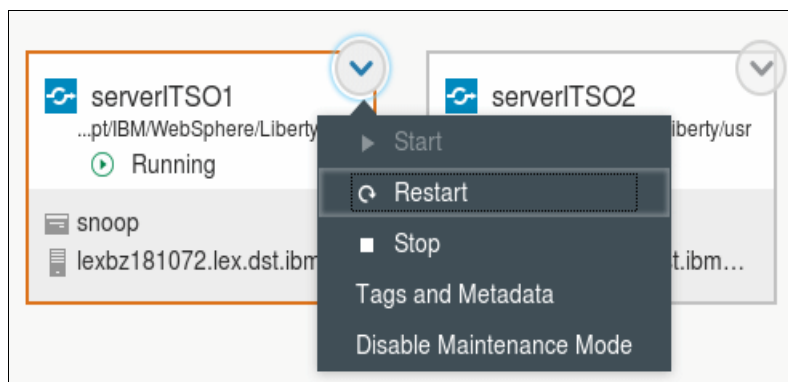


Figure 10-9 Disable maintenance mode

10.5 Health management

Health management allows you to monitor the status of your servers. The health management feature was added to WAS Classic in V8.5. In V8.5.5.7, health management functionality was added to Liberty.

You can use health management to detect and respond to problem areas before an outage occurs in your environment. You can manage the health of an environment with a policy-driven approach that enables specific actions to occur when monitored criteria are met. For example, when memory usage exceeds a percentage of the heap size for a specified time, actions can run to recover from the situation.

Health monitoring can help you with both unexpected issues and unanticipated problems in your environment. It can help you recover from problems that would otherwise disrupt operations and affect performance.

Health management does not require the use of auto scaling or dynamic routing. However, if you want to gain the full functionality of the health management feature, it is recommended that both the auto scaling and dynamic routing features be enabled.

The health management feature consists of:

- ▶ Health policies
- ▶ Health management controller

10.5.1 Health policies

With health management, you can define health policies. A health policy provides a health standard for the Liberty environment. Health policies are designed to identify potential problems, and take recovery actions when a particular event occurs. You can define health policies for common conditions in Liberty.

Each health policy consists of a condition, one or more targets, and one or more actions. A condition indicates what you want to monitor in your Liberty environment. The action defines what happens when you encounter that condition. When defining a health policy, you also define a health target, which defines where you want to monitor the condition and where you want the action to take place. Health targets are identified as a host, server, or cluster.

A key advantage of health management is that when a health policy violation is detected, an action plan can be put into effect automatically without requiring administrative interaction. Actions to be taken when a monitored condition is detected are designed to recover from the problem and help in diagnosis.

Health conditions

Health conditions define the variables that you want to monitor in your environment. The health conditions that you can define in Liberty include:

- ▶ Excessive memory usage
 - Triggers when the members associated with this detection-based policy use more memory than a percentage of the maximum heap size for a certain amount of time.
- ▶ Memory leak
 - Looks for consistent downward trends in free memory that is available to a server in the Java heap.

- ▶ Excessive average response time
Triggers when the members that are associated with this detection-based policy have an average response time for requests that exceed a certain amount of time.
- ▶ Excessive request timeout
Triggers when HTTP requests that are directed to an associated member timeout, and the percentage of timeouts exceed the specified value.

You can configure only one health condition per health policy.

There are a few key points to keep in mind regarding health management. In Liberty, the dynamic routing feature in Intelligent Management is not required. However, if you want to monitor the excessive response time or excessive request timeout conditions by defined health policies, you must enable dynamic routing. The excessive response time or excessive request timeout conditions also require the IBM HTTP Server plug-in configuration and routing requests to your environment.

If you want to monitor excessive memory usage and memory leak conditions, you must configure the Health Analyzer and enable the JVM option `-Xhealthcenter:level=inprocess` on each collective member where you want to monitor the conditions.

