General Settings

The settings.xml file holds environmental information and properties about the Aspire installation as a whole.

On this page

- Separation of Scope and Portability
- Settings File Location
- Structure of settings.xml
- Auto Start Section
- Repositories Section
- Properties
- Properties for Applications
- Apache Felix Configuration
- Distributed Processing
- Security Configuration
- Zookeeper Configuration
- Entitlements

Separation of Scope and Portability

The settings.xml file holds environmental information (server addresses, passwords, system properties, repository settings, etc.) for your Aspire installation.

What goes in the Settings File vs the Application XML configuration file?

Appropriate for the settings.xml Configuration File:

- Server names
- Port addresses
- User names & passwords
- App Bundle properties
- Applications to launch on startup

Appropriate for the application.xml Configuration File:

- Component configuration
- Pipeline configuration
- All document processing functionality

Many components in the system configuration file require server names, user names, etc. as configuration elements. To make your system configuration file portable over multiple installations, these configuration elements should be specified as properties in the settings.xml file.

Settings File Location

On startup, the Aspire application will automatically attempt to load the settings from the following locations:

- The file whose name is given in the **ASPIRE_SETTINGS_PROPERTY** ("*com.searchtechnologies.aspire.settings*"). This can be set supplying the parameter **-Dcom.searchtechnologies.aspire.settings=*filename*** on the JVM command line.
- The **/config/settings.*hostname*.xml** under the Aspire Home directory, set via the **ASPIRE_HOME_PROPERTY** ("*com.searchtechnologies.aspire.home*"). This can be set supplying the parameter **-Dcom.searchtechnologies.aspire.home=*directory*** on the JVM command line.
- The **/config/settings.xml** under the Aspire Home directory.

Structure of settings.xml

The settings.xml file contains sections for automatically starting system configuration files, setting Aspire system properties, and setting Apache Felix system properties. The overall structure is as follows:

```
<settings>
  <!-- Specify system properties which are available to all Aspire system configuration files -->
  <properties>
    <property name="solrServer">http://localhost:8080</property>
    <property name="autoStart">false</property>
    .
    .
    .
  </properties>
 <!--Entitlements -->
 <entitlementsServer online="true">https://entitlements.searchtechnologies.com</entitlementsServer>

  <!-- Specify properties for the OSGI Configuration Administration server. These are specific
       properties required by individual components within Apache Felix. Many of these properties
       may be better configured in the "felix.properties" file -->
  <configAdmin>
    <properties pid="org.apache.felix.webconsole.internal.servlet.OsgiManager">
      <property name="username">admin</property>
      <property name="password">admin1</property>
```

```
      <property name="manager.root">/osgi</property>
    </properties>
  </configAdmin>

  <!-- Specify applications to automatically install on startup, and provide the system
       configuration file for each one -->
  <autoStart>
  <!-- Workflow manager is required for 2.0 UI -->
   <application config="com.searchtechnologies.aspire:app-workflow-manager" id="1">
     <properties>
       <property name="templateFile">${appbundle.home}/data/templates.xml</property>
       <property name="libraryPath">${aspire.config.dir}/workflow-libraries</property>
       <property name="planFile"/>
       <property name="disableInternalTemplates">false</property>
       <property name="allowCustomRule">true</property>
       <property name="debug">false</property>
     </properties>
   </application>
  </autoStart>


  <!-- Repositories specify where components can be found or downloaded -->
  <repositories>
    <repository type="distribution">
      <directory>bundles/aspire</directory>
    </repository>

    <repository type="maven">
      <defaultVersion>3.2</defaultVersion>
      <updatePolicy>always</updatePolicy>
      <remoteRepositories>
        <remoteRepository>
          <id>stPublic</id>
          <url>https://repository.searchtechnologies.com/artifactory/public/</url>
          <user>YOUR-REGISTERED-USERNAME</user>
          <password>YOUR-REGISTERED-PASSWORD</password>
        </remoteRepository>
      </remoteRepositories>
    </repository>

  </repositories>
 <configAdministration>
   <zookeeper libraryFolder="config/workflow-libraries" root="/aspire">
     <clientPort>1112</clientPort>
     <dataDir>config</dataDir>
     <maxConnections>60</maxConnections>
     <tickTime>2000</tickTime>
   </zookeeper>
 </configAdministration>
 <authentication>
   <type>none</type>
 </authentication>
 <ldapConfig>
   <config>
     <server>ldap://localhost:389</server>
     <authentication>simple</authentication>
     <searchBase>dc=localhost, dc=com</searchBase>
     <group>OU=Users,OU=Group</group>
   </config>
 </ldapConfig>

</settings>
```

