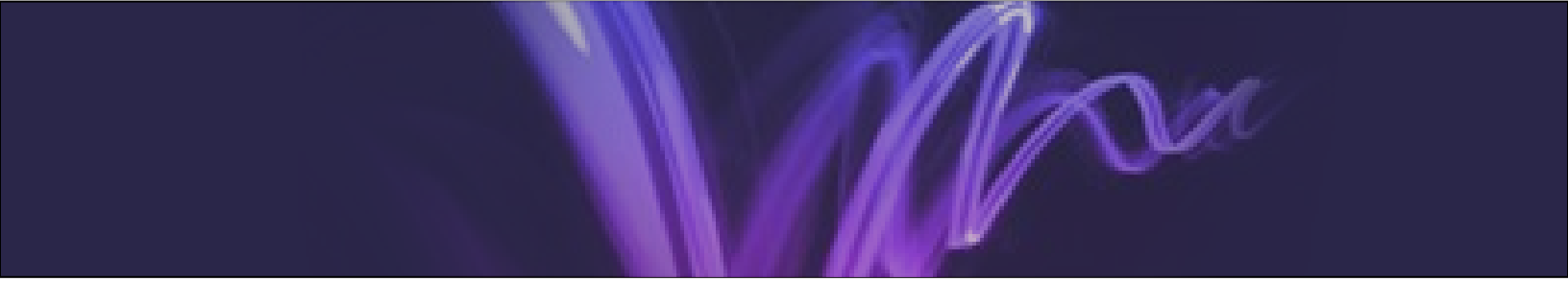


Build and Administer Your Own Liberty Application Cluster, new in WebSphere Application Server 8.5.5

Lab Instructions

Authors:
Michael C Thompson, WebSphere Developer, mcthomps@us.ibm.com
Chris Vignola, WebSphere Architecture, cvignola@us.ibm.com

TABLE OF CONTENTS

1 Objective

In this lab, you learn:

- The concepts and operations of a Liberty collective and clustering with the WebSphere Application Server Liberty profile.
- Hands-on experience creating, configuring and performing operations on a collective and cluster.
- Hands-on experience with the Jython scripting support.

2 Prerequisite knowledge

- Basic Windows or Linux knowledge – the instructions are written for Linux. Commands and variable references must be converted for Windows instructions.
- This lab uses gedit as the editor of choice in the command examples. You are free to use any editor or browser you wish.
- The server names used in these instructions were chosen to be illustrative of their purposes. The only reserved or special server names for the Liberty profile is “defaultServer”, which is the default target name for the bin/server command.
- The user names, user passwords and keystore passwords used in this lab are chosen to be illustrative and it is recommended that the values be used for the purpose of following the lab, as they are re-used in the provided command instructions and sample scripts. There are no default user names, user passwords or keystore passwords shipped with the Liberty profile.

3 Lab environment setup

3.1 Environment requirements

- ✔ Java 1.6
- ✔ WebSphere Application Server Liberty profile Network Deployment V8.5.5
- ✔ Jython-2.5.3

3.2 Environment setup

1. Install a Java 1.6 runtime if necessary

2. Install WebSphere Application Server Liberty Network Deployment V8.5.5
   See wasdev.net or this lab's article for installation instructions.

2a. Set the WLP_INSTALL_DIR to the Liberty profile installation directory.
   For example:
   On Windows: set WLP_INSTALL_DIR=C:\wlp
   On Linux: export WLP_INSTALL_DIR=~/wlp

3. Install jython-2.5.3

3a. Download the jython-installer from jython.org

3b. Install jython using the jython-installer.

3c. Add jython to your system path:
   For example:
   On Windows: set PATH=%PATH%;C:\jython2.5.3\bin
   On Linux/Unix: export PATH=$PATH:~/jython2.5.3/bin

4. Obtain the lab materials from wasdev.net

4a. Extract lab-materials.zip to C:\lab-materials or ~/lab-materials.
   Set the environment variable LAB_MATERIALS to the extracted contents.
   For example:
   On Windows: set LAB_MATERIALS=C:\lab-materials
   On Linux: export LAB_MATERIALS=~/lab-materials

5. File transfer and server commands require remote execution and access (RXA). Before running the sample scripts, ensure the target system(s) are configured for remote access. For more information on the remote execution and access requirements and setup, refer to "Requirements for using Remote Execution and Access (RXA)" in the information center.

4 Step-by-Step Instructions

4.1 Step 1: Introduction to Liberty

These steps take you through the most basic operations supported by the Liberty profile. This entails starting and stopping the default server, changing server configuration dynamically and installing an application.