Health actions

Health actions define the action to take when defined conditions are met. The health actions that you can define in Liberty include:

- ▶ Capture diagnostics such as a heap or thread dump
Taking a JVM heap dump is supported for servers that are running on the IBM software development kit (SDK).
- ▶ Restart the server
- ▶ Put a server into or take a server out of maintenance mode
This action takes advantage of the server maintenance mode feature that is also part of the Intelligent Management features in Liberty.

10.5.2 Health management controller

The *health management controller* is an autonomic manager that constantly monitors the defined health policies. When a condition that is specified by a health policy is met in the environment, the health management controller executes the configured actions.

10.5.3 Configuring health management features for Liberty

To use the health management features for Liberty, you must first configure the collective feature. Each server where you want to define health policies must be a member of a collective. Then, to configure health management for Liberty, you simply modify the `server.xml` file.

A new element type of `<healthPolicy>` is used to define a single health policy. If more than one policy is needed, you need to add multiple `<health policy>` elements.

Configuring health conditions

To configure the health conditions, add one of the following elements to the <healthPolicy> element in your server.xml file on the collective controller:

- ▶ Configure the excessive timeout condition as described in Example 10-24. This condition specifies the percentage of HTTP requests that can time out. When the percentage of requests exceeds the defined value, the health actions run.

Example 10-24 Configuring the excessive timeout condition

```
<excessiveRequestTimeout timeoutPercentage="5"/>
```

- ▶ Configure the excessive response time condition as described in Example 10-25. For this condition, if the time exceeds the defined response time threshold, the health actions run.

Example 10-25 Configuring the excessive response time condition

```
<excessiveResponseTime responseTime="10s"/>
```

- ▶ Configure the excessive memory usage condition as described in Example 10-26. For this condition, when the memory usage exceeds the defined percentage of the heap size for the specified time, health actions run.

Example 10-26 Configuring the excessive memory usage condition

```
<excessiveMemoryUsage heapSizePercentage="85" timePeriod="5m"/>
```

- ▶ Configure the memory leak condition as described in Example 10-27. For this condition, when a downward trend in free memory is detected, health actions run.

Example 10-27 Configuring the memory leak condition

```
<memoryLeak/>
```

Configuring health actions

To configure a health action, add any of the following to the <healthPolicy> element in your server.xml file on the collective controller.

Configure health actions as described in Example 10-28.

Example 10-28 Configuring health actions

```
<action action="generateServerDump"/>  
<action action="generateHeapDump"/>  
<action action="restartServer"/>  
<action action="setMaintenanceMode"/>  
<action action="unsetMaintenanceMode"/>
```

For each health policy where you indicate the condition, you can have multiple actions. You can define a single action or indicate multiple actions. The actions are executed in the order that they are defined within the policy.

Configuring health targets

To configure a health target, add the following to the <healthPolicy> element in your server.xml file on the collective controller:

- ▶ Configure a cluster health target as described in Example 10-29.

Example 10-29 Configuring the cluster health target

```
<cluster clusterName="MyClusterName"/>
```

- ▶ Configure a server health target as described in Example 10-30.

Example 10-30 Configuring the server health target

```
<server hostName="MyHostName" wlpusrDirectory=/opt/IBM/WebSphere/Liberty  
serverName="MyServerName"/>
```

- ▶ Configure a host health target as described in Example 10-31.

Example 10-31 Configuring the host health target

```
<host hostName="MyHostName"/>
```

For each health policy where you indicate the condition, you can have multiple actions and either a single or multiple targets. All servers identified by all targets listed are monitored on an individual server basis for the condition of the policy.

Configuring health management for your Liberty environment

The health management functionality is enabled by two Liberty features: health manager and health analyzer. The health manager feature is added to the collective controller configuration and the health analyzer is added to the collective member configuration when needed. If memory conditions are not used, it is not needed.