Auto Start Section

The <autoStart> section will automatically load applications when Aspire is initialized. It contains a simple list of application files to load, for example:

```
<autoStart>
  <application config="config/application-common.xml"/>
  <application config="config/application-arc.xml"/>
  <application config="config/application-companydb.xml"/>
  <application config="config/application-rss-feeds.xml"/>
  <application config="config/application-single-pages.xml"/>
  <application config="com.searchtechnologies.appbundles:cs-rdbms-connector:1.0-SNAPSHOT">
    <properties>
      <property name="rdbmsHasDefaults">false</property>
      <property name="debug">true</property>
    </properties>
  </application>
  <!-- Workflow manager is required for 3.2 UI -->
   <application config="com.searchtechnologies.aspire:app-workflow-manager" id="1">
     <properties>
       <property name="templateFile">${appbundle.home}/data/templates.xml</property>
       <property name="libraryPath">${aspire.config.dir}/workflow-libraries</property>
       <property name="planFile"/>
       <property name="disableInternalTemplates">false</property>
       <property name="allowCustomRule">true</property>
       <property name="debug">false</property>
     </properties>
   </application>
</autoStart>
```

Applications are loaded in the order specified. However, since Aspire has component-dependency checking built-in, the order of load is usually not that important.

Both Application XML Files and App Bundles

The <application> tag can launch an application either from an application XML file or an App Bundle.

- For application XML files: The *@config* attribute should hold the file name of the Application XML file to load.
- For App Bundles: The *@config* attribute should hold the Maven coordinates of the App Bundle to start.

Rename Auto-Started Applications

In general, the name of the application will be taken as the "default name" as specified at the top of the application.xml file. See Configuration File Basics for more information.

However, you can specify other names for the configuration file using the *@name* attribute, as shown below:

```
 <application name="RDBConnector2" config="com.searchtechnologies.appbundles:cs-rdbms-connector:2.0>
   <properties>
     <property name="rdbmsHasDefaults">false</property>
     <property name="debug">true</property>
   </properties>
 </application>
```

This lets you install the same App Bundle multiple times, but with different top-level names.

Application Properties

Finally, as shown above, applications can have a nested <properties> tag which holds properties that are defined just for that application. These properties can then be used with the ${propName} substitution pattern within the application.xml file.

Repositories Section

The <repositories> tag identifies where to find component code to load into Aspire. There are two types of repositories: Distribution and Maven.

Distribution Repository

The Distribution Repository will load the component Jar files in a directory within your Aspire distribution, typically the "bundles/aspire" directory.

It is configured as follows:

```
<repository type="distribution">
   <directory>bundles/aspire</directory>
</repository>
```

The <directory> tag identifies the directory where the bundles can be located.

On startup, Aspire will scan through the entire directory looking for bundles to load. If at any time you add new bundles (or update bundles) in this directory, then click on "Check for Updates" on the Aspire application home page. This will cause Aspire to re-scan the directory so that the new files are available.

Maven Repository

The Maven Repository loads the component Jar files directly from Maven. The Maven Repository allows Aspire to share the same Jars as Eclipse and the Maven command-line program. Therefore, any newly 'install'ed or 'deploy'ed Jar file artifacts will be automatically available to Aspire.

