

IBM Cúram Social Program Management
Version 7.0.10

*IBM Cúram Social Program Management
on Kubernetes*



Note

Before using this information and the product it supports, read the information in [“Notices” on page 34](#)

Edition

This edition applies to IBM® Cúram Social Program Management v7.0.10 and to all subsequent releases unless otherwise indicated in new editions.

Licensed Materials - Property of IBM.

© **Copyright International Business Machines Corporation 2020, .**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

- Figures..... iv**
- Tables..... v**
- Chapter 1. IBM Cúram Social Program Management on Kubernetes 1**
- Chapter 2. Kubernetes architecture..... 3**
 - Transaction isolation.....3
 - Messaging architecture.....6
 - Elasticity 8
- Chapter 3. Deploying on WebSphere Application Server Liberty..... 10**
 - Installing prerequisite and additional software.....10
 - Managing deployment properties..... 11
 - Configuring WebSphere Liberty.....12
 - Configuring a web server plug-in.....14
 - Configuring security..... 16
 - Default configuration for WebSphere Liberty..... 16
 - Changing the JMS password..... 17
 - Logging the authentication process.....18
 - Configuring single sign-on.....19
 - Building EAR files..... 28
 - Deploying applications..... 31
 - Testing the deployment by logging in to the application.....32
 - Debugging IBM Cúram Social Program Management.....32
 - Known issues and limitations..... 33
- Notices.....34**
 - Privacy Policy considerations..... 35
 - Trademarks..... 35

Figures

- 1. Social Program Management on IBM Cloud Kubernetes Service..... 3
- 2. Kubernetes cluster with JMS Producer and JMS Consumer replicas..... 4
- 3. JMS-based messaging architecture..... 6
- 4. Kubernetes cluster architecture with elasticity implemented through elastic replicas..... 9
- 5. IdP-initiated flow..... 21
- 6. SP-initiated flow..... 22
- 7. SSO configuration components..... 24

Tables

1. List of XML files and descriptions.....	13
2. Property overrides and their default values.....	15

Chapter 1. IBM Cúram Social Program Management on Kubernetes

You can build IBM Cúram Social Program Management as a containerized application by using WebSphere® Application Server Liberty, IBM MQ Long Term Support (IBM MQ LTS), and Docker. You can then deploy the containerized application by using Helm charts and IBM Cloud Kubernetes Service. The [IBM Cúram Social Program Management on Kubernetes Runbook](#) provides instructions about how to containerize IBM Cúram Social Program Management and is accompanied by an open source [git repository](#) that contains sample Helm charts, Docker files, and other assets.

As human services organizations adapt to meet citizens needs, the complexity of their IT systems can grow. This can be challenging, especially when trying to manage the addition of new features for case workers, enacting new changes in legislation, or preparing for increases in demand for Universal Access application renewals.

To support cloud native architectures, IBM Cúram Social Program Management has been enhanced to support the technologies that are described in the following list from version 7.0.10.0.

WebSphere Application Server Liberty

IBM Cúram Social Program Management supports WebSphere Application Server Liberty only when it is containerized and deployed on IBM Cloud Kubernetes Service.

The architecture of WebSphere Application Server Liberty provides a low-overhead Java™ runtime environment that is suited for hosting cloud applications. WebSphere Application Server Liberty has been designed to optimize ease of development and the minimization of operational costs. From a development perspective, it supports many programming frameworks such as Sprint and Tapestry, and provides easy integration with Docker, Chef, Jenkins, Node.js, Java Platform, Enterprise Edition (Java EE), and Linux.

IBM Cloud Kubernetes Service

IBM Cúram Social Program Management supports IBM Cloud Kubernetes Service only when containerized with WebSphere Application Server Liberty .

IBM Cloud Kubernetes Service is a managed container service that is built on the open source Kubernetes system for automating the deployment, scaling, and management of containerized applications, while adding in IBM-specific capabilities. IBM Cloud Kubernetes Service provides scheduling capabilities, self-healing, horizontal scaling, service discovery and load balancing, automated rollouts and rollbacks, and secret and configuration management. The Kubernetes service also has advanced capabilities that are related to simplified cluster management, container security and isolation policies, the ability to design your own cluster, and integrated operational tools for consistency in deployment.

Docker

IBM Cúram Social Program Management supports Docker for packaging IBM Cúram Social Program Management for deployment on IBM Cloud Kubernetes Service.

Docker is an open platform that enables organizations to package, develop, run, and ship applications in environments called containers. A container is a unit of software that includes the dependencies, libraries, and configuration files that are needed to run the application in a docker container image.

Developers can now package an IBM Cúram Social Program Management application in containers for deployment on IBM Cloud Kubernetes Service, and include all the dependencies, libraries, and configuration files that are needed to run the application in a container image. The newly created container images can be downloaded from the container registry and installed in all stages of your environment, therefore simplifying deployments.

Helm

Helm is a package manager that helps you to find, share, and use software that is built for Kubernetes . Helm streamlines the installation and management of Kubernetes applications.

Kubernetes can become complex, and developers need to consider services, ConfigMaps, pods, and persistent volumes, in addition to managing the number of releases. Helm provides an easier way to package everything into one application and to advertize what can be configured.

IBM Cúram Social Program Management supports Helm for deploying IBM Cúram Social Program Management containers on IBM Cloud Kubernetes Service.

IBM MQ Long Term Support

IBM MQ LTS offers proven, enterprise-grade messaging capabilities that safely move information between applications.

IBM Cúram Social Program Management requires IBM MQ Long Term Support when containerized and deployed on IBM Cloud Kubernetes Service .

Technology updates

WebSphere Application Server Liberty , IBM Cloud Kubernetes Service, Docker, Helm, and IBM MQ Long Term Support updates occur throughout the year. IBM Cúram Social Program Management will be updated frequently to adopt newer versions of the previous technologies. For more information about the exact supported versions, see the [system prerequisites report](#).

Up next...

In the subsequent topics in this section, you can read more about the architectural differences in WebSphere Application Server Liberty that impact IBM Cúram Social Program Management, and about how to deploy to WebSphere Application Server Liberty in a native, single-server environment.

Chapter 2. Kubernetes architecture

IBM Cúram Social Program Management has been enhanced in version 7.0.10.0 to enable it for deployment into cloud native hosting platforms. While previously Social Program Management could be cloud-hosted in an IaaS cloud delivery model, it was not possible to leverage the benefits of flexibility, elasticity, efficiency and the strategic value offered by cloud native architecture.

Social Program Management can be built as a containerized application by using WebSphere Application Server Liberty, packaged as Docker containers, orchestrated by Kubernetes, and then run on IBM Cloud Kubernetes Service. Note that database support and IBM MQ support remain on VMs as part of the initial offering.

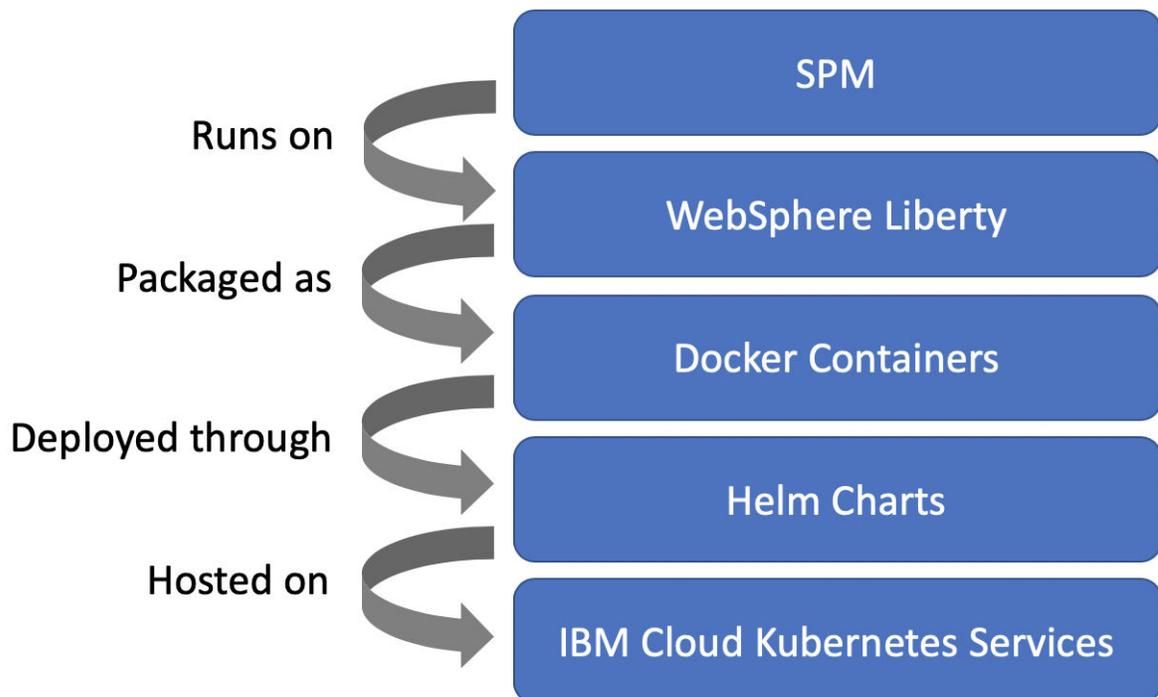


Figure 1. Social Program Management on IBM Cloud Kubernetes Service

Containerization makes it easier for developers to develop, deploy, and operate applications by simplifying the packaging and deployment process. For information that includes the steps, sample Helm charts, and Docker files that are required to containerize your Social Program Management application, see the [IBM Cúram Social Program Management on Kubernetes Runbook](#).

To support containerized cloud native architectures, fundamental architectural changes were required. The following section documents the changes, which apply only when Social Program Management runs on IBM Cloud Kubernetes Service.

Transaction isolation

In relation to IBM Cúram Social Program Management, the following two fundamental differences exist between traditional IBM WebSphere Application Server, Oracle WebLogic Server, and WebSphere Application Server Liberty:

- Traditional WebSphere Application Server and Oracle WebLogic Server enables the creation of multiple thread pools within the same JVM. However, WebSphere Application Server Liberty has a single thread pool, which is named the default executor, that is used to run all application threads.

- WebSphere Application Server Liberty aligns with the EJB specifications, which declare that EJB remote interfaces use pass-by-value, and that EJB local interfaces use pass-by-reference. While traditional IBM WebSphere Application Server and Oracle WebLogic Server provide an optimization to use pass-by-reference with EJB remote interfaces when the clients of the interfaces are collocated in the same JVM, WebSphere Liberty doesn't provide such an optimization.

The previous differences introduced a risk of thread exhaustion in the runtime application. To mitigate against the risk, the following multi-faceted solution was developed:

1. Isolate the client HTTP initiated transactions and the JMS initiated transactions.
2. Introduce EJB session bean local interfaces.

Thread isolation

In traditional IBM WebSphere Application Server, the `WebContainer` thread pool is set up to process HTTP requests, and the `SIBJMSRThreadPool` is set up to process JMS messages. Therefore, the client HTTP initiated transactions and the JMS initiated transactions can be isolated on the same JVM. A similar concept applies to Oracle WebLogic Server. However, HTTP and JMS initiated transactions cannot be isolated in WebSphere Application Server Liberty because it has one single thread pool, which is named the default executor, that runs all application and JMS processing.