To configure health management in your environment, use the following steps:

- ▶ Configure a Liberty collective
- ▶ Configure dynamic routing and auto scaling
- ▶ Configure the JVM option for memory conditions
- ▶ Configure the health manager feature
- ▶ Configure the health analyzer feature
- ▶ Define your health policies in the server.xml file on your collective controller

For more information about configuring health management in your environment, use the following steps:

1. Configure a Liberty collective.

For more information about how to create a collective and collective controller, see Chapter 5, “Administering the WebSphere Liberty profile” on page 71.

2. Configure dynamic routing and auto scaling

If you want to gain the full functionality of the health management feature, it is recommended that both the dynamic routing and auto scaling features be enabled. For details about configuring dynamic routing and auto scaling, refer to the sections on each of these topics earlier in the chapter.

3. Configure the JVM option for memory conditions

If memory leak or excessive memory usage conditions are being monitored in a collective member, it is necessary to enable a JVM option for the Health Center before configuring health management. To configure, add the JVM option `-Xhealthcenter:level=inprocess` to the collective member.

4. Configure the health manager feature.

Add the `healthManager-1.0` feature to the `server.xml` file on your collective controller or controllers as described in Example 10-32.

Example 10-32 Configuring the health management feature on the collective controller

```
<featureManager>
  <feature>collectiveController-1.0</feature>
  <feature>dynamicRouting-1.0</feature>
  <feature>scalingController-1.0</feature>
  <feature>healthManager-1.0</feature>
</featureManager>
```

5. Configure the health analyzer feature

Add the `healthAnalyzer-1.0` feature to the `server.xml` file for each collective member, as described in Example 10-33.

Example 10-33 Configuring the healthAnalyzer feature on the collective members

```
<featureManager>
  <feature>collectiveMember-1.0</feature>
  <feature>clusterMember-1.0</feature>
  <feature>scalingMember-1.0</feature>
  <feature>healthAnalyzer-1.0</feature>
</featureManager>
```

6. If you are going to define health policies that target a cluster, you must also configure the collective member to be a part of the cluster. To configure a server cluster, see Chapter 5, “Administering the WebSphere Liberty profile” on page 71.

7. Define your health policies in the `server.xml` file on your collective controller

The example described in Example 10-34 is a basic example where `HealthPolicy1` is defined with a target of all servers on host `lexbz181072.lex.dst.ibm.com`. `HealthPolicy1` monitors the `excessiveMemoryUsage` condition based on the defined thresholds. When the memory exceeds 55% of the heap for 5 minutes, the server is put into maintenance mode.

Example 10-34 Configuring a health policy

```
<healthPolicy id="HealthPolicy1" enable=true">
  <host hostName="lexbz181072.lex.dst.ibm.com" />
  <excessiveMemoryUsage heapSizePercentage="55" timePeriod="5m" />
  <action action="enterMaintenanceMode" />
</healthPolicy>
```

8. Examine the `messages.log` file on the collective controller to observe the details of the health policies.

Example 10-35 shows the output in the `monitor.log` file on the collective controller when the policy is added. You can see that the policy `HealthPolicy1` is activated.

Example 10-35 Output in the monitor.log file of the addition of a health policy

```
CWWKV0609I: The health policy HealthPolicy1 is added.
CWWKV0603I: The health condition
com.ibm.ws.health.manager.healthPolicy.condition.excessiveMemoryUsage is being
monitored on target lexbz181072.lex.dst.ibm.com.
CWWKV0607I: The health policy HealthPolicy1 is activated for serverITS01.
CWWKV0607I: The health policy HealthPolicy1 is activated for serverITS02.
```

9. Configure multiple health policies if needed.

Example 10-36 shows multiple health policies defined in the file.