It is configured as follows:

```
  <repository type="maven">
    <localRepository>~search/.m2/repository</localRepository>
    <defaultVersion>3.2</defaultVersion>
    <updatePolicy>always</updatePolicy>
    <offline>false</offline>
    <remoteRepositories>
      <remoteRepository>
        <id>stPublic</id>
        <url>https://repository.searchtechnologies.com/artifactory/public/</url>
        <user>YOUR-REGISTERED-USERNAME</user>
        <password>YOUR-REGISTERED-PASSWORD</password>
      </remoteRepository>
    </remoteRepositories>
  </repository>
```


Replace "YOUR-REGISTERED-USERNAME" and "YOUR-REGISTERED-PASSWORD" with the actual user name and password you used to register for Aspire.

The Maven Repository uses the standard APIs for accessing artifacts from Maven repositories (the same as used by the Maven program itself). It will download artifacts as necessary from any of the the remote repositories specified and store them in the local repository. It will then load the bundle Jar files directly from the local repository into Aspire.

The following options are available for the maven repository:

Element	Type	Default	Description
localRepository	string	(user home directory)/.m2/repository	(optional) Specifies the location of the Maven local repository, where Jars will reside locally once they are downloaded from the remote repository. This is also the location where Maven "install" will install new or updated artifacts.
defaultVersion	string	LATEST	(Strongly Recommended) Specifies the default version for all artifacts for which no version is specified. Note that this defaults to a version of "LATEST" - but unfortunately this has some odd behavior between the local and remote repositories (it only checks the local repository if the version is available on the remote repository, and the remote repository has been "scanned").
offline	boolean	false	(optional) Specifies if the system is "offline" - in which case the Maven repository will only ever look to the local repository for artifacts, and never the remote repositories.
remoteRepository/id	string	none	(required for all remote repositories) The server identifier for the remote repository. This identifier should be the same as the identifier used for the repository as specified in the Eclipse or maven command-line Maven settings file. This is usually "stPublic".
remoteRepository/url	string	none	(required for all remote repositories) The URL where the remote repository is located
user	string	none	(optional) The user name for logging into the remote repository.
password	string	none	(optional) The password for logging into the remote repository. You can encrypt it following the instructions here.

Use Specific Versions of Bundle

If required, you can force the Maven repository to give you a specific version of a bundle if you don't specify it in the *factoryName* in application.xml files or in the *config* attribute in the *autoStart* section of the settings file.

Normally in Aspire, if references to Maven artifacts to not give the *version*, then the *defaultVersion* (see above) is used. However, you may add a *bundleVersions* section to the settings file to give more precise control over the versions of bundles loaded. The parameters are shown below:

Element	Type	Default	Description
bundleVersions\bundle\@groupId	String	com.searchtechnologies	The (optional) group id for the bundle to version
bundleVersions\bundle\@artifactId	String		**MANDATORY**: The artifact id for the bundle to version
bundleVersions\bundle\@version	String		**MANDATORY**: The version of the bundle to request from Maven

If a requested bundle is not configured in the *bundleVersions* section, then the *defaultVersion* (as configured above) of that bundle will be requested.

If the version specified is not located in Maven, an error will occur.

Example:

The following snippet will load all requested bundles at version *1.0*, except the three specified, which will be loaded at the requested version

```
 <repository type="maven">
  <defaultVersion>1.0</defaultVersion>
  <bundleVersions>
    <bundle artifactId="aspire-tools" version="1.1"/>
    <bundle groupId="com.myCustomer" artifactId="myCustomerArtifact" version="0.1-SNAPSHOT"/>
    <bundle groupId="com.searchtechnologies" artifactId="aspire-business-rules" version="2.0"/>
  </bundleVersions>
  <remoteRepository>
    <id>stPublic</id>
    <url>https://repository.searchtechnologies.com/artifactory/public/</url>
    <user>YOUR-REGISTERED-USERNAME</user>
    <password>YOUR-REGISTERED-PASSWORD</password>
  </remoteRepository>
<repository>
```

Proxy Settings

You can configure Maven remote repositories to use a HTTP proxy for outgoing communications. This is useful when your Aspire server has restricted access to the Internet, and you want to be able to fetch bundles as normal from the configured repository. To do this add a <proxy> section to your remote repository and set the following properties:

Element	Type	Default	Description
host	string	null	(optional) Proxy server hostname or IP address.
port	string	0	(optional) Proxy server port number.
user	string	null	(optional) User required for authentication against the proxy server.
password	string	null	(optional) Password for authenticating against the proxy server. You can encrypt it following the instructions here.

Example:

```
    <remoteRepository>
         <id>stPublic</id>
         <url>{{RepositoryUrl}}</url>
         <user>YOUR-REGISTERED-USERNAME</user>
         <password>YOUR-REGISTERED-PASSWORD</password>
         <proxy>
             <host>127.0.0.1</host>
             <port>8888</port>
             <user>PROXY-USER</user>
             <password>PROXY-PASSWORD</password>
         </proxy>
    </remoteRepository>
```

Default

If no <repositories> tag is specified, a single "distribution" repository with "bundles/aspire" as the directory will be automatically specified.

Properties

Properties are specified as name/value pairs. For example:

```
   <properties>
    <property name="solrServer">http://localhost:8080</property>
    <property name="autoStart">false</property>
    <property name="crawlDataBase">data/crawler</property>
    <property name="ccdBase">data</property>
  </properties>
```

Once specified in the settings.xml file, these properties become available for use in Application XML files. Careful use of such properties will make your system configuration files portable to multiple Aspire installations without modification.

You can use these properties from the UI when configuring content sources, services and workflow applications:

For example, you might use "http://localhost:8080" as your SOLR server on your personal laptop, but then use "http://customer.searchtechnologies.com:8983" for the production site. Using a property will allow the exact same system configuration file to be tested on one machine and then installed on another machine without modification.

Properties & Environment Variables in Application XML Files

Properties declared in the settings.xml file can be used in application XML files with the ${propertyName} syntax. As an example:

```
<component name="feed2Solr" subType="default" factoryName="aspire-post-xml">
  <postXsl>config/aspire2solr.xsl</solrXsl>
  <postUrl>${solrServer}/solr/update</postUrl>
</component>
```

In the above example, the "solrServer" property was defined in the settings.xml file and then referenced with ${solrServer} in the application XML file.

This property value substitution occurs automatically on the component configurations by the Component Manager. It does not require any further intervention or programming on the part of any individual component.

The *${XXX}* syntax can also be used for substitution of environment variables and Java system properties (i.e., those defined on the command line with -Dxxx=yyy). Substitution prefers properties defined in the settings.xml file. If the property is not found in the settings.xml file, the system properties are checked and if still not found, the system environment is checked. Also, in these versions, properties may be defined from other properties:

```
<property name="baseDir">/home/user/aspire</property>
<property name="configDir">${baseDir}/cfg</property>
```

Note: Property references for properties that are *not* in the settings.xml file will be left as-is. This allows for other configurations that use the same syntax (specifically, the Groovy Scripting component) to continue to operate properly.

XML Property Expansion

You can also use properties to add entire XML structures to a component configuration. To achieve this, define a property containing XML in a *<![CDATA[..]]>* section. Then, when expanding this property, you prefix the property name with *xml:* and the contents of the property will be interpreted as XML and added to the structure.

For example:

```
<settings>
  <properties>
    <property name="expanders"><![CDATA[<expanders><expander>${prop1}</expander><expander>two</expander><
/expanders>]]></property>
    <property name="prop1">Lincoln</property>
    <property name="testProp">testVal</property>
    <property name="testDoc"><![CDATA[<doc><title/></doc>]]></property>
    <property name="testDoc1"><![CDATA[<doc1><title1/></doc1>]]></property>
  </properties>
</settings>
```

and

```
 <config>
  ${xml:expanders}
  XXX
  ${xml:testDoc}
  <expanders2>${xml:expanders}</expanders2>
  <expanders_text>${expanders}</expanders_text>
  <test>${testProp}</test>
  ${xml:testDoc1}
  YYY
</config>
```

expands to

```
 <config>XXX<expanders2>
    <expanders>
      <expander>Lincoln</expander>
      <expander>two</expander>
    </expanders>
  </expanders2>
  <expanders_text><expanders><expander>Lincoln</expander><expander>two</expander></expanders><
/expanders_text>
  <test>testVal</test>YYY<expanders>
    <expander>Lincoln</expander>
    <expander>two</expander>
  </expanders>
  <doc>
    <title/>
  </doc>
  <doc1>
    <title1/>
  </doc1>
</config>
```

Property Escaping (for Groovy Scripts)

If you need to insert a property into a Groovy script, assigning it to a String that contains the **** character (such as *${aspire.home}* would), will cause Groovy to raise an error as it sees invalid escaped characters. To avoid this, you can prefix the property name with *escape:* and any **** characters in the contents of the property will be replaced with ****.

For example:

```
<settings>
 <properties>
   <property name="file">c:\top-directory\directory\file.html</property>
 </properties>
</settings>
```

and

```
<config>
 <file>${file}</file>
 <escapefile>${escape:file}</escapefile>
 <fileattr attr="${file}">somevalue</fileattr>
 <escapefileattr escapeattr="${escape:file}">somevalue</escapefileattr>
 <cdata>
   <![CDATA[
      def a = "${file}";
   ]]>
 </cdata>
 <escapecdata>
   <![CDATA[
      def a = "${escape:file}";
   ]]>
 </escapecdata>
</config>
```

expands to

```
<config>
 <file>c:\top-directory\directory\file.html</file>
 <escapefile>c:\\top-directory\\directory\\file.html</escapefile>
 <fileattr attr="c:\top-directory\directory\file.html">somevalue</fileattr>
 <escapefileattr escapeattr="c:\\top-directory\\directory\\file.html">somevalue</escapefileattr>
 <cdata>
   <![CDATA[
     def a = "c:\top-directory\directory\file.html";               <<<< THIS LINE WOULD ERROR IN A GROOVY
SCRIPT
   ]]>
 </cdata>
 <escapecdata>
   <![CDATA[
     def a = "c:\\top-directory\\directory\\file.html";
   ]]>
 </escapecdata>
</config>
```

Properties for Applications

You can specify properties that apply to a specific application (rather than the properties above which apply to all components).

```
<autoStart>
  <application config="config/system.xml">
    <properties>
      <property name="debug">true</property>
      <property name="managerExternalRDB">false</property>
      <property name="managerRDB">CSRDB</property>
    </properties>
  </application>
  <application config="com.searchtechnologies.appbundles:cs-manager:3.2">
    <properties>
      <property name="debug">true</property>
      <property name="managerExternalRDB">false</property>
      <property name="managerRDB">CSRDB</property>
      <property name="managerExternalJDBCUrl"></property>
      <property name="managerExternalJDBCDriverJar"></property>
      <property name="managerExternalJDBCUser"></property>
      <property name="managerExternalJDBCPassword"></property>
    </properties>
  </application>
</autoStart>
```

These properties are passed to all components (and only those components) that exist "under" the component manager. If the same property names are used both at the global level and the component manager level, the component manager definition will be used for components "under" that manager, whilst the global value would be used for other components.

Apache Felix Configuration

Some Apache Felix configuration parameters can also be placed in the settings.xml file, as follows:

```
<configAdmin>
   <properties pid="org.apache.felix.webconsole.internal.servlet.OsgiManager">
     <property name="username">admin</property>
     <property name="password">admin1</property>
     <property name="manager.root">/osgi</property>
   </properties>
</configAdmin>
```

Although, before using this approach, check to see if these parameters can be stored in the Apache Felix system properties file (called "felix.properties" for most Aspire installations). That may be the better location for these properties.

Inside <configAdmin>, each <property> tag contains a "pid" attribute which is the "persistent ID" of the configuration element. The nested properties are the OSGi Configuration properties

See the following for more information about OSGi and Apache Felix configuration properties:

- Apache Felix Framework Usage
- OSGi specifications download page - Go here and download the "Compendium Specification" to get more details on the OSGi configuration server and how it's used.
- Apache Felix Web Console Properties
- Apache Felix HTTP Service Properties

Distributed Processing

If you are operating in a high-volume environment, you can set up Aspire so that document jobs are automatically and efficiently routed to various machines. There is an automated Discovery Manager which detects servers that are brought online and offline, re-routing jobs as needed between machines.

Distributed processing allows for jobs to be sent to remote Aspire Distributions (called remote nodes). This increases performance when there are high resource consuming pipelines by load balancing the work across all available remote nodes.

Communications between remote nodes must be tightly coupled, meaning that they must be on the same intranet, geographically in the same place and (recommended) with no firewalls or other security mechanisms between nodes on the cluster.

See below for information on setting up distributed communications between nodes. Refer to [Branch handler] for information on sending jobs to remote nodes.

Distributed Communications

Enabling distributed communication allows Aspire to communicate with other Aspire nodes (such as node discovery, fetching configurations, receiving jobs from other nodes, using Admin UI, etc).

The following options are available:

Element	Type	Default	Description
@enabled	boolean	false	(optional) Enables distribution communications if true.
checkPointJobRequests	boolean	false	(optional) Specifies if jobs should be saved by the Checkpoint Manager right before sending them to be processed by remote nodes.
connectionIdleTimeout	long	120000	(Optional) Sets the max time an idle connection is open until being closed by a background thread.
port	int	51505	(optional) Specifies the port used for distributed communications. This port must be different if you have multiple instances of Aspire running on the same machine. All distributed messages (sending jobs, fetching resources, ...) are received using this port. This port is normally 1000+ of Aspire administration port (which defaults to 50505).
pollTimeout	long	5000	(optional) Once a remote node is detected, it is added to an update queue. Every *pollTimeout* milliseconds, remote nodes are taken from the queue and registered for remote branching. Use this option to make remote node updates to be processed faster or slower.
tcp		none	(expert) List of TCP options that control the way underlying socket connections are handled. For more details on available options, please refer to Interface SocketOptions .
discoveryManager		none	(optional) Sets discovery managers that manage all remote nodes discovered by the configured discovery methods.
discoveryManager/@type	string	none	Specifies the type of Discovery Manager that will be used.
discoveryManager/discovery		none	(optional) Sets discovery methods that will be used to detect and register new remote nodes. For example, you may configure broadcasting discovery method to automatically detect Aspire remote nodes. This is optional, since you may want this node to be a worker node (no remote branching).
installIgnoredPaths		none	(optional) When installing Aspire remotely, a copy of the current distribution will be transferred to the target server. This option defines a comma separated list of paths (relative from Aspire Home) that will be ignored when cloning the current distribution. Example: appbundle-cache, cache, data, felix-cache, log
receipt		no	(optional) Sets whether or not to wait for remote job receipts. When enabled, remote jobs will not be marked as complete until a receipt from the remote node, which is processing the job, sends information of its status (failed or completed). Supports "no", "yes" and "full". The difference between "yes" and "full" is that with "full" the job result (AspireObject) will be returned as well.

Discovery Managers

Default Discovery Manager

This Discovery Manager has the basic functionality for remote nodes and resource discovery.

Example configuration:

```
<discoveryManager type="default">
</discoveryManager>
```

Discovery Methods

One of the following discovery methods must be configured inside the `<discoveryManager type="default">` tag.

Static Discovery

Useful for debugging or in well known cluster setups (with static IP addresses configurations). Reads a list of remote nodes and registers them for remote branching.

Each node is identified by its IP Address and its distributed communications port.

Available options are:

Element	Type	Default	Description
checkTimeout	long	15000	(optional) Time in milliseconds to wait between each *ping* to the configured remote node. After a few retries, if a node is unavailable, it will be blacklisted (remote branching to that node will not be possible).
remoteNodes		none	Static list of remote nodes to register.

Example configuration:

```
 <discoveryManager type="default">
        <discovery type="static">
                <checkTimeout>45000</checkTimeout>
                <remoteNodes>
                        <remoteNode portNumber="51510">10.10.30.122</remoteNode>
                        <remoteNode portNumber="51515">10.10.20.139</remoteNode>
                </remoteNodes>
        </discovery>
</discoveryManager>
```

You must specify each remote node IP Address and port as shown above. Notice that the port number corresponds to the distributed communications port, not the Aspire ordinary HTTP port.

Broadcast Discovery

Intercepts discovery messages sent by other nodes. If the incoming message is from a new node, it is registered. Otherwise, that node information is updated.

You must enable this discovery method if you want a node to broadcast information about itself.

Available options are:

Element	Type	Default	Description
broadcastPort	int	none	(required) Port used to listen for other nodes messages and to broadcast information about the current node.
multicastAddressGroup		none	Multicast address group used to listen and send broadcast messages. This group must be the same on all nodes on the same cluster.

Example configuration:

```
 <discoveryManager type="default">
        <discovery type="broadcast">
                <broadcastPort>10324</broadcastPort>
                <multicastAddressGroup>230.0.0.1</multicastAddressGroup>
        </discovery>
</discoveryManager>
```

Complete example

```
<distributedCommunications enabled="true">
       <checkpointJobRequests>true</checkpointJobRequests>
       <connectionIdleTimeout>120000</connectionIdleTimeout>
       <port>51510</port>
       <pollTimeout>100</pollTimeout>
       <tcp>
              <keepAlive>false</keepAlive>
              <trafficClass>2</trafficClass>
              <reuseAddress>false</reuseAddress>
              <readTimeout>10000</readTimeout>
              <tcpNoDelay>false</tcpNoDelay>
       </tcp>
       <discoveryManager type="default">
              <discovery type="static">
                     <checkTimeout>45000</checkTimeout>
                     <remoteNodes>
                            <remoteNode portNumber="51515">192.168.0.122</remoteNode>
                            <remoteNode portNumber="51515">10.10.20.139</remoteNode>
                     </remoteNodes>
              </discovery>

              <discovery type="broadcast">
                     <broadcastPort>10324</broadcastPort>
                     <multicastAddressGroup>230.0.0.1</multicastAddressGroup>
              </discovery>
       </discoveryManager>
</distributedCommunications>
```

Amazon EC2 Discovery Manager

aspire-amazonec2-dm THIS ITEM IS BEING DEPRECATED.

Since you cannot use the broadcast discovery method at Amazon Elastic Compute Cloud because of network restrictions, you can use the Amazon EC2 Discovery Manager for dynamic discovering of remote nodes.

Element	Type	Default	Description
implementation	string	none	(required) Maven coordinates of the jar file that contains the implementation for the Discovery Manager.

Example configuration:

```
<discoveryManager type="amazonec2">
     <implementation>com.searchtechnologies.aspire:aspire-amazonec2-dm</implementation>

</discoveryManager>
```

Amazon EC2 Discovery Method

Element	Type	Default	Description
accessKey	string	none	(required) Encrypted access key provided by Amazon for your AWS Account.
secretKey	string	none	(required) Encrypted secret key provided by Amazon for your AWS Account.
usePublicIP	boolean	false	(Optional) Specifies whether or not the remote nodes should be accessed by its public IP Address.
securityGroup	string	none	(Optional) Specifies the name of the security group of the ec2 instances that should be discovered.
pollFrequency	int	1000	(Optional) Specifies the frequency for polling the information from Amazon EC2.

Example configuration:

```
 <discoveryManager type="amazonec2">
        <implementation>com.searchtechnologies.aspire:aspire-amazonec2-dm</implementation>

        <discovery type="amazonec2">
                <accessKey>encrypted:ENCRYPTED_ACCESS_KEY</accessKey>
                <secretKey>encrypted:ENCRYPTED_SECRET_KEY</secretKey>
        </discovery>

</discoveryManager>
```

Advanced configuration:

```
 <discoveryManager type="amazonec2">
        <implementation>com.searchtechnologies.aspire:aspire-amazonec2-dm</implementation>

        <discovery type="amazonec2">
                <accessKey>encrypted:ENCRYPTED_ACCESS_KEY</accessKey>
                <secretKey>encrypted:ENCRYPTED_SECRET_KEY</secretKey>
                <usePublicIP>false</usePublicIP>
                <securityGroup>MySecurityGroup</securityGroup>
                <pollFrequency>1000</pollFrequency>
        </discovery>

</discoveryManager>
```

Zookeeper Discovery Manager

aspire-zk-dm THIS ITEM IS BEING DEPRECATED.

This discovery manager uses zookeeper as a centralized site for discovering remote nodes and their resources as well. Click here for details about Zookeeper installation and configuration.

Element	Type	Default	Description
implementation	string	none	(required) Maven coordinates of the jar file that contains the implementations for the Discovery Manager.
zookeeperConnection	string	none	(required) Comma separated list of zookeeper servers.
zookeeperPath	string	/aspire/nodes	(Optional) Znode root path where the remote nodes are registered.
zookeeperTimeout	int	3000	(Optional) Timeout (milliseconds) for the zookeeper server connection. After this time the connection will be considered as disconnected until the zookeeper server comes back online or connection is established to another server.
resourceUpdateTime	int	2000	(Optional) Frequency (milliseconds) for checking for new local resources to register in zookeeper.

Example configuration:

```
 <discoveryManager type="zookeeper">
        <implementation>com.searchtechnologies.aspire:aspire-zk-dm</implementation>
        <zookeeperConnection>127.0.0.1:2182,127.0.0.1:2183,127.0.0.1:2181</zookeeperConnection>
</discoveryManager>
```

Advanced configuration:

```
<discoveryManager type="zookeeper">
        <implementation>com.searchtechnologies.aspire:aspire-zk-dm</implementation>
        <zookeeperConnection>127.0.0.1:2182,127.0.0.1:2183,127.0.0.1:2181</zookeeperConnection>
        <zookeeperPath>/aspire/nodes</zookeeperPath>
        <zookeeperTimeout>3000</zookeeperTimeout>
        <resourceUpdateTime>2000</resourceUpdateTime>
</discoveryManager>
```

Security Configuration

This is the configuration to use the Login page. The type options are (*ConfigFile* or *Ldap*). For more details see Security Page.

Options for authentication are 'None' (default), 'ConfigFile', 'Ldap'

```
<authentication>
   <type>none</type>
</authentication>
```

LDAP authentication options are 'none', 'simple'(default), 'DIGEST-MD5'. If none was selected, anonymous access will be attempted

```
 <ldapConfig>
   <config>
     <server>ldap://localhost:389</server>
     <authentication>simple</authentication>
     <searchBase>dc=localhost, dc=com</searchBase>
     <group>OU=Users,OU=Group</group>
   </config>
 </ldapConfig>
```

Zookeeper Configuration

For keeping the stability and configuration consistent over multiple Aspire servers for content source crawls. For this, Aspire uses Apache ZooKeeper to synchronize configurations (content sources and workflow applications) and coordinate and resume failed crawls, among several Aspire servers. For more details see Failover for Aspire using Zookeeper .

```
  <configAdministration>
   <zookeeper libraryFolder="config/workflow-libraries" root="/aspire">
     <!-- <externalServer>127.0.0.1:2182,127.0.0.1:2183,127.0.0.1:2181</externalServer> -->
     <clientPort>1112</clientPort>
     <dataDir>config</dataDir>
     <maxConnections>60</maxConnections>
     <tickTime>2000</tickTime>
   </zookeeper>
 </configAdministration>
```

Using external ZooKeeper servers is an Aspire Enterprise feature.

Entitlements

This is the server site where entitlements will be located. This is used to populate the *config/entitlements.xml* file used to fill the Home Page connector list and the Workflow section publisher and application lists.

```
  <entitlementsServer online="true">https://entitlements.searchtechnologies.com</entitlementsServer>
```

The *online* attribute can be set to *false* if you have no internet access from the Aspire Server. In this case if you want to populate the connector list of the Home Page, and the publishers and applications list of the Workflow Section you will have to edit the *entitlements.xml* file (also in the config directory) by yourself. You can generate the file contents for your user by visiting Entitlements and following the instructions.