To mitigate against the risk of thread exhaustion, the client HTTP initiated transactions and the JMS initiated transactions run on different WebSphere Application Server Liberty instances, integrated through an IBM MQ messaging engine. The Application/EAR file that is responsible for processing client HTTP initiated transactions is called the `JMS Producer`. The `JMS Producer` has no JMS message consumption because the EJB message driven beans (MDBs) are disabled. The Application/EAR file that is responsible for processing JMS initiated transactions is called the `JMS Consumer`. The `JMS Consumer` has JMS message consumption because the EJB MDBs are enabled. See the following diagram:

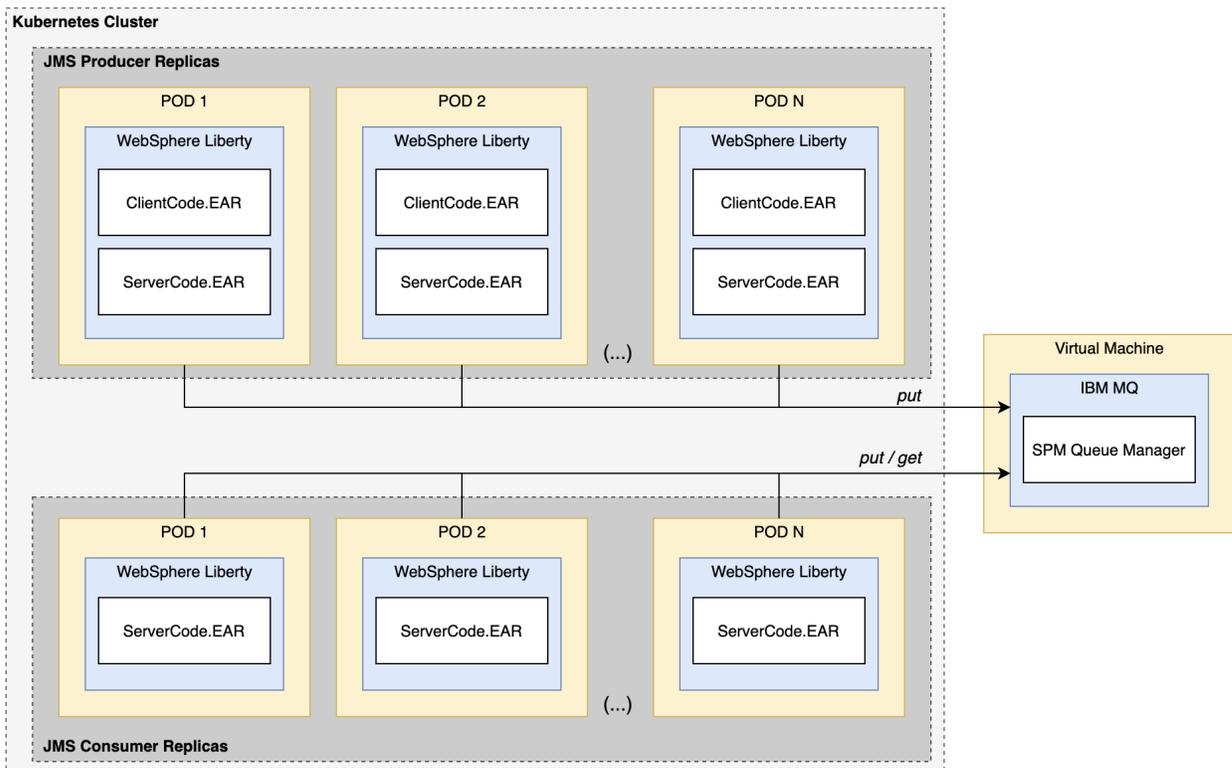


Figure 2. Kubernetes cluster with JMS Producer and JMS Consumer replicas

Note the following important points about the `JMS Producer` and the `JMS Consumer`:

Queue manager

IBM Cúram Social Program Management allows only one dedicated queue manager per set of JMS Producer replicas. Also, IBM Cúram Social Program Management allows only one set of JMS Consumer replicas per queue manager. Finally, IBM Cúram Social Program Management allows only one queue manager per IBM Cúram Social Program Management application.

JMS message processing

JMS Producer does not process any JMS message except the cache invalidation messages. The JMS Consumer does not have the IBM Cúram Social Program Management web interface available. After IBM Cúram Social Program Management puts a JMS message on the queue manager, all further processing is handled by the JMS Consumer, which can also put subsequent messages on the queue manager.

Independent scaling

A benefit of the split is that the JMS Producer and the JMS Consumer can scale independently. For example, if a significant increase in backend processing is expected because of eligibility and entitlement calculations, the JMS Consumer can scale up independently of the JMS Producer. After the backend processing is completed, the JMS Consumer can scale down to the normal operational architecture.

Server code split

The solution was facilitated by splitting the server code. For more information about the server code split, see [“Building EAR files”](#) on page 28.

EJB local interfaces

When an EJB session bean remote interface is started by using the pass-by-reference optimization in traditional IBM WebSphere Application Server and in Oracle WebLogic Server, the call is made on the same thread. However, WebSphere Application Server Liberty aligns with the EJB specifications, which declare that EJB remote interfaces use pass-by-value and EJB local interfaces use pass-by-reference. As a result, every EJB session bean remote interface invocation results in the use of a new thread from the default executor thread pool.

The existing remote interfaces have been preserved for traditional IBM WebSphere Application Server and Oracle WebLogic Server. An EJB local interface has been added to the following session beans:

- EJBMethodBean
- LoginBean
- AsyncMethodBean
- SLMTimerBean
- JDETimerBean

Also, a new JAR file that is named `coreinf-ejb-interfaces.jar` has been created, which is a consolidation of all the duplicate interface classes in Social Program Management. In WebSphere Application Server Liberty, the `coreinf-ejb-interfaces.jar` file has been added to a shared resources directory. Therefore, the JAR file is available for each Application/EAR file. In traditional deployments such as traditional IBM WebSphere Application Server and Oracle WebLogic Server, each Application/EAR file has been updated to include the new JAR file within its library path.

For developers, the addition of the new JAR file requires an update to the Eclipse class path, which will be done automatically if the `build createClasspaths` build target is started as part of a build. For more information, see the [Eclipse .classpath](#) file section.

The following classes have been added to the new `CuramSDEJ/lib/coreinf-ejb-interfaces.jar` file:

- AsyncMethod.class
- AsyncMethodLocal.class
- Authentication.class
- AuthenticationBase.class

- AuthenticationLocal.class
- Method.class
- MethodImpl.class
- MethodLocal.class
- SvrRemoteException.class
- TimerMethod.class
- TimerMethodBase.class
- TimerMethodLocal.class

The following classes have been removed from the CuramSDEJ/lib/jde-commons.jar file, and added to the CuramSDEJ/lib/coreinf-ejb-interfaces.jar file:

- AsyncMethod.class
- Authentication.class
- Method.class

The following classes have been removed from the CuramSDEJ/lib/coreinf.jar file, and added to the CuramSDEJ/lib/coreinf-ejb-interfaces.jar file:

- Authentication.class
- Method.class
- TimerMethod.class

Related information

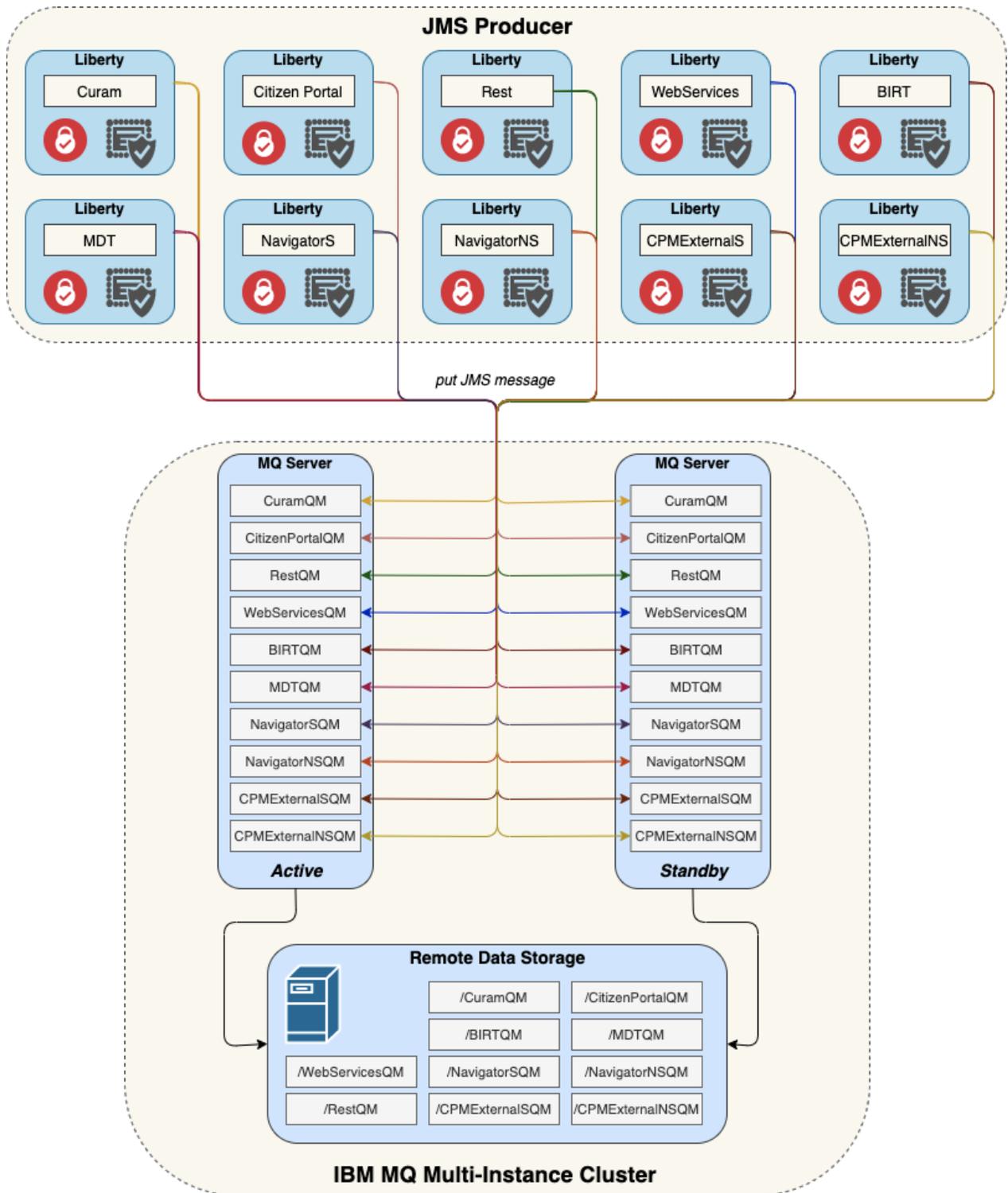
[Thread pools](#)

Messaging architecture

When IBM Cúram Social Program Management is containerized on Kubernetes, it uses IBM MQ to manage JMS messages for Cúram Deferred Processes and Cúram Workflows.

Each Social Program Management application, such as Cúram, Citizen Portal, Rest, and so on, must have its own dedicated queue manager. The following reference diagram illustrates the JMS-based messaging architecture:

Figure 3. JMS-based messaging architecture



For more information about the Social Program Management JMS Producer, see the [Transaction Isolation](#) topic.

Note: Social Program Management supports only IBM MQ on a VM. IBM MQ on Kubernetes, IBM MQ as a service, and other message engines have not been verified with Social Program Management.

Multi-instance queue manager support

For IBM MQ Cluster, Social Program Management supports only multi-instance queue managers, with one active/primary queue manager, and one standby/secondary queue manager. Multi-instance queue

managers are instances of the same queue manager that are configured on different servers. One instance of the queue manager is defined as the active instance, and another instance is defined as the standby instance. If the active instance fails, the multi-instance queue manager restarts automatically on the standby server.