Example 10-36 Configuring multiple health policies

```
<healthPolicy id="HealthPolicy1" enable=true">
  <host hostName=lexbz18.lex.dst.ibm.com" />
  <excessiveMemoryUsage heapSizePercentage="55" timePeriod="5m" />
  <action action="enterMaintenanceMode"/>
</healthPolicy>

healthPolicy id="HealthPolicy2" enable=true">
  <host hostName="lexbz19.lex.dst.ibm.com" />
  <host hostName=lexbz20.lex.dst.ibm.com" />
  <server hostName="lexbz21.lex.dst.ibm.com"
wlpUsrDirectory="/opt/IBM/WebSphere/liberty/wlp" serverName="serverITS03" />
  <memoryLeak />
  <action action="enterMaintenanceMode" />
  <action action="generateThreadDump" />
  <action action="generateHeapDump" />
  <action action="restartServer" />
  <action action="exitMaintenanceMode" />
</healthPolicy>

<healthPolicy id="HealthPolicy3" enable=true">
  <cluster ClusterName=ITS0Cluster" />
  <excessiveRequestTimeout timeoutPercentage=5" />
  <action action="enterMaintenanceMode" />
</healthPolicy>
```

Again, `HealthPolicy1` is defined with a target of all servers on host `lexbz18.lex.dst.ibm.com`. `HealthPolicy1` monitors the `excessiveMemoryUsage` condition based on the defined thresholds. When the memory exceeds 55% of the heap for 5 minutes, the action to take is to put the server into maintenance mode.

`HealthPolicy2` is defined with a target of all servers on host `lexbz19.lex.dst.ibm.com` and `lexbz20.lex.dst.ibm.com` and server `serverITS03` on `lexbz21.lex.dst.ibm.com`. `HealthPolicy2` monitors the memory leak condition. When a memory leak is detected, the affected health target is placed into maintenance mode, a thread dump is generated, a heap dump is generated, the server is restarted, and then maintenance mode is removed.

HealthPolicy3 is defined with a target the cluster ITSOCcluster. HealthPolicy3 monitors the excessiveRequestTime condition based on the defined thresholds of 5%. When 5% of the HTTP requests to the cluster time-out, the action to take is to place the server into maintenance mode. The excessiveRequestTime condition requires the dynamic routing feature, so the server.xml file must also include the dynamicRouting-1.0 feature.

If you have a health target defined for a cluster, you must also configure the collective members to be part of a server cluster. The server.xml file of the collective member in the cluster must include the clusterMember-1.0 feature. For more information about how to create a cluster, see Chapter 5, “Administering the WebSphere Liberty profile” on page 71.

If you want to use both the health management memory conditions and auto scaling features in your environment, you must also define the hostSingleton element. Each scaling member needs to define a hostSingleton element with a port in the server.xml file. All scaling members on the same host must use the same port. You can specify any port number, but the port number must be unique on the host computer.

Example 10-37 shows the details in the server.xml file for a collective member that is part of a cluster and is using both health management and auto scaling features in your environment. The hostSingleton element is also defined indicating the unique name and port numbers. The hostSingleton specification must have a unique port that is separately set for each service.

Example 10-37 Configuring the hostSingleton element

```
<featureManager>
  <feature>collectiveMember-1.0</feature>
  <feature>clusterMember-1.0</feature>
  <feature>scalingMember-1.0</feature>
  <feature>healthAnalyzer-1.0</feature>
</featureManager>

<hostSingleton name="ScalingMemberSingletonService" port="20020"/>
<hostSingleton name="HealthAnalyzerSingletonService" port="20021"/>
```

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only:

- ▶ *WebSphere Application Server: New Features in V8.5.5*, REDP-4870
- ▶ *WebSphere Application Server V8.5.5 Technical Overview*, REDP-4855
- ▶ *IBM WebSphere Application Server V8.5 Concepts, Planning, and Design Guide*, SG24-8022
- ▶ *WebSphere Application Server Liberty Profile Guide for Developers*, SG24-8076

You can search for, view, download, or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- ▶ WAS Liberty:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/cwlp_about.html

- ▶ Tools editions:

<http://www.ibm.com/software/webservers/appserv/was/tools>

- ▶ Rational Application Developer for WebSphere Software V9:

<http://www.ibm.com/software/awdtools/developer/application>

- ▶ List of the supported Java EE 6 and Java EE 7 feature combinations:

https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/rwlp_prog_model_supported_combos.html

