Cúram 8.1.2

Cúram on Kubernetes

Note

Before using this information and the product it supports, read the information in Notices on page 39

Edition

This edition applies to Cúram 8.1, 8.1.1, and 8.1.2.

Contents

1 Cúram on Kubernetes

You can build Cúram 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 Cúram on Kubernetes Runbook provides instructions about how to containerize Cúram 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, Cúram has been enhanced to support the technologies that are described in the following list from version 7.0.10.0.

WebSphere® Application Server Liberty

Cúram 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

Cúram 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®

Cúram supports Docker® for packaging Cúram 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 a Cúram 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.

Cúram supports Helm for deploying Cúram 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.

Cúram 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. Cúram 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.

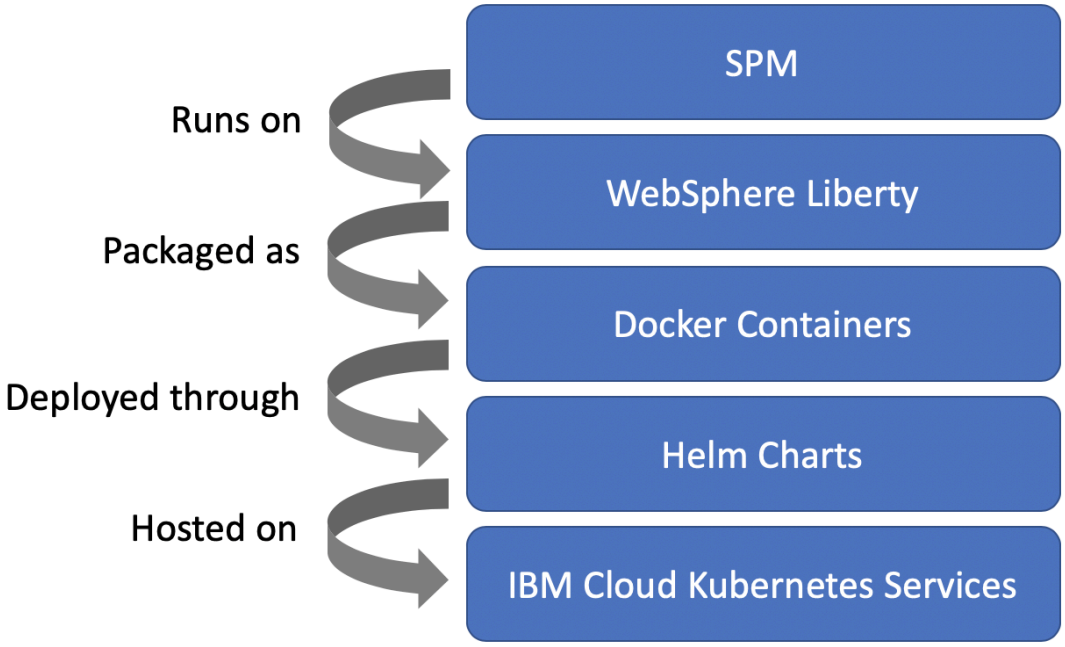
Up next...

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

2 Kubernetes architecture

Cúram was enhanced in version 7.0.10.0 to enable it for deployment into cloud native hosting platforms. While previously Cúram 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.

Cúram 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.



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 Cúram application, see the Cúram on Kubernetes Runbook.

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

2.1 Transaction isolation

In relation to Cúram, 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.

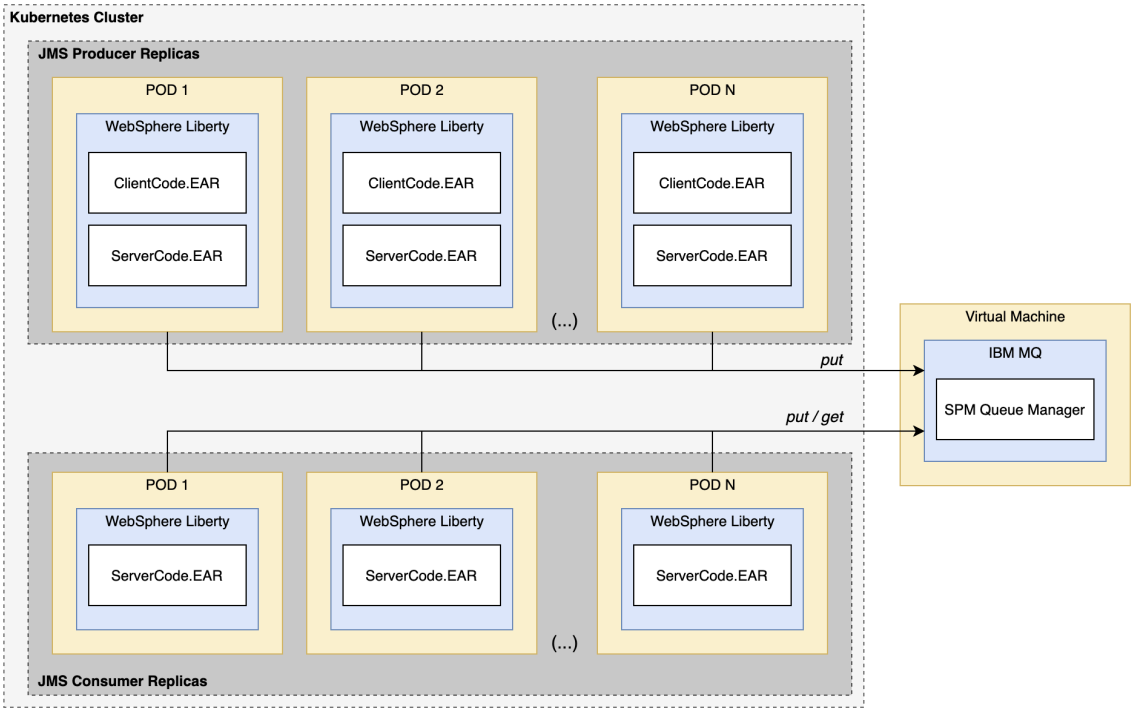
The previous differences introduced a risk of thread exhaustion in the runtime application. To mitigate against the risk, the following multi-faceted solution was developed:

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 `SIBJMSRAThreadPool` 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 inWebSphere® 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:



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

- **Queue manager**
  Cúram allows only one dedicated queue manager per set of `JMS Producer` replicas. Also, Cúram allows only one set of `JMS Consumer` replicas per queue manager. Finally, Cúram allows only one queue manager per Cúram application.
- **JMS message processing**
  `JMS Producer` does not process any JMS message except the cache invalidation messages. The `JMS Consumer` does not have the Cúram web interface available. After Cúram 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 3.6 Building EAR files on page 31.

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 Cúram. 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

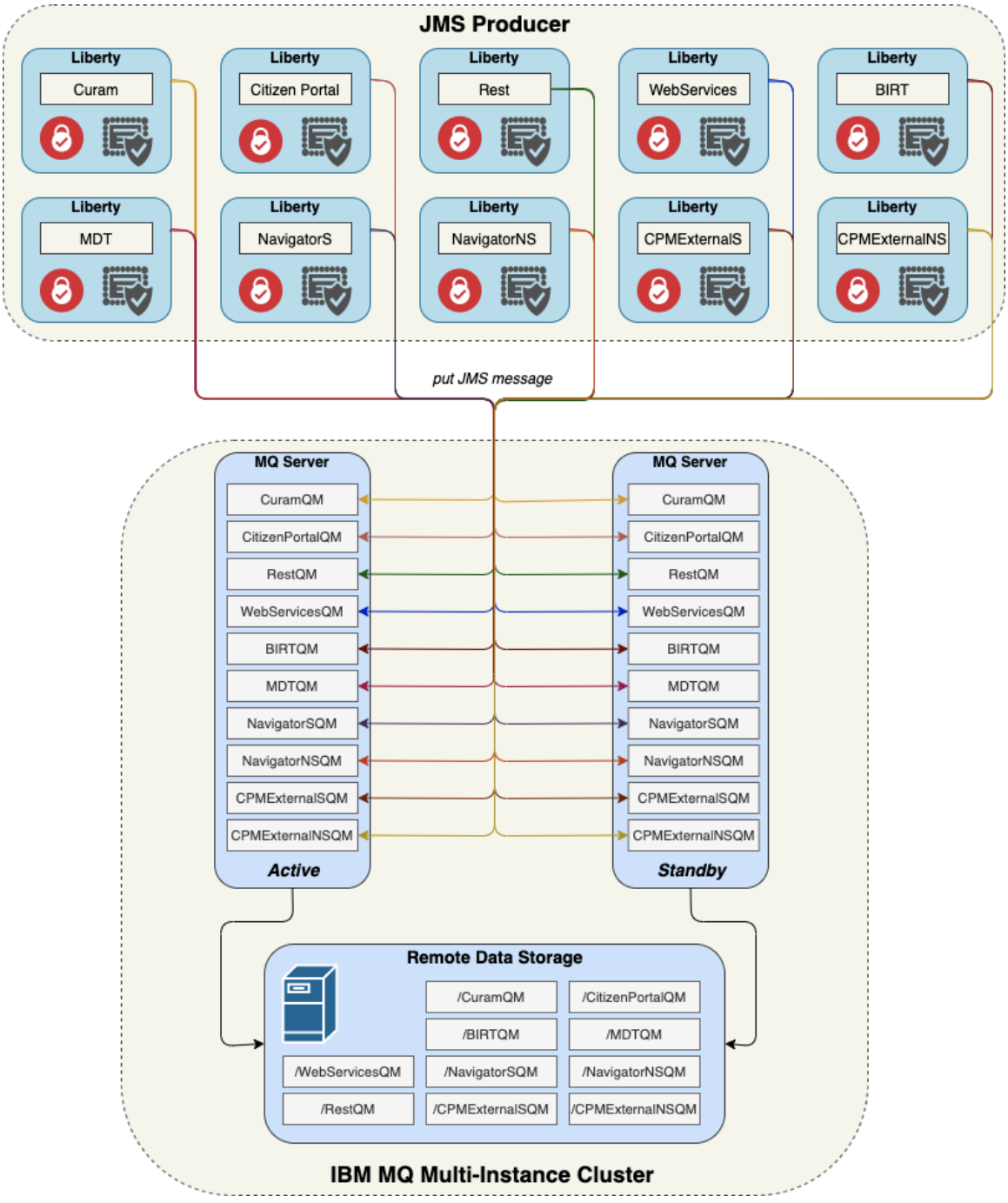
Performance tuning

2.2 Messaging architecture

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

Each Cúram 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 Cúram `JMS Producer`, see the Transaction Isolation topic.

Note: Cúram supports IBM® MQ LTS on VMs only. IBM® MQ CD is supported only as a container on Red Hat® OpenShift®. IBM® MQ on Kubernetes, IBM® MQ as a service, and other message engines have not been verified with Cúram.

Multi-instance queue manager support

For IBM® MQ Cluster, Cúram 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 Cúram

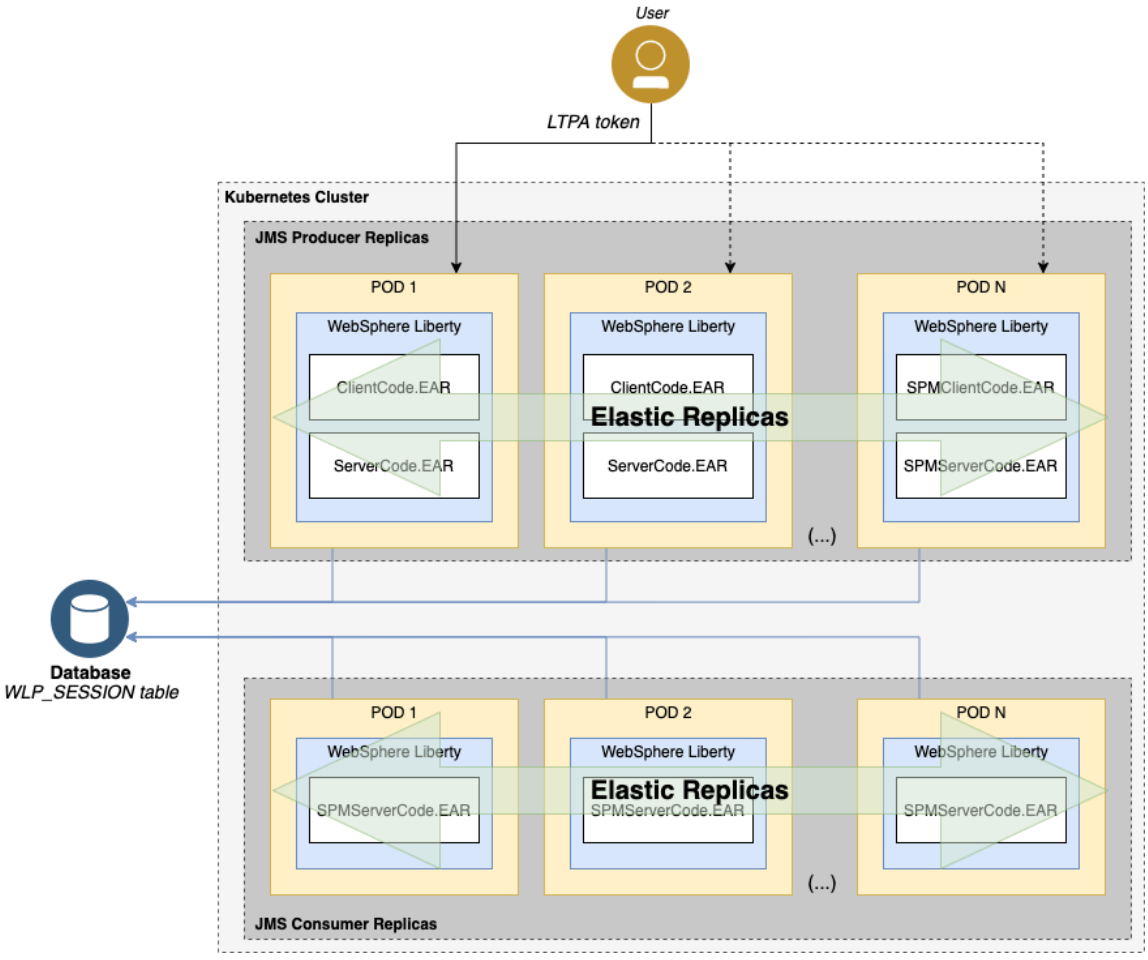
The cardinality between queue managers and IBM® MQ servers is flexible. You can configure all queue managers on one IBM® MQ cluster, or you can configure one queue manager per IBM® MQ cluster. For example, you can configure some queue managers for an internal application in one IBM® MQ cluster, and some 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.

2.3 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 Cúram 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. Figure 1 illustrates the architecture:



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

WebSphere® Liberty collectives support

Cúram does not support WebSphere® Application Server Liberty collectives on Kubernetes.

Related concepts

Configuring security on page 28

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.

3 Deploying on WebSphere® Application Server Liberty

Because you can now deploy and run Cúram on WebSphere® Liberty, you can benefit by deploying Cúram on stand-alone 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.

3.1 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 Cúram, you must install and configure the following software on one of the supported platforms, see the Cúram Supported Prerequisites:

- Install Cúram. For more information, see Installing a development environment.
- WebSphere® Application Server Liberty. For more information, see WebSphere® Application Server 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, see Installing and configuring Apache Ant. For version information, see the Cúram Supported Prerequisites.
- For more information on minimum hardware requirements, see Hardware requirements for Cúram 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 Cúram) 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*

3.2 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 Cúram 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=XOVRjjVTebM8gV953LGMLQ==

# 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.

3.3 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 3.2 Managing deployment properties on page 22. Then, customize the server.xml and the files that it includes to reflect your implementation of Cúram.

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 Cúram 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 *$WLP_HOME/usr/servers/CuramServer/server.xml* file and the files that it includes so that they contain WebSphere® Liberty features and configurations to support Cúram. The configuration files are placed in *$WLP_HOME/usr/servers/CuramServer/adc_conf/*. Table 1 lists all the changed files and describes the role of each file.

Table 1: List of XML files and descriptions

List of XML configurations and descriptions for each.

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 Cúram.
server_resources_tx.xml	Specifies WebSphere® Liberty transaction settings.

XML file	Description
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 3.7 Deploying applications on page 34:	
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 product documentation 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 3.2 Managing deployment properties on page 22.
- 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 Cúram.

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
```

3.4 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 Cúram deployed on WebSphere® Liberty. Also, for information about how to configure the web server's HTTP verb permissions to mitigate verb tampering, see Enabling HTTP verb permissions.

The Ant **configurewebserverplugin** 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.

Table 2: Property overrides and their default values

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

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.

3.5 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 Cúram configured *adc_conf/server_security.xml* file and consists of the following elements:

- The configuration for the Cúram system login module, *CuramLoginModule*, is defined by various elements and this 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 Cúram 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 Cúram 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 Cúram security requirements such as the system login module. For more information see, Securing Liberty and its applications.

Related concepts

Elasticity on page 18
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 an application user.

Note: The security.username user is automatically added to the WebSphere® Liberty basicRegistry by the Social Program Management-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 Cúram 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 Cúram 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 Cúram 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 administration system

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

1. Log in to Cúram 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 (SSO)

Federated SSO 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 Cúram application.

For more information on configuring SSO, see .

3.6 Building EAR files

Cúram 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 application EAR files by using the Ant **`libertyEAR`** target.

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 **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 Cúram 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 Cúram 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 command to build EAR files that contain only the web client application:

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

Use the following command 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 command:

```
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 Cúram 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, Cúram 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>`*

3.7 Deploying applications

Deploy the packaged Cúram 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 3.2 Managing deployment properties on page 22.

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
```

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: If you are installing BIRT, add the *biapp.configure.birtviewer* attribute to the **`installapp`** target.

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
```

3.8 Pre-compiling JSPs

During deployment, use the Ant **precompilejsp** target to pre-compile the JSPs of a client EAR before installing the EAR file. Pre-compiling the JSPs before installation speeds up the display of a page in the web browser the first time it is accessed.

The options for the **precompilejsp** target are:

- -Dear.file

  The fully qualified name of the EAR file to be pre-compiled.

```
        ./build.sh precompilejsp -Dear.file=$SERVER_DIR/build/
ear/WLP/Curam.ear
```

Figure 5: Example of Usage

3.9 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 Citizen Portal, see Universal Access.

3.10 Debugging WebSphere® Liberty

Use resources such as the *messages.log* file to monitor and debug Cúram 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

3.11 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**
  Cúram 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. Cúram 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 Cúram application EAR files.

Notices

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the Merative website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of Merative

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of Merative.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

Merative reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by Merative, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

MERATIVE MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Merative or its licensors may have patents or pending patent applications covering subject matter described in this document. The furnishing of this documentation does not grant you any license to these patents.

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

Any references in this information to non-Merative 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 Merative product and use of those websites is at your own risk.

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.

The licensed program described in this document and all licensed material available for it are provided by Merative under terms of the Merative Client Agreement.

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 Merative, 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. Merative, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. Merative shall not be liable for any damages arising out of your use of the sample programs.

Privacy policy

The Merative privacy policy is available at https://www.merative.com/privacy.

Trademarks

Merative ™ and the Merative ™ logo are trademarks of Merative US L.P. in the United States and other countries.

IBM®, the IBM® logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

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.

Oracle and Java are 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™, 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 company, product, and service names may be trademarks or service marks of others.