For more information about multi-instance queue managers, see the [IBM MQ](#) product documentation.

IBM MQ and queue managers for Social Program Management

The cardinality between queue managers and IBM MQ servers is flexible. You can configure all queue managers on one IBM MQ cluster, or you can configure one queue manager per IBM MQ cluster. For example, you can configure some queue managers for an internal application in one IBM MQ cluster, and some queue managers for external applications in another IBM MQ cluster. The same cardinality applies to the remote data storage. The configuration depends on the level of fault tolerance and security isolation that is required by the application in production.

Elasticity

In Kubernetes , you can implement elastic replicas. Elasticity is the ability to scale up or down pods and nodes to adjust to the load to meet the end user demand.

Kubernetes cluster architecture with elasticity

Unlike middleware managed clusters like traditional IBM WebSphere Application Server Network Deployment, in WebSphere Application Server Liberty on Kubernetes , each IBM Cúram Social Program Management application is independent, and a middleware-managed cluster does not exist. The multiple replicas of the same application are independent, and the management of the replicas is delegated to Kubernetes . Because the replicas are independent from each other, they can be elastic and scale up or down as required. The following diagram below illustrates the architecture:

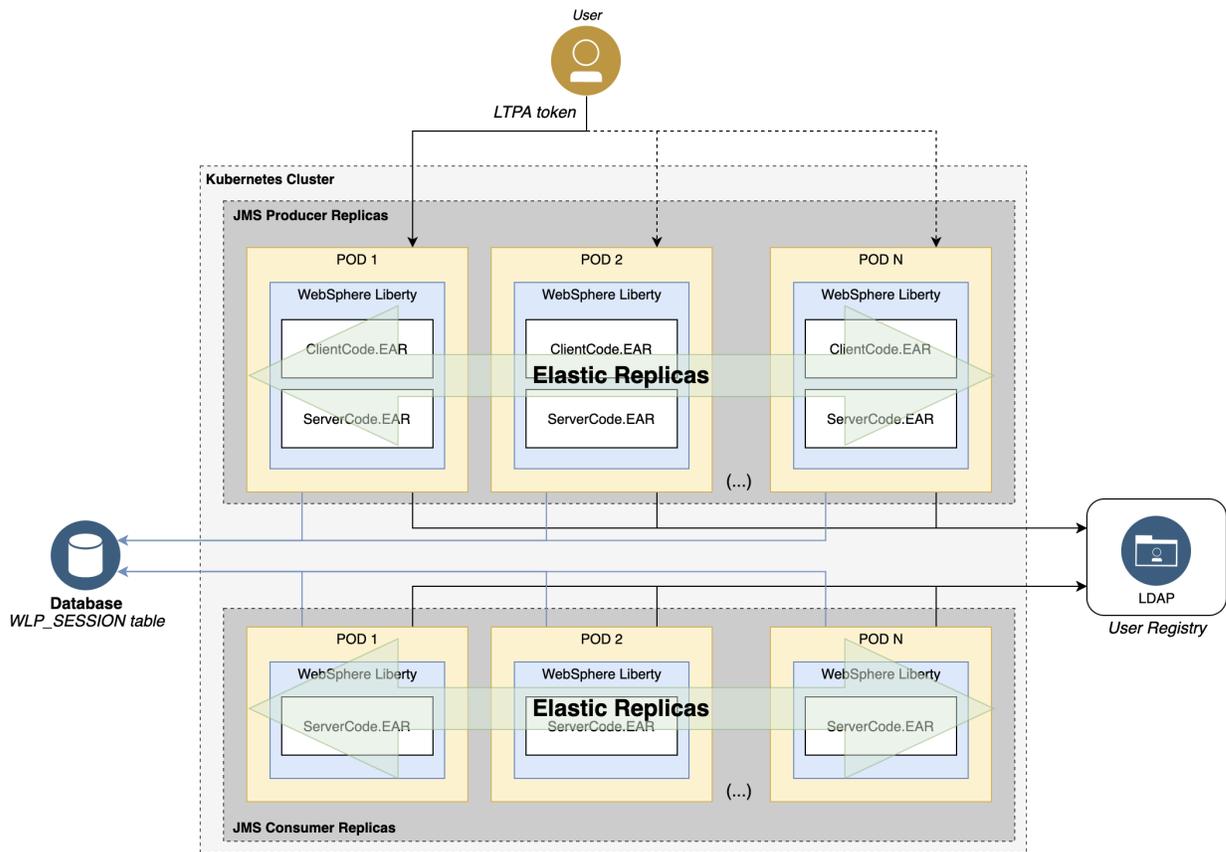


Figure 4. Kubernetes cluster architecture with elasticity implemented through elastic replicas

In the previous diagram, the end user is connected to the application that is hosted on POD 1. If POD 1 goes down, the user connection fails over to another available pod in the same application, assuming that multiple replicas of the same application are available. Although the session is persisted in the database, because of a limitation on WebSphere Liberty 19.0.0.12, the WebSphere Liberty Token Login Module requires a user registry to decrypt the LTPA token, validate the user, and update the shared state in the failed over pod with the user information. The Curam Login Module requires the Token Login Module behavior to preserve the user authentication on the failed-over pod. Therefore, the Token Login Module must be called before the Curam Login Module in the JAAS authentication process. For more information about the configuration, see the related link to the security configuration section.

You can configure the target database and table name to store the session in the WebSphere Liberty `server.xml` file.

User registry support

Although WebSphere Application Server Liberty supports custom user registry, IBM Curam Social Program Management has verified only LDAP as a user registry for the Token Login Module to verify users to support elasticity.

WebSphere Liberty collectives support

IBM Curam Social Program Management does not support WebSphere Application Server Liberty collectives on Kubernetes .

Related concepts

Configuring security

The default security configuration for WebSphere Liberty, provided by the Ant **configure** target, is a basic configuration and must not be used without modification. Your in-house security team should review this basic security configuration to reflect your security requirements and modify as needed.

Chapter 3. Deploying on WebSphere Application Server Liberty

Because you can now deploy and run IBM Cúram Social Program Management on WebSphere Liberty, you can benefit by deploying Social Program Management on standalone WebSphere Liberty instances for the purposes of testing and development. For a successful deployment on WebSphere Liberty, you must install the required software, configure WebSphere Liberty for security and deployment, build your application EAR (Enterprise ARchive) files, and then install your application EAR files.

Installing prerequisite and additional software

Install the prerequisite software, including Java and WebSphere Liberty from the IBM installation media by using IBM Installation Manager. Then, create system environment variables and add paths for software packages such as WebSphere Liberty, Java, and Apache Ant.

Prerequisite software

Before you can use IBM Cúram Social Program Management, you must install and configure the following software:

- IBM Cúram Social Program Management. For more information, see [Installing Cúram](#).
- WebSphere Liberty. For more information, see [Installing Liberty](#).
- Java Platform, Enterprise Edition 8. Liberty Profile requires the full Java Platform, Enterprise Edition 8. For more information on installing Java, see [Installing, updating, and uninstalling IBM SDK, Java Technology Edition](#).
- Apache Ant. For more information, see [Apache Ant](#). For version information, see the [system prerequisites report](#).
- For more information on minimum hardware requirements, see [Hardware requirements for Cúram Social Program Management \(SPM\) Development and Test Environments](#).

Additional software

For some functions you might need additional software, depending on your site requirements as follows:

- Databases - You can use Oracle or IBM Db2[®] for LUW Family (Linux/Unix/Windows) database servers. For more information, see [Installing DB2[®]](#). You cannot configure WebSphere Liberty to use the open source H2 database (supplied with Social Program Management) however, you can use it for development. For example, you can use it with the Ant **database** target to run JUnit tests.
- Web servers - For web browser access you can use the original WebSphere Liberty support or you can install IBM HTTP Server. For more information, see [Adding a plug-in configuration to a web server](#).

Note: For JMS, you can use the Liberty-embedded JMS or IBM MQ 9.1 and its associated resource adapter for WebSphere Liberty. For more information, see [Installing IBM MQ](#). However, you cannot use Liberty-embedded JMS in a production environment.

Create environment variables and add paths

After you install the prerequisite software, create the following environment variables:

- ANT_HOME system environment variable with the value set to the Apache Ant installation directory. For example, `ANT_HOME=/Ant<version>`
- ANT_OPTS system environment variable with the value set to `-Xmx1400m`:

```
ANT_OPTS=-Xmx1400m -Dcmp.maxmemory=1400m
```

- WLP_HOME system environment variable with the value set to the WebSphere Liberty installation directory. For example, *WLP_HOME=/opt/IBM/WebSphere/Liberty*
- JAVA_HOME system environment variable with the value set to the Java SE Development Kit installation directory, not the Java Runtime Environment (JRE) software. For example, *JAVA_HOME=/Java<version>*

Add paths for the following software:

- Add the Ant bin folder to the system PATH environment variable. For example, *\$ANT_HOME/bin*
- Add the WebSphere Liberty bin folder to the system PATH environment variable. For example, *\$WLP_HOME/bin*
- Add the Java bin folder to the system PATH environment variable. For example, *\$JAVA_HOME/bin*

Managing deployment properties

Create two properties files, `Bootstrap.properties` and `AppServer.properties`. The properties that you define in these files are used to deploy and customize IBM Cúram Social Program Management WebSphere Liberty. Then, verify your property files and environment by running the **configtest** target.

Bootstrap.properties

`Bootstrap.properties` contains the properties that are needed to connect to the database.

Create a `Bootstrap.properties` and place it in the `$SERVER_DIR/project/properties` directory.

`Bootstrap.properties` is packed in the Enterprise Archive (EAR) when the EAR file is built. When packed in the EAR file, `Bootstrap.properties` is edited to contain a subset of properties of the source `Bootstrap.properties` file and is extended with relevant properties from `AppServer.properties`.

WebSphere Liberty has its own `bootstrap.properties` (note the lowercase "b") that contains the properties for WebSphere Liberty runtime. For more information, see [Specifying Liberty bootstrap properties](#).

Sample Bootstrap.properties

```
# Tnameserv Port
curam.environment.tnameserv.port=900
curam.environment.bindings.location=C:/Bindings

curam.db.username=db2admin
curam.db.password=wwW5UTMnFOe1SeCBEQy/Zg==
curam.db.type=DB2
curam.db.name=CURAM
curam.db.serverport=50000
curam.db.servername=localhost

# property to specify Oracle service name.
curam.db.oracle.servicename=orcl.<host_name>

# For remote mode also specify:
curam.db.serverport=9092
curam.db.servername=localhost

# Lock Time Out in ms. Default is 1000, i.e. 1 second. (Optional)
curam.db.h2.locktimeout=20000

# Property to disable MVCC. Default: true. (Optional)
curam.db.h2.mvcc=true
```

AppServer.properties

`AppServer.properties` is used to specify properties relevant to your application server environment.

Create an `AppServer.properties` file and place it in the `$SERVER_DIR/project/properties` directory.

The user that is defined by the `security.username` and `security.password` properties is assigned to the WebSphere Liberty administrator role by the Ant **configure** target. This role provides access to the WebSphere Liberty JMX methods and MBeans by using its REST connector.