- ▶ Features for network deployment:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_feat.html?cp=SSAW57_8.5.5%2F3-0-2-3-0

- ▶ Using the **server** command:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_command_server.html?cp=SSAW57_8.5.5%2F3-3-11-0-3-2-1-0

- ▶ Using the **securityUtility** command:
http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_command_securityutil.html?cp=SSAW57_8.5.5%2F3-3-11-0-4-1-2-0
- ▶ Using the **featureManager** command:
http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_command_featuremanager.html?cp=SSAW57_8.5.5%2F3-3-11-0-1-2-2-0
- ▶ Using the **productInfo** command:
http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_command_productinfo.html?cp=SSAW57_8.5.5%2F3-3-11-0-1-3-0
- ▶ Customizing the Liberty profile environment:
http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_admin_customvars.html?cp=SSAW57_8.5.5%2F3-3-11-0-3-2-0
- ▶ Liberty profile files and directories:
http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_dirs.html?cp=SSAW57_8.5.5%2F3-3-11-0-2-0
- ▶ Optional files:
http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_admin_customvars.html?cp=SSAW57_8.5.5%2F3-3-11-0-3-2-0
- ▶ Configuring security for your applications on the Liberty profile:
<https://www.ibmdev.net/wasdev/category/repo/config-snippets>
- ▶ Liberty profile package on WASdev community website:
<https://www.ibm.com/developerworks/mydeveloperworks/blogs/wasdev/entry/download?lang=en>
- ▶ Information on Docker and Liberty:
https://hub.docker.com/_/websphere-liberty
- ▶ Liberty features supported in Bluemix:
<https://www.ng.bluemix.net/docs/starters/liberty/index.html#libertyfeatures>
- ▶ Liberty profile security topic:
http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/cwlp_sec.html
- ▶ Information on Liberty Asset Repository Service (LARS):
<https://github.com/WASdev/tool.lars>
- ▶ Configuration monitoring examples:
https://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/twlp_setup_dyn_upd.html?lang=en
- ▶ Class loaders:
https://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/twlp_classloader.html?lang=en

- ▶ Databases supported for the Liberty profile:
<http://www.ibm.com/support/docview.wss?rs=180&uid=swg27006921#8.5>
- ▶ Configuring database connectivity in the Liberty profile:
http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/twlp_dep_configuring_ds.html?lang=en
- ▶ Application-defined data sources:
http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/rwlp_ds_appdefined.html?lang=en
- ▶ Configuring a Liberty profile server to access MongoDB APIs directly:
http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/twlp_mongodb_create.html?cp=SSAW57_8.5.5%2F3-3-11-0-3-2-20-0-1
- ▶ Installing and configuring an Apache Derby database:
http://db.apache.org/derby/papers/DerbyTut/install_software.html#derby_download
- ▶ Information on the Health Center:
<http://www.ibm.com/developerworks/java/jdk/tools/healthcenter>
- ▶ Configuring bootstrap.properties settings:
http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/rwlp_logging.html?cp=SSAW57_8.5.5%2F3-17-0-0
- ▶ Logging messages:
http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/rwlp_messages.html?cp=SSAW57_8.5.5%2F3-17-0-7
- ▶ Binary logging service:
http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/twlp_confHPEL.html?cp=SSAW57_8.5.5%2F1-0-2-10-1
- ▶ Configuring the **binarylog** command:
http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/rwlp_logviewer.html?cp=SSAW57_8.5.5%2F1-0-2-10-0
- ▶ Bluemix Instant Runtimes:
https://www.ng.bluemix.net/docs/starters/rt_landing.html
- ▶ Liberty for Java runtime, powered by the Cloud Foundry Liberty build pack:
<https://www.ng.bluemix.net/docs/starters/liberty/index.html>
- ▶ Boilerplates that use Liberty for Java for its runtime in the Bluemix catalog:
<https://console.ng.bluemix.net/catalog>

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



SG24-8170-01

ISBN 0738441082

Printed in U.S.A.

Get connected