1.1 Start the default server.

```
# cd $WLP_INSTALL_DIR
# bin/server start
```

1.2 Add the sample application "snoop" by adding the war file to the dropins directory. The application will be automatically installed and started.

```
# cp $LAB_MATERIALS/applications/snoop.war
     usr/servers/defaultServer/dropins
```

1.3 Access the sample application "snoop".

Go to the URL http://localhost:9080/snoop

1.4 Change the default HTTP port for the server.

```
# gedit usr/servers/defaultServer/server.xml
```

Change the HTTP port from 9080 to 8080 (modify the XML):

```
 <httpEndpoint id="defaultHttpEndpoint" httpPort="8080" />
```

1.5 Use the new port to access the application (no restart required).

Go to URL http://localhost:8080/snoop

1.6 View the console.log to see messages for the web application.

```
# gedit usr/servers/defaultServer/logs/console.log
```

1.7 Stop the default server.

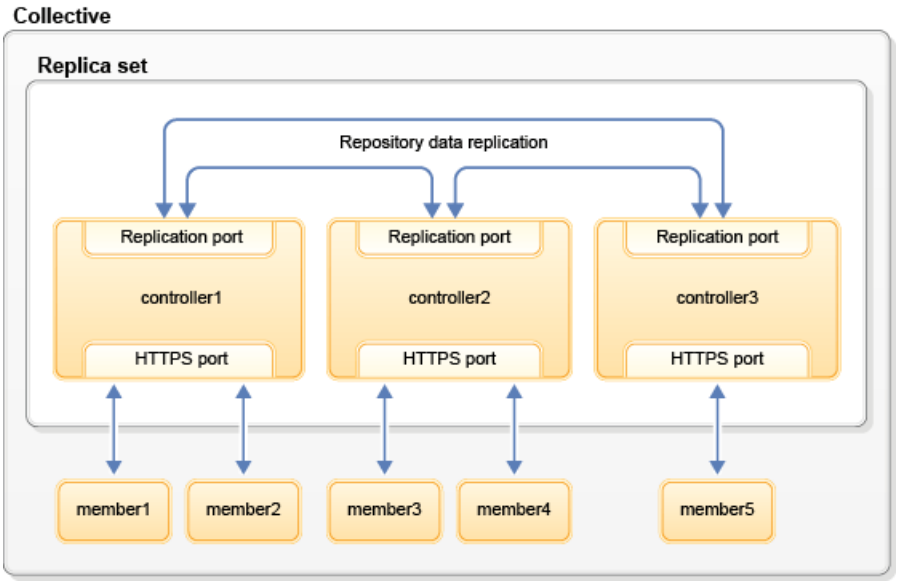
```
# bin/server stop
```

You now have experience with basic server operations, dynamic configuration and installing an application!

4.2 Step 2: Create a collective

These steps take you through the configuration of a basic collective. A collective is the set of Liberty servers in a single administrative domain. A collective consists of at least one “collective controller”, a server with the collectiveController-1.0 feature enabled. Optionally, a collective may have many "collective members", servers with the collectiveMember-1.0 feature enabled. A collective may be configured to have many collective controllers, called a replica set. Configuration of the replica set is done in Section 4.5 Building out a highly available controller topology.

Example topology diagram of a collective



2.1 Create a server which acts as the collective controller.

```
# bin/server create controller1
```

2.2 Create the collective controller configuration. This will establish the administrative domain security configuration.

NOTE: You may see "Error determining the host name.". This is not a problem; the host's network configuration may be incorrect or incomplete and the script was not able to detect the Fully Qualified Domain Name (FQDN) for the host. You can specify --hostName=myhost.mydomain.com to specify a desired FQDN.

```
# bin/collective create controller1 --keystorePassword=password
```

2.3 Update the server.xml with the XML output from the command (copy & paste). Be sure to set the user and password in <quickStartSecurity>: <quickStartSecurity userName="**admin**" userPassword="**adminpwd**" />

```
# gedit usr/servers/controller1/server.xml
```


The updated server.xml for controller1 should look like this:

```
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>jsp-2.2</feature>
    </featureManager>

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

    <featureManager>
        <feature>collectiveController-1.0</feature>
    </featureManager>

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

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

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

    <!-- inbound (HTTPS) keystore -->
    <keyStore id="defaultKeyStore" password="{xor}Lz4sLCgwLTs="
              location="${server.config.dir}/resources/security/key.jks" />

    <!-- inbound (HTTPS) truststore -->
    <keyStore id="defaultTrustStore" password="{xor}Lz4sLCgwLTs="
              location="${server.config.dir}/resources/security/trust.jks" />

    <!-- server identity keystore -->
    <keyStore id="serverIdentity" password="{xor}Lz4sLCgwLTs="
              location="${server.config.dir}/resources/collective/serverIdentity.jks" />

    <!-- collective trust keystore -->
    <keyStore id="collectiveTrust" password="{xor}Lz4sLCgwLTs="
              location="${server.config.dir}/resources/collective/collectiveTrust.jks" />

    <!-- collective root signers keystore -->
    <keyStore id="collectiveRootKeys" password="{xor}Lz4sLCgwLTs="
              location="${server.config.dir}/resources/collective/rootKeys.jks" />

</server>
```

2.4 Start the collective controller.

```
# bin/server start controller1
```

2.5 Verify the server started correctly and is ready to receive members.

```
# gedit usr/servers/controller1/logs/messages.log
```

Look for the following message:

```
CWWKX9003I: CollectiveRegistration MBean is available.
```


2.6 Create a member to join the collective.

```
# bin/server create member1
```

2.7 Join the member to the collective.
You will need to accept the SSL certificate presented by the controller.

```
# bin/collective join member1 --host=localhost --port=9443
--user=admin --password=adminpwd --keystorePassword=member1
```

2.8 Update the server.xml with the output XML from the command.
Update the HTTP and HTTPS port for the server. The default ports will already be in use by the controller. Manually update the port numbers in the configuration. Use ports numbers: http=9081, https=9444

```
# gedit usr/servers/member1/server.xml
```

The updated server.xml for member1 should look like this:

```
<server description="new server">
    <!-- Enable features -->
    <featureManager>
        <feature>jsp-2.2</feature>
    </featureManager>

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

    <featureManager>
        <feature>collectiveMember-1.0</feature>
    </featureManager>

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

    <!-- Connection to the collective controller -->
    <collectiveMember controllerHost="localhost"
                      controllerPort="9443" />

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

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

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

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

    <!-- collective truststore -->
    <keyStore id="collectiveTrust" password="{xor}MjoyPTotbg=="
              location="${server.config.dir}/resources/collective/collectiveTrust.jks" />
</server>
```


2.9 Start the collective member.

```
# bin/server start member1
```

2.10 Verify the server started correctly and is publishing information to the controller.

```
# gedit usr/servers/member1/logs/messages.log
```

Look for the following messages (in any order):

```
CWWKX8112I: The server's host information was successfully
published to the collective repository.
CWWKX8114I: The server's paths were successfully published to the
collective repository.
CWWKX8116I: The server STARTED state was successfully published
to the collective repository.
```

You now have a basic collective topology created. In this topology, member1 is a collective member and controller1 is a collective controller. All collective members publish information about themselves to their collective controller. This published information is available for query directly from the controller without need of forwarding the request down to each collective member.

4.3 Step 3: Perform collective operations

In this section, you use the sample Jython scripts available from wasdev.net to perform MBean operations on the collective controller which allow you to start and stop registered collective members. Direct links to the sample scripts are provided. The sample script downloads are zip files which include the required jython scripts necessary to run. These instructions assume the files are unzipped into the directory $LAB_MATERIALS/sample_scripts

3.1 In order to run the sample scripts, set the CLASSPATH and JYTHONPATH environment variables. Jython uses these variables to locate the required libraries, such as the restConnector.jar and sample script libraries.

```
# export CLASSPATH=$WLP_INSTALL_DIR/clients/restConnector.jar

# export JYTHONPATH=$WLP_INSTALL_DIR/clients/jython/:
$LAB_MATERIALS/sample_scripts/lib
```

For versions of jython earlier than 2.5, it may be necessary to invoke jython with the command argument -Dpython.path=$JYTHONPATH as it does not support the JYTHONPATH environment variable.

3.2 Stop the collective member by invoking the stopServer.py script.
An example invocation (change the serverHost value to be your host name):

```
# jython $LAB_MATERIALS/sample_scripts/stopServer.py
--serverHost=myhost.com --serverUsrdir=$WLP_INSTALL_DIR/usr
--serverName=member1 --host=localhost --port=9443 --user=admin
--password=adminpwd
--truststore=usr/servers/controller1/resources/security/trust.jks
--truststorePassword=password
```

3.3 Verify the server stopped.

```
# bin/server status member1
```

3.3 Start the collective member by invoking the startServer.py script.

```
# jython $LAB_MATERIALS/sample_scripts/startServer.py
--serverHost=myhost.com --serverUsrdir=$WLP_INSTALL_DIR/usr
--serverName=member1 --host=localhost --port=9443 --user=admin
--password=adminpwd
--truststore=usr/servers/controller1/resources/security/trust.jks
--truststorePassword=password
```

3.4 Verify the server has been started.

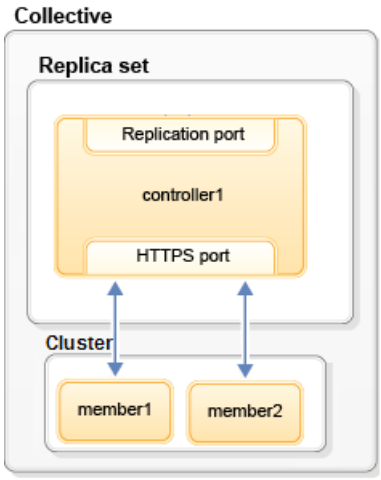
```
# bin/server status member1
```

You now have experience executing the sample administrative Jython scripts available from wasdev.net and have used the MBeans available on the collective controller to perform operations against collective members.

4.4 Step 4: Create a cluster

In this section, you expand the collective by adding a new member, configuring both member servers to join the default cluster, and use Jython scripting to perform MBean operations on the collective controller which allow you to start and stop entire clusters.

Topology diagram of a collective created in this step



1.

4.1 Create a second member to join the collective.

```
# bin/server create member2
```

4.2 Join the member to the collective.
You will need to accept the SSL certificate presented by the controller.

```
# bin/collective join member2 --host=localhost --port=9443
--user=admin --password=adminpwd --keystorePassword=member2
```

4.3 Update the server.xml with the output XML from the command.
Update the HTTP and HTTPS: http=9082, https=9445

```
# gedit usr/servers/member2/server.xml
```


The updated server.xml for member2 should look like this:

```
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>jsp-2.2</feature>
    </featureManager>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="localhost"
                  httpPort="9082"
                  httpsPort="9445" />

    <featureManager>
        <feature>collectiveMember-1.0</feature>
    </featureManager>

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

    <!-- Connection to the collective controller -->
    <collectiveMember controllerHost="localhost"
                      controllerPort="9443" />

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

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

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

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

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


</server>
```


4.4 Start the collective member.

```
# bin/server start member2
```

4.5 Verify the server started correctly and is publishing information to the controller.

```
# gedit usr/servers/member2/logs/messages.log
```

Look for the following messages (in any order):

```
CWWKX8112I: The server's host information was successfully
published to the collective repository.
CWWKX8114I: The server's paths were successfully published to the
collective repository.
CWWKX8116I: The server STARTED state was successfully published
to the collective repository.
```

4.6 Assign member1 and member2 to the default cluster. The cluster configuration is dynamically updated and published to the collective controller.
Add the following configuration to each server's server.xml:

```
<featureManager>
    <feature>clusterMember-1.0</feature>
</featureManager>
```

4.7 Get status for "defaultCluster" using the getClusterStatus.py script.

```
# jython $LAB_MATERIALS/sample_scripts/getClusterStatus.py
defaultCluster --host=localhost --port=9443 --user=admin
--password=adminpwd
--truststore=usr/servers/controller1/resources/security/trust.jks
--truststorePassword=password
```

4.8 Stop the cluster using the stopCluster.py script.

```
# jython $LAB_MATERIALS/sample_scripts/stopCluster.py
--clusterName=defaultCluster --host=localhost --port=9443
--user=admin --password=adminpwd
--truststore=usr/servers/controller1/resources/security/trust.jks
--truststorePassword=password
```

4.9 Verify the servers in the cluster group have stopped.

```
# bin/server status member1
# bin/server status member2
```

4.10 Start the cluster using the startCluster.py script.

```
# jython $LAB_MATERIALS/sample_scripts/startCluster.py
--clusterName=defaultCluster --host=localhost --port=9443
--user=admin --password=adminpwd
--truststore=usr/servers/controller1/resources/security/trust.jks
--truststorePassword=password
```


The servers member1 and member2 now belong to the cluster group defaultCluster. Multiple clusters can be defined with a single collective, but a server may only belong to one cluster group at a time.

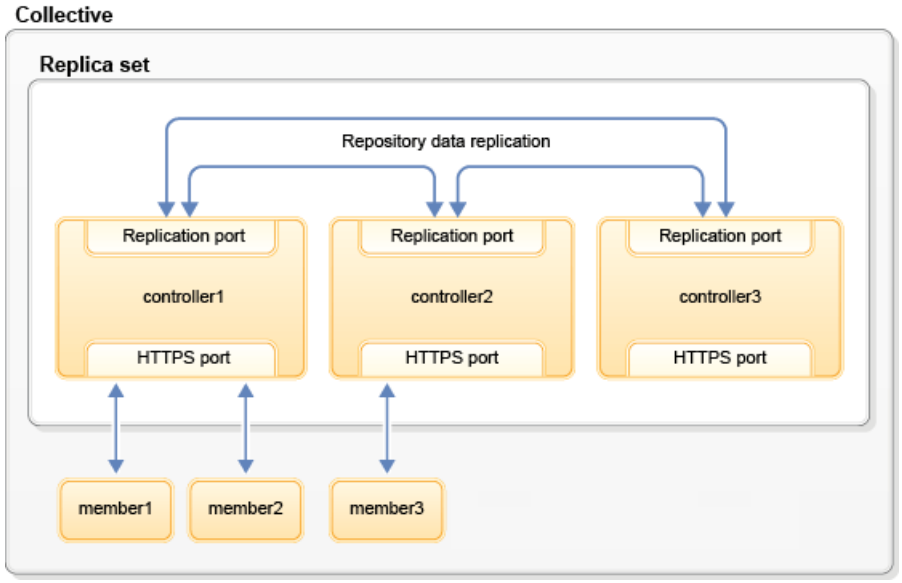
The ClusterManager MBean is an example of how the collective controller acts as an operational repository. Data queries, such as the listClusterNames, listMembers and getStatus operations are performed entirely against the operational cache within the collective controller, while operations which require action on a target such as start and stop are performed against the respective collective member.

The cluster will be built upon in Section 4.6 Deploying applications to the cluster.

4.5 Step 5: Building out a highly available controller topology

In this section, you increase the number of collective controllers to provide a highly available administrative domain. Each new collective controller is called a "replica", as they have the same security configuration as the original controller, and because all information written to any one controller is automatically replicated to all other active controllers. Once configured, all collective controllers in the replica set are capable of performing the same operations as the original controller.

Topology diagram of a collective created in this step



5.1 Create the second and third controllers (replicas).

```
# bin/server create controller2
# bin/server create controller3
```


5.2 Replicate the original collective controller's administrative security domain configuration into the new replicas. Update the HTTP and replicaPort values.
controller2: http=9083, https=9446, replicaPort=10011
controller3: http=9084, https=9447, replicaPort=10012
Be sure to set the security configuration and root keystore password as defined in the original controller.

```
# bin/collective replicate controller2 --host=localhost
--port=9443 --user=admin --password=adminpwd
--keystorePassword=controller2

# gedit usr/servers/controller2/server.xml

# bin/collective replicate controller3 --host=localhost
--port=9443 --user=admin --password=adminpwd
--keystorePassword=controller3

# gedit usr/servers/controller3/server.xml
```


The updated server.xml for controller2 should look like this:

```
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>jsp-2.2</feature>
    </featureManager>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="localhost"
                  httpPort="9083"
                  httpsPort="9446" />

    <featureManager>
        <feature>collectiveController-1.0</feature>
    </featureManager>

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

    <!-- Configuration of the collective controller replica.
         TODO: If this replica is on the same host as the original controller,
               change the replicaPort.
         TODO: If the target controller's replica port is not 10010
               (the default) change the value in replicaSet. -->
    <collectiveController replicaPort="10011"
                          replicaSet="localhost:10010"
                          isInitialReplicaSet="false" />

    <!-- TODO: Define the security configuration exactly as defined in the
               target controller from which this was replicated. -->
    <quickStartSecurity userName="admin" userPassword="adminpwd" />

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

    <!-- inbound (HTTPS) keystore -->
    <keyStore id="defaultKeyStore" password="{xor}PDAxKy0wMzM6LW0="
              location="${server.config.dir}/resources/security/key.jks" />

    <!-- inbound (HTTPS) truststore -->
    <keyStore id="defaultTrustStore" password="{xor}PDAxKy0wMzM6LW0="
              location="${server.config.dir}/resources/security/trust.jks" />

    <!-- server identity keystore -->
    <keyStore id="serverIdentity" password="{xor}PDAxKy0wMzM6LW0="
              location="${server.config.dir}/resources/collective/serverIdentity.jks" />

    <!-- collective truststore -->
    <keyStore id="collectiveTrust" password="{xor}PDAxKy0wMzM6LW0="
              location="${server.config.dir}/resources/collective/collectiveTrust.jks" />

    <!-- collective root signers keystore
         TODO: set password to the collectiveRootKeys password in the
         original controller -->
    <keyStore id="collectiveRootKeys" password="password"
              location="${server.config.dir}/resources/collective/rootKeys.jks" />

</server>
```


The updated server.xml for controller3 should look like this:

```
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>jsp-2.2</feature>
    </featureManager>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="localhost"
                  httpPort="9084"
                  httpsPort="9447" />

    <featureManager>
        <feature>collectiveController-1.0</feature>
    </featureManager>

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

    <!-- Configuration of the collective controller replica.
         TODO: If this replica is on the same host as the original controller,
               change the replicaPort.
         TODO: If the target controller's replica port is not 10010
               (the default) change the value in replicaSet. -->
    <collectiveController replicaPort="10012"
                          replicaSet="localhost:10010"
                          isInitialReplicaSet="false" />

    <!-- TODO: Define the security configuration exactly as defined in the
               target controller from which this was replicated. -->
    <quickStartSecurity userName="admin" userPassword="adminpwd" />

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

    <!-- inbound (HTTPS) keystore -->
    <keyStore id="defaultKeyStore" password="{xor}PDAxKy0wMzM6LWw="
              location="${server.config.dir}/resources/security/key.jks" />

    <!-- inbound (HTTPS) truststore -->
    <keyStore id="defaultTrustStore" password="{xor}PDAxKy0wMzM6LWw="
              location="${server.config.dir}/resources/security/trust.jks" />

    <!-- server identity keystore -->
    <keyStore id="serverIdentity" password="{xor}PDAxKy0wMzM6LWw="
              location="${server.config.dir}/resources/collective/serverIdentity.jks" />

    <!-- collective truststore -->
    <keyStore id="collectiveTrust" password="{xor}PDAxKy0wMzM6LWw="
              location="${server.config.dir}/resources/collective/collectiveTrust.jks" />

    <!-- collective root signers keystore
         TODO: set password to the collectiveRootKeys password in the
         original controller -->
    <keyStore id="collectiveRootKeys" password="password"
              location="${server.config.dir}/resources/collective/rootKeys.jks" />

</server>
```


5.3 Start the new collective controller replicas.

```
# bin/server start controller2
# bin/server start controller3
```

5.4 Validate that the original collective controller can communicate with the new replicas.

```
# gedit usr/servers/controller1/logs/messages.log
```

Look for the following messages (note the IP addresses may be different):

```
CWWKX6009I: The collective controller successfully connected to
replica 127.0.0.1:10011. Current active replica set is
[127.0.0.1:10010]. The configured replica set is [127.0.0.1:10010
]. The connected standby replicas are [127.0.0.1:10011].

CWWKX6009I: The collective controller successfully connected to
replica 127.0.0.1:10012. Current active replica set is
[127.0.0.1:10010]. The configured replica set is [127.0.0.1:10010
]. The connected standby replicas are [127.0.0.1:10011,
127.0.0.1:10012].
```

5.5 Validate that the new replicas can communicate with the original controller.

```
# gedit usr/servers/controller2/logs/messages.log
# gedit usr/servers/controller3/logs/messages.log
```

Look for the following message (note the IP addresses may be different):

```
CWWKX6009I: The collective controller successfully connected to
replica 127.0.0.1:10012. Current active replica set is []. The
configured replica set is []. The connected standby replicas are
[127.0.0.1:10011, 127.0.0.1:10010, 127.0.0.1:10012].
```

5.6 Add the newly replicated controllers to the repository configuration using the updateRepositoryConfig.py script. Ensure that the value for --endpoint is the correct IP address as reported by the above messages. For example, if the reported replica contains 12.34.56.78:10011, then --endpoint should be --endpoint=12.34.56.78.

```
# jython $LAB_MATERIALS/sample_scripts/updateRepositoryConfig.py
add --endpoint=myhost.com:10011 --host=localhost --port=9443
--user=admin --password=adminpwd
--truststore=usr/servers/controller1/resources/security/trust.jks
--truststorePassword=password

# jython $LAB_MATERIALS/sample_scripts/updateRepositoryConfig.py
add --endpoint=myhost.com:10012 --host=localhost --port=9443
--user=admin --password=adminpwd
--truststore=usr/servers/controller1/resources/security/trust.jks
--truststorePassword=password
```


5.7 Validate that the original collective controller has been updated with the new replica endpoints, and that the new replicas are successfully synchronized.

```
# gedit usr/servers/controller1/logs/messages.log
```

Look for the following messages:

```
CWWKX6015I: A request to change the active collective controller
replica set was received and is now processing. The current
active replica set is {127.0.0.1:10010,127.0.0.1:10011}. The
requested new active replica set is
{127.0.0.1:10010,127.0.0.1:10011,127.0.0.1:10012}.

CWWKX6016I: The active collective controller replica set changed
successfully. The current active replica set is
{127.0.0.1:10010,127.0.0.1:10011,127.0.0.1:10012}. The previous
active replica set was {127.0.0.1:10010,127.0.0.1:10011}.

CWWKX6011I: The collective controller is ready, and can accept
requests. The leader is 127.0.0.1:10010. Current active replica
set is [127.0.0.1:10012, 127.0.0.1:10011, 127.0.0.1:10010]. The
configured replica set is [127.0.0.1:10010, 127.0.0.1:10011,
127.0.0.1:10012].

CWWKX6014I: This collective controller replica finished
synchronizing the data with the other replicas.
```

5.8 Validate the new replicas are successfully synchronized and ready for work. The newly added replicas will publish their state into the repository once they are activated in the replica set.

```
# gedit usr/servers/controller2/logs/messages.log
# gedit usr/servers/controller3/logs/messages.log
```

Look for the following messages (in any order):

```
CWWKX6016I: The active collective controller replica set changed
successfully. The current active replica set is
{127.0.0.1:10010,127.0.0.1:10011,127.0.0.1:10012}. The previous
active replica set was {127.0.0.1:10010,127.0.0.1:10011}.
CWWKX6011I: The collective controller is ready, and can accept
requests. The leader is 127.0.0.1:10010. Current active replica
set is [127.0.0.1:10012, 127.0.0.1:10011, 127.0.0.1:10010]. The
configured replica set is [127.0.0.1:10010, 127.0.0.1:10011,
127.0.0.1:10012].

CWWKX8112I: The server's host information was successfully
published to the collective repository.
CWWKX8114I: The server's paths were successfully published to the
collective repository.
CWWKX8116I: The server STARTED state was successfully published
to the collective repository.
```


5.9 Add another member using the new controller2 replica.
Update the HTTP and HTTPS port for the server: http=9085, https=9448

```
# bin/server create member3

# bin/collective join member3 --host=localhost --port=9446
--user=admin --password=adminpwd --keystorePassword=member3

# gedit usr/servers/member3/server.xml

# bin/server start member3
```

The updated server.xml for member3 should look like this:

```
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>jsp-2.2</feature>
    </featureManager>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="localhost"
                  httpPort="9085"
                  httpsPort="9448" />

    <featureManager>
        <feature>collectiveMember-1.0</feature>
    </featureManager>

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

    <!-- Connection to the collective controller -->
    <collectiveMember controllerHost="localhost"
                      controllerPort="9446" />

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

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

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

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

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

</server>
```


5.10 Verify that the server started correctly and is publishing information to the collective controllers.

```
# gedit usr/servers/member3/logs/messages.log
```

Look for the following messages (in any order):

```
CWWKX8112I: The server's host information was successfully
published to the collective repository.
CWWKX8114I: The server's paths were successfully published to the
collective repository.
CWWKX8116I: The server STARTED state was successfully published
to the collective repository.
```

5.11 Reconfigure the members to be able to connect to any of the controllers. Adding this configuration will allow each collective member to connect to any of the active replicas. A collective member only makes one connection to any of the available controllers. When the connection is lost, the member will try to re-establish a connection to any of the other configured controllers.

Change:

```
    <!-- Connection to the collective controller -->
    <collectiveMember controllerHost="localhost"
                      controllerPort="9443" />
```

To:

```
    <!-- Connection to the collective controller -->
    <collectiveMember controllerHost="localhost"
                      controllerPort="9443">
        <failoverController host="localhost" port="9446"/>
        <failoverController host="localhost" port="9447"/>
    </collectiveMember>
```

```
# gedit usr/server/member1/server.xml
# gedit usr/server/member2/server.xml
# gedit usr/server/member3/server.xml
```

Changing the connection configuration for the member will result in a new connection. You can confirm the new connection in the logs. Look for the following message:

```
CWWKX8055I: The collective member has established a connection to
the collective controller.
```

5.12 Stop the original controller. The members will automatically fail-over to a different active controller.

```
# bin/server stop controller1
```

You now have a highly available collective topology, with three collective controllers: controller1, controller2 and controller3. Each collective controller can perform the exact same duties as any other controller, including all collective or cluster operations.

The topology now also has three members: member1, member2 and member3. Each member is configured with the connection details for all three controllers so if any one controller is unreachable, the members will be able to establish a connection to one of the remaining active controllers. In the event a collective member is unable to connect to a controller, all publishing activities will block until the connection to the controller becomes available.

Further notes:

- Servers may only be part of one collective at any give time.
- In order for the collective controllers to perform operations, at least a majority (N/2+1) must be available. This is required to prevent the "split-brain" problem. See http://en.wikipedia.org/wiki/Split-brain_(computing)

4.6 Deploying applications to the cluster

In this section, you build upon the work done in previous sections and use some sample scripts, available from wasdev.net, which will allow you to deploy applications and configuration to all members of a cluster. This section demonstrates the type of compound operations which are possible with the MBeans available in WebSphere Application Server Liberty profile V8.5.5, as well as describes a best practice for cluster configuration.

6.1 Update the server.xml for each member to support writing to the server's configuration directory. This is necessary for the subsequent steps which will use the file transfer service to deploy applications and new configuration to the cluster. By default, the file transfer service does not permit write operations.

```
# gedit usr/servers/member1/server.xml
# gedit usr/servers/member2/server.xml
```

Add the following lines to the server.xml. These lines will enable the file transfer service to write to the server's configuration directory.

```
<remoteFileAccess>
      <writeDir>${server.config.dir}</writeDir>
</remoteFileAccess>
```

6.2 Create a bootstrap.properties file for each cluster member. This file will contain the configuration which is unique to each server, such as port numbers or host names. This will support using a common server.xml for use by all of the cluster members. Set the contents of each bootstrap.properties as describe below.

```
# gedit usr/servers/member1/bootstrap.properties
host.name=myhost.com
http.port=9081
https.port=9444
keystore.password=member1
```


6.2 Create a bootstrap.properties file for each member. (continued)

```
# gedit usr/servers/member2/bootstrap.properties
host.name=myhost.com
http.port=9082
https.port=9445
keystore.password=member2
```

6.3 Copy the server.xml for member1 and edit it to use the new values defined in the bootstrap.properties file. This copy will serve as the basis for the common cluster member configuration and will use variable substitution for the server specific configuration values.

```
# cp usr/servers/member1/server.xml clusterServer.xml
# gedit clusterServer.xml
```

Modify the following XML lines to use the new values in bootstrap.properties:

```
<httpEndpoint id="defaultHttpEndpoint"
              host="*"
              httpPort="${http.port}"
              httpsPort="${https.port}" />

<variable name="defaultHostName" value="${host.name}" />

<keyStore id="defaultKeyStore" password="${keystore.password}"

<keyStore id="defaultTrustStore" password="${keystore.password}"

<keyStore id="serverIdentity" password="${keystore.password}"

<keyStore id="collectiveTrust" password="${keystore.password}"
```


The updated clusterServer.xml should look like this:

```
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>jsp-2.2</feature>
    </featureManager>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="${http.port}"
                  httpsPort="${https.port}" />

    <featureManager>
        <feature>collectiveMember-1.0</feature>
        <feature>clusterMember-1.0</feature>
    </featureManager>

    <remoteFileAccess>
        <writeDir>${server.config.dir}</writeDir>
    </remoteFileAccess>

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

    <!-- Connection to the collective controller -->
    <collectiveMember controllerHost="localhost"
                      controllerPort="9443">
        <failoverController host="localhost" port="9446"/>
        <failoverController host="localhost" port="9447"/>
    </collectiveMember>

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

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

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

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

    <!-- collective truststore -->
    <keyStore id="collectiveTrust" password="${keystore.password}"
              location="${server.config.dir}/resources/collective/collectiveTrust.jks" />

</server>
```


6.4 Deploy the application "snoop" to the cluster. The transferAppToCluster.py script uses a combination of the ClusterManager and FileTransfer MBeans to push the application to all of the cluster members. This operation will push the application to ${server.config.dir}/apps/ for each cluster member.

```
# jython $LAB_MATERIALS/sample_scripts/transferAppToCluster.py
$LAB_MATERIALS/applications/snoop.war --clusterName=defaultCluster
--host=localhost --port=9446 --user=admin --password=adminpwd
--truststore=usr/servers/controller2/resources/security/trust.jks
--truststorePassword=controller2
```

You can confirm the file was transferred to each member of the cluster:

```
# ls usr/servers/member1/apps
# ls usr/servers/member2/apps
```

6.5 Update the common cluster configuration with the application "snoop" configuration.

```
# gedit clusterServer.xml
```

Add the following lines to the clusterServer.xml:

```
<application type="war" id="snoop" name="snoop"
             location="${server.config.dir}/apps/snoop.war"/>
```

6.6 Restart the cluster members using the stopCluster.py and startCluster.py scripts. This step is required as the bootstrap.properties file is not dynamically loaded. The bootstrap.properties is processed only during JVM start-up.

```
# jython $LAB_MATERIALS/sample_scripts/stopCluster.py
--clusterName=defaultCluster --host=localhost --port=9446
--user=admin --password=adminpwd
--truststore=usr/servers/controller2/resources/security/trust.jks
--truststorePassword=controller2

# jython $LAB_MATERIALS/sample_scripts/startCluster.py
--clusterName=defaultCluster --host=localhost --port=9446
--user=admin --password=adminpwd
--truststore=usr/servers/controller2/resources/security/trust.jks
--truststorePassword=controller2
```

6.7 Deploy the common cluster configuration to all of the members of the cluster using the transferConfigToCluster.py script.

```
# jython $LAB_MATERIALS/sample_scripts/transferConfigToCluster.py
clusterServer.xml --clusterName=defaultCluster --host=localhost
--port=9446 --user=admin --password=adminpwd
--truststore=usr/servers/controller2/resources/security/trust.jks
--truststorePassword=controller2
```


6.8 Access the application "snoop" running on member1 and member2.

Go to the URLs: http://localhost:9081/snoop
http://localhost:9082/snoop

6.9 Install the application "ImpactWeb" to the cluster using the manageAppOnCluster.py script. This operation combines application deploy and configuration file update actions and demonstrates an alternate pattern for installing an application to a cluster.

```
# jython $LAB_MATERIALS/sample_scripts/manageAppOnCluster.py
--install=$LAB_MATERIALS/applications/ImpactWeb.war
--clusterName=defaultCluster --host=localhost --port=9446
--user=admin --password=adminpwd
--truststore=usr/servers/controller2/resources/security/trust.jks
--truststorePassword=controller2
```

6.10 Access the application "ImpactWeb" running on member1 and member2.

Go to the URLs: http://localhost:9081/ImpactWeb/WorkingServlet
http://localhost:9082/ImpactWeb/WorkingServlet

You now have an active cluster group with the "snoop" and "ImpactWeb" applications deployed. The cluster configuration is written in such a way that new cluster members can be easily added by creating new servers and setting their bootstrap.properties file accordingly. The use of bootstrap.properties to separate server specific properties is one strategy to enable common configuration.

An alternate choice to support common configuration is to use includes supported by the server.xml to store the server specific configuration in a separate file. More details on this can be found in the InfoCenter article "Using include elements in configuration files".

5 Additional Steps

These additional steps are recommendations to continue learning and experimenting with the collective and cluster capabilities. They are listed in no particular order and are listed here as suggestions of things to try. Feel free to play!

- Bring up the topology in any order.
  - Collective members can be started without any controller being active.
  - Collective controllers can be started in any order, the initial controller does not need to be started first (or even active) once the replicas have been added.

- Further experiment with Jython scripting.
  - All of the available APIs can be found in $WLP_INSTALL_DIR/dev/api/ibm/javadoc/

- Route JMX requests through the controller.
  - The routing context can be set using the RoutingContext MBean: WebSphere:feature=collectiveController,type=RoutingContext,name=RoutingContext

    This MBean allows MBean requests to be routed through the controller to a registered, active collective member.

6 Appendix – Additional Resources

- InfoCenter for WebSphere Application Server V8.5

- wasdev.net

- The Liberty profile JavaDoc can be found in the installation directory:
  - API - $WLP_INSTALL_DIR/dev/api/ibm/javadoc
  - SPI - $WLP_INSTALL_DIR/dev/spi/ibm/javadoc

Thank you!
This lab is available from wasdev.net

https://www.ibmdw.net/wasdev/docs/lab-build-and-administer-your-own-liberty-application-cluster