Sample `AppServer.properties`

```
# Property to indicate that WebSphere Liberty is installed
as.vendor=WLP

# The username and password for the administrator role
security.username=websphere
# Encrypt the plain-text password using 'build encrypt -Dpassword=<password>'
# Below is the encryption for the default password ("websphere")
security.password=X0VRjjVTebM8gV953LGMLQ==

# The name of the server on which the application will be hosted
curam.server.name=CuramServer

# The Curam client HTTP port
curam.client.httpport=10101

# The Curam web service port
curam.webservices.httpport=10102

curam.server.port=2809
curam.db.auth.alias=databaseAlias
```

Check your settings

When you create the properties files, check your settings by running the Ant **configtest** target:

```
cd $SERVER_DIR
./build.sh configtest
```

Review the output for any errors or warnings and resolve them.

Configuring WebSphere Liberty

Configure WebSphere Liberty to reflect your tuning needs and organizational requirements. Use the properties that you defined in the `.properties` files to configure the database, security settings, and default JMS. For more information, see “Managing deployment properties” on page 11. Then, customize the `server.xml` and the files that it includes to reflect your implementation of IBM Cúram Social Program Management.

Configure a WebSphere Liberty server

Run the Ant **configure** target from the `$SERVER_DIR` directory:

```
./build.sh configure
```

The Ant **configure** target configures a WebSphere Liberty server for IBM Cúram Social Program Management by using the properties that you defined in `AppServer.properties` and `Bootstrap.properties`. Configuration items include the database configuration, security settings, and default JMS configuration.

Note: `AppServer.properties` and `Bootstrap.properties` are in the `$SERVER_DIR/project/properties` directory. You can override the default location for the properties files by specifying **-Dprop.file.location=<new location>** when you run the **configure** target.

WebSphere Liberty configuration files that change when you run the Ant **configure** target

The Ant **configure** target changes the `$WLP_HOME/usr/servers/CuramServer/server.xml` file and the files that it includes so that they contain WebSphere Liberty features and configurations to support IBM Cúram Social Program Management. The configuration files are placed in `$WLP_HOME/usr/`

servers/CuramServer/adc_conf/. Table 1 lists all the changed files and describes the role of each file.

<i>Table 1. List of XML files and descriptions</i>	
XML file	Description
server_endpoints.xml	Specifies the application port configuration (based on AppServer.properties).
server_logging.xml	Specifies the WebSphere Liberty logging configuration.
server_resources_jdbc_DB2.xml or server_resources_jdbc_ORA.xml	Specifies the required Cúram database configuration. The file that is used is determined by your database configuration in Bootstrap.properties.
server_resources_messaging.xml	Specifies the embedded JMS configuration that is required by IBM Cúram Social Program Management.
server_resources_tx.xml	Specifies WebSphere Liberty transaction settings.
server_security.xml	Specifies a basic security configuration.
When you run the installapp or uninstallapp targets the following files are modified. For more information, see “Deploying applications” on page 31:	
server_applications.xml	Specifies global application settings for the WebSphere Liberty server and includes an application-specific file for each installed application EAR file.
application_*.xml	One file for each installed application EAR file is created. For example, installing Curam.ear generates an application_Curam.xml file.

Customizing the Cúram WebSphere Liberty server configuration

You can customize the server.xml and the files that it includes as described in the [Liberty Knowledge Center](#) to meet your requirements, however, note the following restrictions:

- Make your changes by changing your Bootstrap.properties and AppServer.properties files and running the Ant **configure** target. For more information, see [“Managing deployment properties”](#) on page 11.
- You must track and document all custom changes for your own records.

You can integrate your configuration changes with the Cúram WebSphere Liberty server in one of the following ways:

- By changing the adc_conf/server_extra_config.xml file.
- By using a custom Ant script to make WebSphere Liberty configuration changes.

The following sections describe the steps that are required to manage customizations.

Changing the adc_conf/server_extra_config.xml file

When the server is configured by the Ant **configure** target, it places a functionally empty server_extra_config.xml configuration file in the following server directory:

```
${server.config.dir}/adc_conf
```

This file is intended for your customizations. You can edit or replace the contents of server_extra_config.xml. The changes that you make take effect when you restart the server.

You can combine editing `server_extra_config.xml` and a customized Ant script, by using the Ant script to modify:

```
{server.config.dir}/adc_conf/server_extra_config.xml
```

as described in the following section.

Using a custom Ant script to make Liberty custom configuration changes

You can extend the Ant **configure** target by creating a custom Ant script named `extra_wlp_configuration.xml`. Place the script in `$CURAMSDEJ/bin` with the following contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="extra_wlp_configuration" default="setup" basedir=".">
<property name="do.extra.file" value="extra_wlp_configuration."/>
<!-- ***** -->
<!-- *** D O . E X T R A *** -->
<!-- ***** -->
  <target name="do.extra" description="Customization of the Liberty configuration for Curam.">

    <!-- Your custom Ant script code. -->

  </target>
</project>
```

Replace the `<!-- Your custom Ant script code. -->` with your custom script code. For example, you might use your `extra_wlp_configuration.xml` script to modify.

```
{server.config.dir}/adc_conf/server_extra_config.xml
```

to include your custom WebSphere Liberty configuration for IBM Cúram Social Program Management.

The Ant script in `$CURAMSDEJ/bin/extra_wlp_configuration.xml` can also be started independently of the Ant **configure** target by using the **do.extra** target:

```
./build.sh do.extra
```

Configuring a web server plug-in

Run the Ant **configurewebserverplugin** target to configure the web server plug-in to work with IBM HTTP Server and IBM Cúram Social Program Management deployed on WebSphere Liberty. The target is suitable for internal development and testing purposes only. It uses generated, self-signed certificates that are not appropriate for public access. For more information, see [Configuring a web server plug-in for Liberty](#).

Ensure that the following paths and commands are available in your environment:

- The WebSphere Liberty server must be running.
- The plug-in path must be available. This path defaults to `/opt/IBM/WebSphere/Plugins`. If this default is not appropriate for your environment, you must override `plugin.home` on the command line when you start the target.
- The **openssl** command must be available in the environment by using the PATH (available on most platforms, or can be added as a separate activity).
- The **keytool** command must be available in the environment by using the PATH (available using the Java installation).
- The **gskcmd** command must be available in the environment by using the PATH (available in the IBM HTTP Server bin folder).
- The WebSphere Liberty **pluginUtility** command must be available in the environment by using the PATH.

Running the Ant `configurewebserverplugin` target

Running the Ant `configurewebserverplugin` target requires properties that are passed on the command line and provides other optional property overrides. You must specify the following non-defaulting property overrides:

- **-Dcertificate.password=** - specifies the password that you require for the WebSphere Liberty and plug-in certificate files.
- **-Dsubject.string=** - specifies a subject string for the generated self-signed certificate, for example:

```
-Dsubject.string="/CN=`hostname -f`/O=MyOrg/OU=MyOrgUnit/L=MyLocation/ST=MyLocationState/C=IE"
```

Table 1 lists the optional property overrides available for use with the target.

Property Name	Maps To	Default
<code>certificate.location</code>	All openssl output arguments	Defaults to: <code>\$WLP_HOME/usr/servers/\${curam.server.name}/resources/security</code>
<code>subject.string</code>	<code>-subj</code> argument of openssl	No default, the property must be specified
<code>certificate.days</code>	<code>-days</code> argument of openssl	Defaults to: 3650
<code>key.length</code>	numbits value of openssl	Defaults to: 2048
<code>certificate.password</code>	All certificate files and or stores	No default, must be specified; this password is encoded in <code>adc_conf/server_security.xml</code>
<code>server.name</code>	Liberty file system folder names	No default, can be specified in <code>AppServer.properties</code> (the same as <code>curam.server.name</code>)
<code>curam.webserver.name</code>	The plug-in file system	Defaults to: <code>ihs_`hostname -s`</code> . If your environment does not support the hostname command with the -s argument, you must provide an override for <code>curam.webserver.name</code> on the command line. Note: When you run the <code>configure</code> target, the same requirement exists, that is, if <code>hostname -s</code> is not available, the webserver name defaults to <code>webserver1</code> as configured in <code>adc_conf/server_endpoints.xml</code> .

Run the Ant `configurewebserverplugin` target in the `$SERVER_DIR` directory. An example, minimal invocation is as follows:

```
cd $SERVER_DIR
./build.sh configurewebserverplugin -Dcertificate.password=MyPassword -Dsubject.string="/
CN=`hostname
-f`/O=MyOrg/OU=MyOrgUnit/L=MyLocation/ST=MyLocationState/C=IE"
```

When the Ant `configurewebserverplugin` target completes, you must restart the WebSphere Liberty server.

Note: The Ant `configurewebserverplugin` target is not supported in a Windows environment.

Configuring security

The default security configuration for WebSphere Liberty, provided by the Ant **configure** target, is a basic configuration and must not be used without modification. Your in-house security team should review this basic security configuration to reflect your security requirements and modify as needed.

The security configuration that is provided by the IBM Cúram Social Program Management is configured by `adc_conf/server_security.xml` and `adc_conf/server_endpoints.XML`.

The only default endpoint is provided in HTTPS and consists of the following elements:

- `server_security.xml` example:

```
<jaasLoginContextEntry id="system.WEB_INBOUND"
loginModuleRef="token,myCustomWebInbound,hashtable,userNameAndPassword,certificate"
name="system.WEB_INBOUND"/>
  <jaasLoginModule className="curam.util.security.CuramLoginModule" controlFlag="REQUIRED"
id="myCustomWebInbound" libraryRef="customLoginLib">
    <options exclude_usernames="websphere,SYSTEM" login_trace="true"/>
  </jaasLoginModule>
  <jaasLoginContextEntry id="system.DEFAULT"
loginModuleRef="token,myCustomDefault,hashtable,userNameAndPassword,certificate"
name="system.DEFAULT"/>
  <jaasLoginModule className="curam.util.security.CuramLoginModule" controlFlag="REQUIRED"
id="myCustomDefault" libraryRef="customLoginLib">
    <options exclude_usernames="websphere,SYSTEM" login_trace="true"/>
  </jaasLoginModule>
```

- `server_endpoints.XML` example:

```
<server description="Server host configuration">
  <httpEndpoint id="defaultHttpEndpoint" httpsPort="8443" httpPort="-1" host="*" />
</server>
```

Customize the example XML files to match your configuration.

- The configuration for the Cúram system login module, *CuramLoginModule*, is defined by various elements and this module is required configuration.
- A `<basicRegistry>` element to support default users and users that are not secured by the Cúram system login module. This element is configured to support the WebSphere Liberty administrator and Social Program Management JMS users.
- An `<orb>` element to support LTPA authentication.
- A web client `<ssl>` element, which is provided as an example to get you started with IBM Cúram Social Program Management in WebSphere Liberty.

Input to this basic security configuration is provided by credentials in the `AppServer.properties` file that you must set before you run the Ant **configure** target.

There are many other security options and settings that are provided by WebSphere Liberty that you can use, provided they are compatible with the default IBM Cúram Social Program Management security requirements such as the system login module. For more information see, [Securing Liberty and its applications](#).

Related concepts

[Elasticity](#)

In Kubernetes, you can implement elastic replicas. Elasticity is the ability to scale up or down pods and nodes to adjust to the load to meet the end user demand.

Default configuration for WebSphere Liberty

The Cúram Java Authentication and Authorization Service (JAAS) login module is configured as a JAAS login module in WebSphere Liberty. The default, scripted security configuration provided by Curam for

WebSphere Liberty configures the Cúram custom JAAS login module and the basicRegistry for non-application users.

Multiple JAAS login contexts exist for WebSphere Liberty. The Cúram JAAS login module is configured for the DEFAULT and WEB_INBOUND configurations. The same login module is used for all three configurations. WebSphere Liberty utilizes these contexts as follows:

- **DEFAULT**

The Cúram JAAS login module specified for the DEFAULT context is utilized for web services and JMS invocations.

- **WEB_INBOUND**

The Cúram JAAS login module specified for the WEB_INBOUND context is used for authentication of web requests

- **RMI_INBOUND**

The login modules that are specified for the RMI_INBOUND configuration are used for authentication of Java clients.

The Cúram JAAS login module is within a chain of login modules that are set up in WebSphere Liberty. It is expected that at least one of these login modules be responsible for adding credentials for the user. By default, the Cúram login module adds credentials for an authenticated Cúram application user. Therefore, Cúram users should not normally be added to the WebSphere Liberty basicRegistry.

As part of the security configuration the users specified by the *security.username* and *curam.security.credentials.async.username* properties in *AppServer.properties* are excluded from authentication by the Cúram JAAS login module and are specified in the WebSphere Liberty basicRegistry. The *security.username* user is classified as an administrative user and is not a Cúram application user.

Note: The *security.username* user is automatically added to the WebSphere Liberty basicRegistry by the SPM-provided configuration scripts. If an alternative, custom security configuration is in place it should take this user into account.

Changing the JMS password

After you have deployed the IBM Cúram Social Program Management application, change the JMS user password. The JMS user is the user under which JMS messages are run.

Before you begin

Change the JMS password during a period of no activity on the application server. Otherwise, JMS message processing might fail while the change is in process, until the application server is restarted. Ensure that the WebSphere Liberty server is started and the Social Program Management application is running.

Overview of the steps to change the user password

To change the JMS user password for deployed applications, take the following steps.

1. Change the password in *AppServer.properties*.
2. Change the password in the WebSphere Liberty configuration.
3. Change the password in the SPM administration system.

The following sections describe how to make each change.

Change the password in *AppServer.properties*

To change the password, update the *security.credentials.async.password* property in the *AppServer.properties* file.

The password must be encrypted by using the Ant **encrypt** target, for example:

```
cd $CURAMSDEJ/bin
./build.sh encrypt -Dpassword=<The password to be encrypted>
```

For more information, see [Cipher-Encrypted Passwords](#).

Change the password in the WebSphere Liberty configuration

To change the JMS user password in the WebSphere Liberty configuration, you must modify the `${server.config.dir}/adc_conf/server_security.xml` file and the `${server.config.dir}/adc_conf/application_*.xml` files, there is one `application_*.xml` for each deployed application.

To change the password, encrypt the new password before you replace it in the configuration files. Use the WebSphere Liberty **securityUtility encode** command to get the encrypted value for the new password. The encrypted password value of the `curam.security.credentials.async.password` property in `AppServer.properties` differs from the encrypted password value in the WebSphere Liberty configuration files due to different encryption techniques. Run the **securityUtility** command as follows:

```
securityUtility encode mypassword
```

The configurations that must be changed in the WebSphere Liberty configuration are for example:

- `${server.config.dir}/adc_conf/application_*.xml`: `<run-as userid="SYSTEM" password=...`
- `${server.config.dir}/adc_conf/server_security.xml`: `<user name="SYSTEM" password=...`

Change the password in the SPM administration system

Change the JMS password by using the administration user interface as follows:

1. Log in to Social Program Management with the admin user.
2. Under **Quick Links**, click **Search for a user...**
3. Enter **system** in the **Last name** field and click **Search**.
4. Click the **SYSTEM** user link that is returned.
5. Click the **Edit...** menu option and set the following fields:
 - Specify a **First Name**
 - Set **Sensistivity: 1**. Otherwise, an error occurs:
This user cannot have a greater sensitivity value than you.
 - Set the fields **New Password** and **Confirm Password**.
6. Click **Save**.
7. Restart WebSphere Liberty.

Logging the authentication process

The CuramLoginModule authentication process can be logged in `console.log` and `messages.log` files.

Trace entries of the CuramLoginModule authentication process can be generated in the `console.log` and `messages.log` files, which can be helpful for debugging.

To generate the log entries, add the following entry to `AppServer.properties` before running the Ant **configure** target:

```
curam.security.login.trace=true
```

For a previously configured server modify the `adc_conf/server_security.xml` file to change the `login_trace` attribute values from "false" to "true" and restart the WebSphere Liberty server.

Note: To enable extensive authentication logging, add the following logging configuration to the `server.xml` file:

```
<logging traceFileName="stdout" consoleLogLevel="INFO"
  traceSpecification="com.ibm.ws.security.*=all:com.ibm.ws.webcontainer.security.*=all:
  com.ibm.ws.session.*=all" />
```

Configuring single sign-on

Single sign-on (SSO) authentication enables users to access multiple secure applications by authenticating once with a single user name and password. Federated single sign-on that uses SAML 2.0 browser profile, using either an IdP-initiated HTTP POST binding or an SP-initiated HTTP POST binding, can be implemented through the IBM Cúram Social Program Management application.

If users authenticate to an SSO system, they are no longer prompted for credentials when they access any of the other applications that are configured to work with the SSO system.

SSO systems usually maintain the user accounts on a lightweight directory application protocol (LDAP) server. If user accounts are stored in one location, it is easier for system administrators to safeguard the accounts. Also, it is easier for users to reset one account password for multiple applications.

The following information describes the scenario where IBM Cúram Social Program Management is deployed on WebSphere Application Server Liberty.

SAML 2.0 single sign-on initiation and flow

For single sign-on, the SAML response, by HTTP POSTs, is interpreted and controlled by logic in IBM Cúram Social Program Management.

In all SAML web Single Sign-On (SSO) profile flows, the binding defines the mechanism that is used to send information through assertions between the identity provider (IdP) and the service provider (SP). WebSphere Liberty supports HTTP POST binding for sending web SSO profiles. The browser sends an HTTP POST request, whose POST body contains a SAML response document. The SAML response document is an XML document that contains data about the user and the assertion, some of which is optional.

Browser-based single sign-on (SSO) through SAML v2.0 works well with many web applications where the SAML flow is controlled by HTTP redirects between the identity provider (IdP) and the service provider (SP). The user is guided seamlessly from login screens to SP landing pages by HTTP redirects and hidden forms that use the browser to POST received information to either the IdP or the SP.

In a single-page application, all the screens are contained within the application and dynamic content is expected to be passed only in JSON messages through XMLHttpRequests. Therefore, the rendering of HTML content for login pages and the automatic posting of hidden forms in HTML content is more difficult. If the SP processes the content in the same way, it would leave the application and hand back control to either the user agent or the browser, in which case the application state would be lost.

IdP-initiated use case

The IdP can send an assertion request to the service provider ACS through one of the following methods:

- The IdP sends a URL link in a response to a successful authentication request. The user must click the URL link to post the SAML response to the service provider ACS.
- The IdP sends an auto-submit form to the browser that automatically posts the SAML response to the service provider ACS.
- The user authenticates into IdP and accesses the application that is configured as a partner to the IdP.

The ACS then validates the assertion, creates a JAAS subject, and redirects the user to the SP resource.

SP-initiated use case

When an unauthenticated user first accesses an application through an SP, the SP directs the user's browser to the IdP to authenticate. To be SAML specification compliant, the flow requires the generation of a SAML AuthnRequest from the SP to the IdP. The IdP receives the AuthnRequest, validates that the request comes from a registered SP, and then authenticates the user. When the user is authenticated, the IdP directs the browser to the Assertion Consumer Service (ACS) application that is specified in the AuthnRequest that was received from the SP.

Assertions and the SAML Response document

To prove the authenticity of the information, the assertion in the SAML response is almost always digitally signed. To protect the confidentiality of parts of the assertion, the payload can be digitally encrypted. A typical SAML response contains information that can be sent only through a login by a POST parameter. After login, an alternative mechanism is typically used to maintain the logged-in security context. Most systems use some cookie-based, server-specific mechanism, such as a specific security cookie, or the server's cookie tied to the user's HTTP session.

IdP-initiated flow

When Social Program Management is configured in WebSphere Liberty with an-IdP initiated web SSO flow, any attempt to connect to a protected resource without first authenticating through IdP results in the application server falling back to an SP-initiated SSO flow. In an SP-initiated SSO flow, any authentication requests that are initiated through SP result in a 403 HTTP response, and the application redirects the user to the IdP login page for the user to authenticate. After the user is authenticated successfully, the control is redirected to the Social Program Management application page.

The following figure illustrates the IdP initiated flow that is supported by Social Program Management in a default installation.

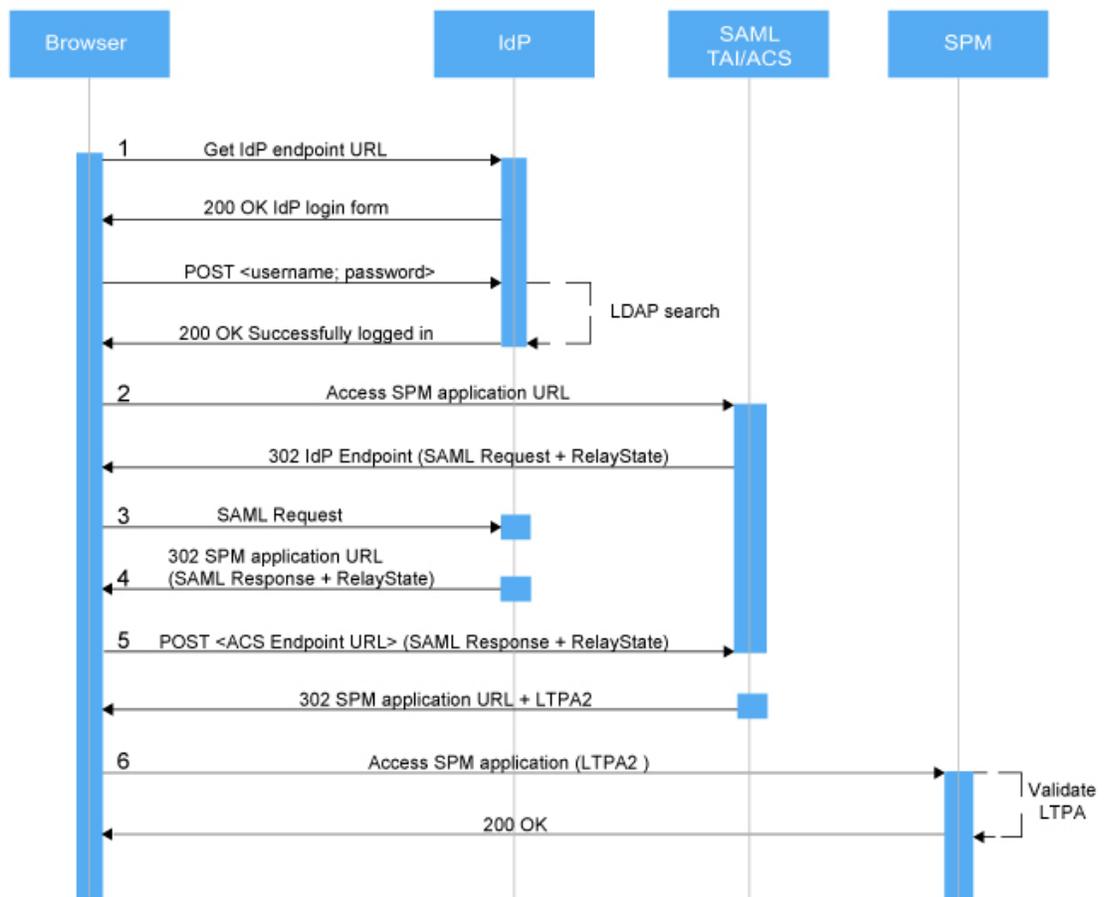


Figure 5. IdP-initiated flow

1. In an IdP-initiated flow, the user completes the IDP login form and authenticates.
2. After successful authentication in IdP, the user tries to access the SPM application that is deployed in the application server.
3. The Trust Association Interceptor (TAI) and Assertion Consumer Service (ACS) (SAML TAI/ACS) that is deployed on the application server intercepts the request and redirects it to the IdP endpoint with a generated SAML request.
4. Because the user already logged into the IdP before the user accessed the SPM application, the IdP responds with a SAML response and redirects the user to the SPM application.
5. The application server ACS validates the signature that is contained in the SAML Response. WebSphere Liberty also ensures that the originator is a Trusted Authentication Realm. If the validation is successful, the ACS sends an HTTP redirect request that points to the configured Social Program Management target landing page, along with an LTPA2 cookie that is used in any subsequent communication.
6. The Social Program Management application landing page is displayed in the browser.

SP-initiated flow

When Social Program Management is configured with an SP-initiated web SSO flow, any attempt to connect to a protected resource without first authenticating results in a 401 HTTP response from the application server Assertion Consumer Service's Trust Association Interceptor, and the generation of the SAML AuthnRequest message to be sent to the IdP.

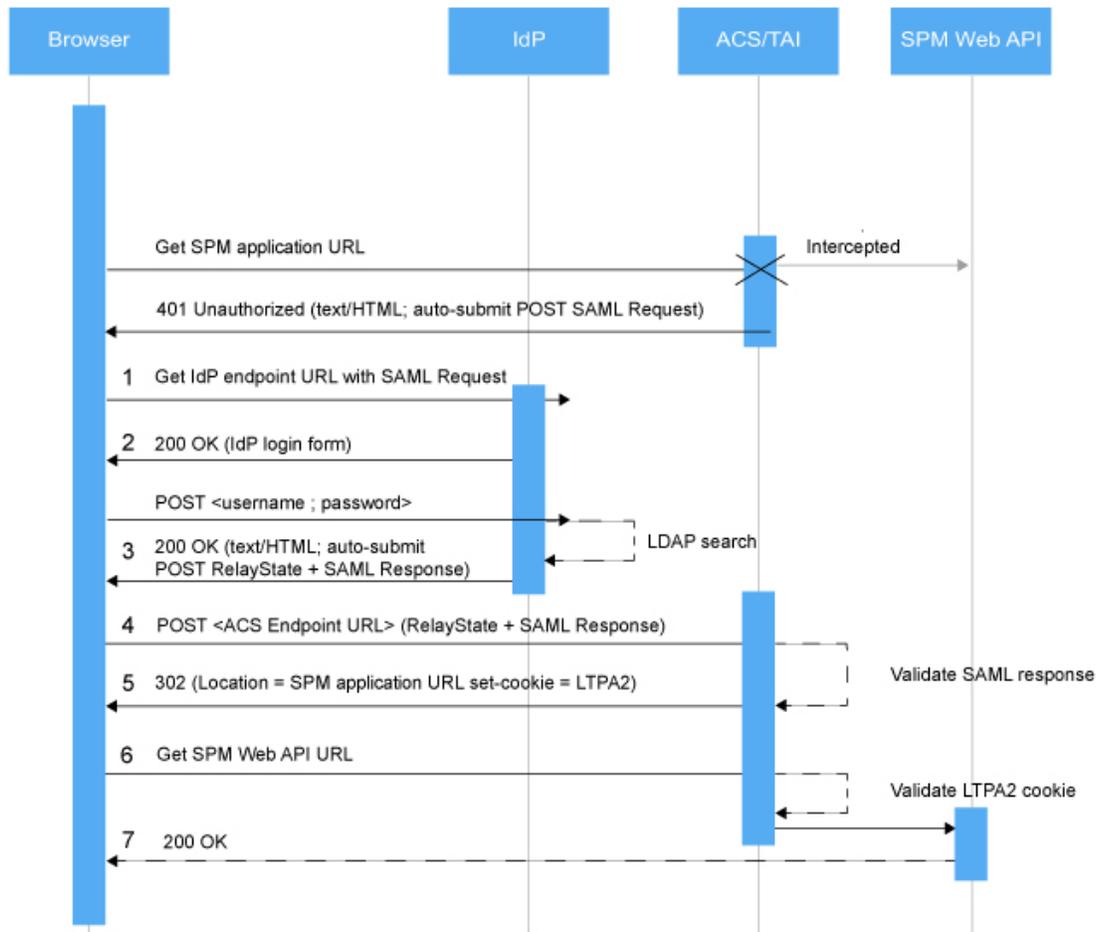


Figure 6. SP-initiated flow

1. When a user tries to access a Social Program Management application resource without authenticating, the TAI intercepts the request and redirects the user to the IdP endpoint with the generated SAML request.
2. The IdP endpoint displays the login form that the user completes to authenticate, then directs the SAML request to the IdP SAML endpoint.
3. After successful validation of the user credentials at the IdP, the IdP populates the SAML response and returns it in an HTML form that contains hidden input fields.
4. The HTML form is autosubmitted to the Social Program Management application with the SAML response and RelayState parameter.
5. The Social Program Management application extracts the RelayState parameter and SAML response values, and inserts them in a new POST request to the application server ACS.
6. The application server ACS validates the signature that is contained in the SAML response. WebSphere Liberty also ensures that the originator is a Trusted Authentication Realm. If the validation is successful, the ACS sends an HTTP redirect that points to the configured Social Program Management target landing page, along with an LTPA2 cookie that is used in any subsequent communication.
7. The Social Program Management application landing page is displayed in the browser.

Configure SAML SSO for IBM Cúram Social Program Management on WebSphere Liberty

Code samples and steps are provided as a guide for enabling SAML SSO in WebSphere Liberty.

About this task

The following code samples and steps are intended for general guidance only. They are not intended to be a substitute for detailed analysis or the exercise of professional judgment.

For more information about configuring SAML SSO for IBM Cúram Social Program Management on WebSphere Liberty, see the related link.

Procedure

1. Enable the SAML feature in the WebSphere Liberty `server.xml` file, as shown in the following example:

```
<featureManager>
  <feature>samlWeb-2.0</feature>
  <feature>appSecurity-2.0</feature>
</featureManager>
```

2. Download the SAML metadata XML and ask your SSO administrator to use it to configure the SSO provider as IBM ISAM, as shown in the following example:

```
https://application-domain.com/ibm/saml20/defaultSP/samlmetadata
```

3. Configure and enable SAML SSO in the WebSphere Liberty `server.xml` file, as shown in the following example:

```
<server description="Curam Server">
  (...)
  <samlWebSso20 id="defaultSP"
    idpMetadata="/path/to/file/federation_metadata.xml"
    wantAssertionsSigned="false"
    authnRequestsSigned="false"
    authFilterRef="curamAuthFilter"
    spHostAndPort="https://application-domain.com"
    disableLtpaCookie="false"
    allowCustomCacheKey="false"
    enabled="true">
  </samlWebSso20>
  <authFilter id="curamAuthFilter">
    <requestUrl id="curamRequestUrl1" urlPattern="/Curam/j_security_check"
    matchType="notContain"/>
    <requestUrl id="curamRequestUrl2" urlPattern="/Curam/logon.jsp" matchType="notContain"/>
    <requestUrl id="curamRequestUrl3" urlPattern="/Curam/logonerror.jsp"
    matchType="notContain"/>
  </authFilter>
  (...)
</server>
```

Note: The `federation_metadata.xml` file is generated by the identity provider, which is IBM ISAM.

4. In the `server.xml` file, change the `spHostAndPort="https://spm-application-url.com"` property to the appropriate domain URL.
5. Verify the authentication attributes that are extracted from the subject in the Curam JAAS Login Module, as shown in the following example:

```
Set<object> privateCredentials = loginSubject.getPrivateCredentials();
if (privateCredentials != null && privateCredentials.size() > 0) {
  for (Object credObject : privateCredentials) {
    if (credObject instanceof java.util.Hashtable) {
      java.util.Hashtable credPrivate = (java.util.Hashtable) credObject;
      username = (String)credPrivate.get("com.ibm.wsspi.security.cred.securityName");
      if (username != null && username.trim().length() > 0)
      {
        authenticationResult = true;
      }
    }
  }
}
```

Related information

[Configuring SAML Web Browser SSO in Liberty](#)

Single sign-on configuration example

IBM Cúram Social Program Management supports SAML-based SSO. The example uses ISAM as an RPL-based SSO and outlines an SSO configuration for IBM Cúram Social Program Management that implements federated single sign-on by using the SAML 2.0 Browser POST profile. The example applies to both IdP-initiated and SP-initiated flows. Some additional steps are required to configure SP-initiated flows.

SSO configuration components

Figure 1 shows the components that are included in a Social Program Management SSO configuration.

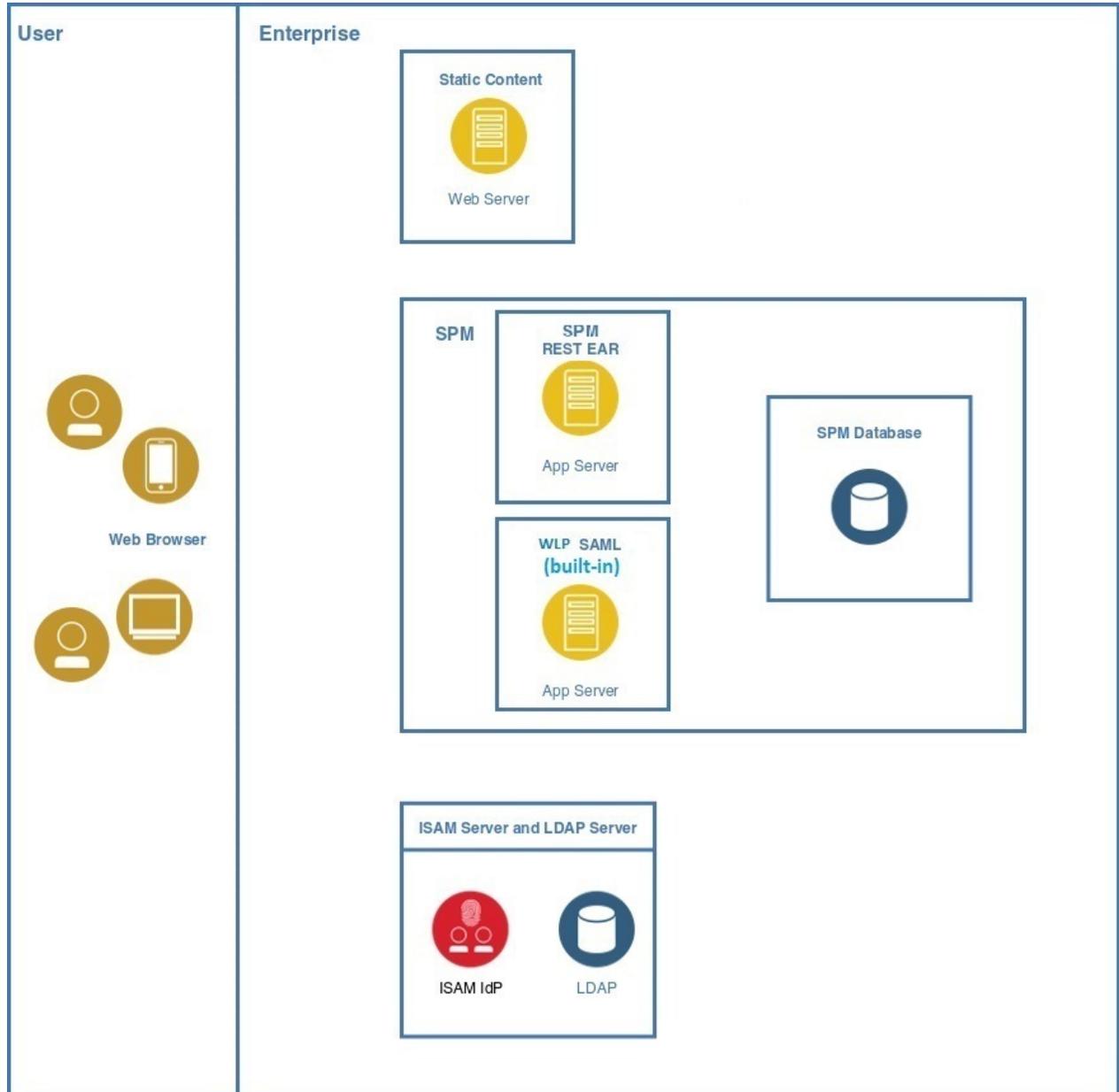


Figure 7. SSO configuration components

Web browser

A user sends requests from their web browser for applications in the SSO environment.

Web server

Social Program Management static content is deployed on a web server.

SAML-based SSO (ISAM) server

The IBM Security Access Manager server includes the identity provider (IdP).

LDAP server (user directory)

Among other items, the LDAP server contains the user name and password of all the valid users in the SSO environment.

WebSphere Application Server Liberty

Among other applications, WebSphere Liberty contains the deployed Social Program Management, Citizen WorkSpace, and REST enterprise applications.

Build-in WebSphere Liberty SAML configuration

Contains the features to run the SAML Trust Assertion Interceptor (TAI) and Consumer Service (ACS).

SPM Database

Data storage for the Social Program Management, Citizen WorkSpace, and REST enterprise applications.

Configuring single sign-on through IBM Security Access Manager

Use the IBM Security Access Manager management console to configure single sign-on (SSO) in IBM Cúram Social Program Management.

Before you begin

1. Start IBM Security Access Manager.
2. In the management console, log on as an administrator.
3. Accept the services agreement.
4. If required, change the administrative password.

About this task

In the IBM Security Access Manager management console, complete the following steps, with reference to [IBM Security Access Manager 9 Federation Cookbook](#).

Procedure

1. Configure the IBM Security Access Manager database:
 - a) In the top menu, click **Home Appliance Dashboard > Database Configuration**.
 - b) Enter the database configuration details, such as **Database Type, Address, Port**, and so on, and click **Save**.
 - c) When the **Deploy Pending Changes** window opens, click **Deploy**.
2. To install all the required product licenses, complete the steps in section *4.3 Product Activation* from [IBM Security Access Manager 9 Federation Cookbook](#)
3. Configure the LDAP SSL database by completing section *25.1.1 Load Federation Runtime SSL certificate into pdsrv trust store* from the [IBM Security Access Manager 9 Federation Cookbook](#) .
4. Configure the runtime component by completing *4.6 Configure ISAM Runtime Component on the Appliance* from the [IBM Security Access Manager 9 Federation Cookbook](#).

Configuring IBM Security Access Manager as an IdP

To configure IBM Security Access Manager as an identity provider (IdP), complete the outlined steps from the IBM Security Access Manager 9.0 Federation Cookbook that is available from IBM Developer Works.

Before you begin

Download the IBM Security Access Manager 9.0 Federation Cookbook from IBM Developer Works, as shown in the related link. Also, download the mapping files that are provided with the cookbook.

About this task

To set up the example environment, complete the specified sections in [IBM Security Access Manager 9 Federation Cookbook](#)

Procedure

1. Complete *Section 5, Create Reverse Proxy instance*.

2. Complete *Section 6, Create SAML 2.0 Identity Provider federation*.

In Section 6.1, if you are using the ISAM docker deployment, it is possible to reuse the existing keystore that is included in the container instead of creating a new keystore. It is important to reflect this change in subsequent sections where the myidpkeys certificate database is referenced.

3. Complete *Section 8.1, ISAM Configuration for the IdP*.

In Section 8.1, use the hostname of the IdP federation.

4. Optional: After you complete Section 8.1.1, if you require ACLs to be defined to allow and restrict access to the IdP junction, then follow the instructions in *Section 25.1.3, Configure ACL policy for IdP*.

5. Complete *Section 9.1, Configuring Partner for the IdP*.

The export from WebSphere Application Server Liberty does not contain all the relevant data.

Therefore, in Section 9.1, after you complete configuring partner for the IdP, you must click **Edit configuration** and complete the remaining advanced configuration.

Add and enable users in LDAP

Add the users from LDAP and enable them in SAML-based SSO.

Procedure

1. To create LDAP and IBM Security Access Manager runtime users, create an `ldif` file that can be used to populate OpenLDAP, as shown in the following sample:

```
# cat usersCreate_ISAM.ldif
dn: dc=watson-health,secAuthority=Default
objectclass: top
objectclass: domain
dc: watson-health

dn: c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: country
c: ie

dn: o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: organization
o: curam

dn: ou=curamint,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: organizationalUnit
ou: curamint

dn: cn=caseworker,ou=curamint,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: person
objectclass: inetOrgPerson
objectclass: top
objectclass: organizationalPerson
objectclass: ePerson
cn: caseworker
sn: caseworkersurname
uid: caseworker
mail: caseworker@curam.com
userpassword: PasswOrd

dn: ou=curamext,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: organizationalUnit
ou: curamext

dn: cn=jamesmith,ou=curamext,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: person
```

```
objectclass: inetOrgPerson
objectclass: top
objectclass: organizationalPerson
objectclass: ePerson
cn: jamesmith
sn: Smith
uid: jamesmith
mail: jamesmith@curamexternal.com
userpassword: Passw0rd
```

2. Add users to the OpenLDAP database:

- a) On the host server that is running the docker containers, enter the following command:

```
docker cp usersCreate_ISAM.ldif idpisam9040_isam-ldap_1:/tmp
```

- b) To log on to the OpenLDAP container, enter the following command:

```
docker exec -ti idpisam9040_isam-ldap_1 bash
```

- c) To add the users to OpenLDAP, enter the following command:

```
ldapadd -H ldaps://127.0.0.1:636 -D cn=root,secAuthority=default -f /tmp/
Curam_usersCreate_ISAM.ldif
```

3. Import the users into IBM Security Access Manager:

- a) To log on to the IBM Security Access Manager command-line interface, enter the following commands:

```
docker exec -ti idpisam9040_isam-webseal_1 isam_cli
isam_cli> isam admin
pdadmin> login -a sec_master -p <password>
```

- b) To import the users into IBM Security Access Manager, enter the following commands:

```
pdadmin sec_master> user import caseworker
cn=caseworker,ou=curamint,o=curam,c=ie,dc=watson-health,secAuthority=Default
pdadmin sec_master> user modify caseworker account-valid yes
pdadmin sec_master> user import jamesmith
cn=jamesmith,ou=curamext,o=curam,c=ie,dc=watson-health,secAuthority=Default
pdadmin sec_master> user modify jamesmith account-valid yes
```

4. To test the identity provider (IdP) flow, enter the following URL in a browser:

```
https://IdP_URL/isam/sps/saml20idp/saml20/
logininitial?RequestBinding=HTTPPost&PartnerId=ACS_URL/samlsp/acs
&NameIdFormat=Email&Target=WLP_hostname:WLP_port/Rest/v1
```

Replace the following values in the URL with the appropriate values for your configuration:

- *IdP_URL* is the IBM Security Access Manager login initial URL
- *ACS_URL* is the SAML Assertion Consumer Service URL
- *WLP_hostname* is the WebSphere Liberty application server host name
- *WLP_port* is the WebSphere Liberty application server port, where in IBM Cúram Social Program Management the default value is 9044

When the IBM Security Access Manager docker container starts, the IdP endpoints are initialized only when the first connection request is received. However, if the first connection request is triggered by Social Program Management, an XHR timeout occurs before the initialization finishes. Therefore, this test step is required to ensure that the initialization of the IdP endpoints is completed.

5. In a browser, go to the home page and log in.

Test IdP-initiated SAML SSO infrastructure

When the IBM Security Access Manager docker container starts, the IdP endpoints are initialized only when the first connection request is received. However, if the first connection request is triggered by IBM

Cúram Social Program Management, an XHR timeout occurs before the initialization finishes. This test step is required to ensure that the initialization of the IdP endpoints is completed.

Procedure

To test the identity provider (IdP) flow, enter the following URL in a browser:

```
https://<isam_url>/isam/sps/saml20idp/saml20/logininitial?
RequestBinding=HTTPPost&PartnerId=https://<wlp_url>/saml20/
acs&NameIdFormat=Email&Target=<wlp_url>/Rest/api/definitions
```

Where:

- `<isam_url>` - The URL for IBM Security Access Manager. It consists of the IBM Security Access Manager hostname, and port number, for example, `https:// 192.168.0.1:12443`.
- `<junction_name>` - The junction name that is used during the federation configuration in reverse proxy. The default value is `isam`.
- `<idp_endpoint>` - The endpoint that is configured for the IDP federation. The default value is `sps`.
- `<federation_name>` - The name that was used when you created the federation.
- `<wlp_url>` - The WebSphere Application Server Liberty hostname

SP-initiated only: Test SP-initiated SAML SSO infrastructure

Complete the following steps to test the SP-initiated SAML SSO infrastructure.

About this task

Open your browser, with network devtools, and load a protected IBM Cúram Social Program Management application URL like this example: `<SPM_Kubernetes_URL>/curamwhere` `<SPM_Kubernetes_URL>` is the SPM deployed in Kubernetes environment URL, for example `https://spm.dev.watson-health.ibm.com/curam` You are redirected to the ISAM SSO log-in page. Log in with the user credentials who is authorized to access SPM application You should be redirected to the SPM application after a successful authentication.

Procedure

1. Open your browser with network devtools, and load a protected IBM Cúram Social Program Management application URL, as shown in the following example:

```
https://application-domain.com
```

You are redirected to the ISAM SSO log-on page.

2. Log on with the credentials of a user who is authorized to access the Social Program Management application.

After a successful authentication, you are redirected to the Social Program Management application.

Building EAR files

IBM Cúram Social Program Management is composed of several applications that you must build into EAR files before deployment. These application EAR files incorporate client and server components.

Building the Cúram EAR files

Build the Cúram EAR files by using the Ant **libertyEAR** target. You must also enable static content in SPM. For more information, see .

Before you run this target, a fully built application must be available. For more information, see .

The Ant **libertyEAR** target takes the output from the previously built application, such as the generated Java classes that represent the model, deployment descriptors, and packages them up into EAR files.

This target creates installable EAR files in the following directory:

```
$SERVER_DIR/build/ear/WLP/$SERVER_MODEL_NAME.ear
```

The environment variables `$SERVER_DIR` and `$SERVER_MODEL_NAME` specify the name of root directory of the project and the model in the project.

Run the Ant **libertyEAR** target from the root directory of the server project to build the application EAR files for WebSphere Application Server Liberty:

```
./build.sh libertyEAR
```

The EAR files include the following structure and contents:

- META-INF Directory:
 - `application.xml`: A generated file that lists the mapping of EJB modules to JAR files that are in the application.
 - `ibm-application-bnd.xmi`: A generated Liberty-specific extension file.
 - `MANIFEST.MF`: A manifest file that details the contents of the EAR files.
- JAR files: `Curam.ear/lib` contains Cúram-specific JAR files, including `application.jar`, `codetable.jar`, `events.jar`, `struct.jar`, `messages.jar`, `implementation.jar`, and `properties.jar`. The `properties.jar` file contains the `Bootstrap.properties` file.

Building an EAR file that contains either the web application or the server application

The Ant **libertyEAR** target builds EAR files that contain both the web client and application components. Alternatively, you can build EAR files that contain only the web client or only the server components, which can support alternative topologies where the web client and server applications are installed on separate servers. For example, to support secure access to the Social Program Management application for external users, a new web client application might be developed. This web application might be deployed on its own WebSphere Liberty server and use existing Social Program Management server application components that are deployed on a different WebSphere Liberty server. For more information on splitting EAR file components, see [Multiple EAR files](#).

Use the following Ant **libertyEAR** to build EAR files that contain only the web client application:

```
./build.sh libertyEAR -Dclient.only=true
```

Use following Ant **libertyEAR** to build EAR files that contain only the server applications:

```
./build.sh libertyEAR -Dserver.only=true
```

Server code split

As part of the migration to WebSphere Application Server Liberty and container enablement, splitting the server and client code within the Cúram EAR file provides an optimum and flexible deployment model for Kubernetes.

The server code is common across all EAR files. However, the client code is different depending on the EAR file. The main benefit of splitting the server code EAR file in containers is that it is reused across all client components such as the main client EAR file, the Citizen Portal EAR file.

The benefit of a single `CuramServerCode.ear`, is that it can be deployed in a container, with any other EAR file without the need to build all EAR files with the server and client code, thus reducing both build and deployment times. This also increases deployment model flexibility. Also, separating the client and server code helps in the mitigation of thread pool starvation in WebSphere Liberty deployments. The new architecture involved the separation of JMS processing.

The Application EAR responsible for processing JMS-initiated transactions is called the JMS Consumer and consumes JMS messages through EJB MDBs enabled. The ability to split and deploy the server and

client is pre-existing when you deploy to traditional IBM WebSphere Application Server or WebLogic Server clusters and is expanded to include the Cúram EAR file. When the Cúram EAR is split, it contains client-only code, and it must be packaged with the `CuramServerCode.ear`, as must all additional EAR files, for example Citizen Portal EAR, and Rest EAR. To package with `CuramServerCode.ear`, take the following steps:

1. Modify `deployment_packaging.xml` in the following location `$SERVER_DIR/project/properties/`
2. Set `requireServer="false"` for `Curam.ear`
3. Build WebSphere Liberty as normal
4. Run the following Ant **libertyEAR** target:

```
build.sh libertyEAR -Dserver.only=true -Dear.name=CuramServerCode -
DSERVER_MODEL_NAME=CuramServerCode
-Dcuram.ejbserver.app.name=CuramServerCode
```

Building the web services application EAR file

Build the Cúram web services EAR file by using the Ant **libertyWebServices** target.

Before you run the Ant **libertyWebServices** target, a fully built IBM Cúram Social Program Management application must be available.

The **libertyWebServices** target takes the previously generated Java files and deployment descriptors and packages them into a ready to install EAR file in the following directory:

```
$SERVER_DIR/build/ear/WLP/${SERVER_MODEL_NAME}WebServices.ear
```

The environment variables `$SERVER_DIR` and `$SERVER_MODEL_NAME` specify the name of root directory of the project and the model in the project.

Run the Ant **libertyWebServices** target from the `$SERVER_DIR` directory of the project to build the web services EAR file:

```
./build.sh libertyWebServices
```

Java files and deployment descriptors are generated during the build process based on the *web service components* that are defined in the model. For more information, see [Building and configuring a Cúram application](#). BPO classes are mapped to server components with a stereotype of web service for this generation to occur. Any server component with a stereotype of web service is treated as if it also had a stereotype of `ejb` because web service interfaces are wrappers on publicly available BPOs. For more information, see [Business Process Objects](#) for details on assigning BPOs to server components.

When deployed, IBM Cúram Social Program Management web services expose their own WSDL. For example, if there is a web service, `MyTestService`, the WSDL can be derived by using a URL of this format: `http://localhost:10102/CuramWS/services/MyTestService?wsdl`

The general URL format for starting a Cúram web service from a web service client such as SoapUI is as follows:

```
http://<web-server>:<port-number>/<ServerModelName>WS/services/<BPO-name>
```

Deploying applications

Deploy the packaged IBM Cúram Social Program Management application and web services application in EAR files to the application server.

Targets for installing and uninstalling applications

The **installapp** and **uninstallapp** targets install and uninstall applications on WebSphere Liberty. The **installapp** and **uninstallapp** targets need the `AppServer.properties` file to be configured correctly. For more information see [“Managing deployment properties” on page 11](#).

Install applications

Use the Ant **installapp** target to install an application EAR file. **installapp** requires the following options:

- `-Dserver.name`
The name of the server to install the application on.
- `-Dear.file`
The fully qualified name of the EAR file to install.
- `-Dapplication.name`
The name of the application.

An example command is as follows:

```
./build.sh installapp -Dserver.name=CuramServer -Dear.file=$SERVER_DIR/build/ear/WLP/Curam.ear -Dapplication.name=Curam
```

Note: For client-only EAR files, there must be a corresponding EAR file with the server module in the environment.

You must restart the server after you install the application.

Note: The Ant **libertyEAR** and **libertyWebServices** targets create several application EAR files in the `$SERVER_DIR/build/ear/WLP` folder. Apply the property specifications, `-Dear.file=` and `-Dapplication.name=`, as is appropriate for the applications relevant to your environment.

Uninstall applications

Use the Ant **uninstallapp** target to uninstall an application by using the following options:

- `-Dserver.name`
The name of the server the application is installed on.
- `-Dapplication.name`
The name of the application to uninstall.

An example command is as follows:

```
./build.sh uninstallapp -Dserver.name=CuramServer -Dapplication.name=Curam
```

The Ant **uninstallapp** target stops the WebSphere Liberty server, so you must start it after you run the target.

Starting and stopping WebSphere Liberty

To start a server, enter the following command:

```
./build.sh startserver -Dserver.name=CuramServer
```

To stop a server, enter the following command:

```
./build.sh stopserver -Dserver.name=CuramServer
```

Testing the deployment by logging in to the application

When the application is deployed, log in to display the application landing page to verify the basic functions of the application.

Ensure that the relevant server is started and enter the application URL in a web browser, for example:

```
https://<some.machine.com>:<port>/<context-root>
```

To obtain the application URL, search the WebSphere Liberty logs for the CWWKT0016I message that identifies the application of interest. For example, enter the following command:

```
grep CWWKT0016I $WLP_HOME/usr/servers/CuramServer/logs/console.log
```

For the Cúram application, the command returns:

```
[AUDIT ] CWWKT0016I: Web application available (client_host): https://your.hostname.com:10101/  
Curam/
```

Use the returned URL to access your application. When the application is deployed and the server is started, log in to display the application landing page to verify the basic functions of the application.

Some applications, like the CitizenPortal context root, don't require an explicit login and the landing page is entered directly. For more information about CitizenPortal, see [IBM Cúram Universal Access](#).

Debugging WebSphere Liberty

Use resources such as the messages .log file to monitor and debug Social Program Management applications.

WebSphere Liberty logs

WebSphere Liberty logs are in the \$WLP_HOME/usr/server/<server_name>/logs directory. The following list outlines the most important logs:

messages.log

Equivalent to SystemOut.log and SystemErr.log in traditional WebSphere Application Server.

trace.log

Detailed WebSphere Liberty trace data is logged here. For more information, see [Set up trace and get a full dump for WebSphere Liberty](#).

./ffdc

Similar content to traditional WebSphere Application Server.

Remote debugging applications in WebSphere Liberty

Use the following .xml file to control logging behavior:

```
$WLP_HOME/usr/server/<server_name>/adc_conf/server_logging.xml
```

Perform remote debugging with Eclipse by specifying the normal options in jvm.options, for example:

```
-Xrunjdwp:transport=dt_socket,address=8787,server=y
```

You must restart the server by using the WebSphere Liberty server debug command, for example:

```
server debug CuramServer
```

You must specify the appropriate Eclipse Remote Java Applications configuration and breakpoints, or the equivalent for your debugging environment.

Reviewing Java Management Extensions (JMX) statistics

The statistics that are generated by the JMX infrastructure can help you to review and debug application performance. For more information, see

[JMX](#) and [Developing with Cúram JMX](#)

Known issues and limitations

Some known issues and limitations can occur when you deploy WebSphere Liberty. Where possible, workarounds are provided.

WebSphere Liberty on premises

IBM Cúram Social Program Management does not support deployments to WebSphere Liberty collectives.

Authentication alias warning: J2CA8050I

When you start a WebSphere Liberty server, the following warning occurs:

```
J2CA8050I: An authentication alias should be used instead of defining a user name and password on dataSource[curamdb]
```

In WebSphere Liberty, the authentication aliases can be used for container-managed and XA recovery data sources. IBM Cúram Social Program Management uses component-managed data sources extensively, so this warning message can be safely ignored.

The logs can be filled by repetitions of the ICWWKS4001I message

The following log extract shows an example of the ICWWKS4001I message:

```
[1/22/19 8:48:18:272 GMT] 000000ba com.ibm.ws.security.token.internal.TokenManagerImpl
ICWWKS4001I: The security token cannot be validated. This can be for the following reasons
1. The security token was generated on another server using different keys.
2. The token configuration or the security keys of the token service which created the token
has been changed.
3. The token service which created the token is no longer available.
```

The root cause is users not clearing the browser cache after the application is redeployed. Users might have old, local cookie files. However, after a redeployment or an upgrade, the application does not recognize the cookies that are presented to it by the machine, which causes the error messages in the logs.

The solution is to ensure that all users clear their browser caches.

Alternatively, add the message CWWKS4001I to the WebSphere Liberty ignore list by editing `$WLP_HOME/usr/server/<server_name>/bootstrap.properties` and adding the line:

```
com.ibm.ws.logging.hideMessage=CWWKS4001I
```

For more information, see [Specifying Liberty bootstrap properties](#).

WebSphere Liberty dropins folder

The WebSphere Liberty dropins folder cannot be used because it is incompatible with the Social Program Management application EAR files.

Notices

This information was developed for products and services offered in the United States.

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, MD-NC119 Armonk, NY 10504-1785 US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing Legal and Intellectual Property Law IBM Japan Ltd. 19-21, Nihonbashi-Hakozakicho, Chuo-ku Tokyo 103-8510, Japan

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 jurisdictions 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 provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing IBM Corporation North Castle Drive, MD-NC119 Armonk, NY 10504-1785 US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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 actual people or business enterprises 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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Privacy Policy considerations

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies or other similar technologies that collect each user's name, user name, password, and/or other personally identifiable information for purposes of session management, authentication, enhanced user usability, single sign-on configuration and/or other usage tracking and/or functional purposes. These cookies or other similar technologies cannot be disabled.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at <http://www.ibm.com/privacy> and IBM's Online Privacy Statement at <http://www.ibm.com/privacy/details> the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at <http://www.ibm.com/software/info/product-privacy>.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at <http://www.ibm.com/legal/copytrade.shtml>.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

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

The registered trademark Linux is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

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

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

Other names may be trademarks of their respective owners. Other company, product, and service names may be trademarks or service marks of others.



Part Number:

(1P) P/N: