MongoDB Settings

Beginning with release 3.0, Aspire uses an external MongoDB instance for connectors built with the new NoSQL Connector Framework. The database is used to keep crawl metadata and allow processing and scanning to be distributed. All MongoDB configuration is done in the **settings.xml** file.

On this page

Basic Example

```
<!-- noSql database provider for the 3.3 connector framework -->
<noSQLConnectionProvider sslEnabled="false" sslInvalidHostNameAllowed="false">
  <implementation>com.searchtechnologies.aspire:aspire-mongodb-provider</implementation>
  <servers>mongodb-host:27017</servers>
</noSQLConnecitonProvider>
```

Aspire will create one MongoDB database for each content source configured.

- When the content source is deleted, the database will be dropped.
- The database name will be taken from the normalised value of the content source name.

Starting in Aspire 3.3, the database names will be prefixed with "aspire-" to avoid possible conflicts of name.

To change the prefix, add a "namespace" to the configuration:

```
<!-- noSql database provider for the 3.3 connector framework -->
<noSQLConnectionProvider sslEnabled="false" sslInvalidHostNameAllowed="false">
  <namespace>myNamespace</namespace>
  <implementation>com.searchtechnologies.aspire:aspire-mongodb-provider</implementation>
  <servers>mongodb-host:27017</servers>
</noSQLConnecitonProvider>
```

Connect to a Multi-Node MongoDB Installation

To connect to a multi-node MongoDB installation, you just need to provide a comma-separated list of hostname:port of the MongoDB nodes in the cluster.

Example:

```
<!-- noSql database provider for the 3.3 connector framework -->
<noSQLConnectionProvider sslEnabled="false" sslInvalidHostNameAllowed="false">
  <implementation>com.searchtechnologies.aspire:aspire-mongodb-provider</implementation>
  <servers>mongodb-host1:27017,mongodb-host2:27017,mongodb-host3:27017,mongodb-host4:27017</servers>
</noSQLConnecitonProvider>
```

Using TLS/SSL

If you need to connect to a MongoDB configured to Use TLS/SSL, set the following attributes into the **noSQLConnectionProvider** tag:

Attribute	Value	Description
sslEnabled	true	Enables the ssl on the Aspire MongoDB client
sslInvalidHostNameAllowed	true/false	Disables the hostname verification from the SSL validation

1. For TLS/SSL, make sure the Certificate Authority (CA) that signed the server certificate (server.pem that MongoDB is using) is a trusted certificate. Otherwise, its trust chain can lead to one.

2. If you are using a self-signed Certificate Authority to sign your server certificate, add it into the java truststore.

3. To use a java truststore that you need the Certificate Authority certificate (.cert) and import it using the following command:

```
$ keytool -import -trustcacerts -alias slc -file <your-CA-certificate.cert> -keystore truststore.jks -
storepass <your-truststore-password> -noprompt
```

4. After importing it into a truststore, add it into the Aspire startup script, read Crawling via HTTPs for more instructions on how to add the truststore into the startup script.

Retries Settings

The Provider will automatically retry the operations in case they couldn't be completed due to connection errors.

The maximum retries to execute is configurable using the **"maxRetries"** option. By default (if nothing is provided), it will not retry operations at all.

```
  <!-- noSql database provider for the 3.3 connector framework -->
  <noSQLConnectionProvider sslEnabled="false" sslInvalidHostNameAllowed="false">
    <namespace>myNamespace</namespace>
    <implementation>com.searchtechnologies.aspire:aspire-mongodb-provider</implementation>
    <servers>mongodb-host:27017</servers>
        <maxRetries>5</maxRetries>
  </noSQLConnecitonProvider>
```

MongoDB Authentication

Aspire 3.3 supports authenticating to MongoDB using X.509 or SCRAM. Based on the requirement , it is necessary modify the **settings.xml** file.

SCRAM Authentication

Aspire 3.3 supports authenticating to MongoDB using SCRAM.

The Salted Challenge Response Authentication Mechanism (SCRAM) is a family of modern, password-based challenge–response authentication mechanisms providing authentication of a user to a server

1. To configure it, add the following to your **settings.xml** file:

settings.xml

```
<!-- noSql database provider for the 3.0 connector framework -->
  <noSQLConnectionProvider sslEnabled="true" sslInvalidHostNameAllowed="false">
    <implementation>com.searchtechnologies.aspire:aspire-mongodb-provider</implementation>
    <servers>mongodb-host:27017</servers>
    <authentication>
                  <scram>
                        <username>aspireUser</username>
                        <source>admin</source>
                        <password>encrypted:302B58140B6ED1FBEBDC33A9263EF742</password>
                </scram>
            </authentication>
  </noSQLConnecitonProvider>
```

MongoDB provider will verify the supplied user credentials against:

- Username -> User's name (must be created in Mongo)
- Password -> User's password, the system accepts passwords encrypted.
- Source -> Authentication database (usually "admin")

2. For the correct Aspire behavior check that the user selected to authenticate has the roles:

- **clusterAdmin**: Provides the greatest cluster-management access. This role combines the privileges granted by the clusterManager, clusterMonitor, and hostManager roles. Additionally, the role provides the dropDatabase action.
- **readWriteAnyDatabase:** Provides the same read and write privileges as readWrite on all databases except local and config. readWriteAnyDatabase also provides the listDatabases privilege action on the cluster.

3. Check the roles of a user using mongo.exe:

mongo.exe

```
> use admin
> db.getUser("aspireAdmin");
  {
    "_id_": "admin.myUserAdmin",
    "user": "myUserAdmin",
    "db": "admin",
    "roles": [
        {
          "role": "clusterAdmin",
          "db"": "admin"
        },
        {
          "role": "readWriteAnyDatabase",
          "db": "admin"
        }
      ]
  }
>
```

Enable Scram Authentication in MongoDB

1. Start MongoDB without access control
   $ mongod.exe --port 27017

2. Connect a mongo shell to the instance.
   $ mongo.exe --port 27017

3. Create the user administrator: The database where you create the user (in this example, admin) is the user's authentication database. For Aspire requirements, create the user with the roles: clusterAdmin and readWriteAnyDatabase.

mongo.exe

```
> use admin
> db.createUser(
  {
        user: "myUserAdmin",
        pwd: "abc123",
        roles: [
                { role: "clusterAdmin", db: "admin" },
                { role: "readWriteAnyDatabase", db: "admin" }
                     ]
                  }
        )
```

4. Re-start the MongoDB

a. Instance with access control.
   $mongod.exe --auth –port 27017
b. Re-start the MongoDB using configuration file. MongoDB configuration files use the YAML format. Adding security.authorization: enable
   $ mongod --config /etc/mongod.conf

Configuration Example

```
systemLog:
   destination: file
   path: "/var/log/mongodb/mongod.log"
   logAppend: true
storage:
```

X.509 Authentication

Aspire 3.3 only supports authenticating to MongoDB using X.509.

- The X.509 mechanism authenticates a user whose name is derived from the distinguished subject name of the X.509 certificate presented by the driver during SSL negotiation.
- This authentication method requires the use of SSL connections with certificate validation.

1. To configure it, add the following to your **settings.xml** file:

```
  <!-- noSql database provider for the 3.3 connector framework -->
  <noSQLConnectionProvider sslEnabled="true" sslInvalidHostNameAllowed="false">
    <implementation>com.searchtechnologies.aspire:aspire-mongodb-provider</implementation>
    <servers>mongodb-host:27017</servers>
    <x509username>CN=user,OU=OrgUnit,O=myOrg</x509username>
  </noSQLConnecitonProvider>
```

2. If you don't know what to use in the <x509username> field, execute the following command using the x509 client certificate:

```
$ openssl x509 -in client.pem -inform PEM -subject -nameopt RFC2253 | grep subject
subject= CN=aaguilar-lptp.search.local,OU=demouser,O=Search Technologies S.A.,ST=Limon,C=CR
```

3. For x509 authentication, import the client x509 certificate into a java keystore (so that Aspire can present it to the server for authentication).

- The truststore should already be set in the startup script for self signed certificates.

4. To import the x509 certificate (client.pem) into a java keystore, execute the following commands:

```
$ openssl pkcs12 -export -out client.pkcs12 -in client.pem
Enter Export Password: <your-password-here>

$ keytool -importkeystore -srckeystore client.pkcs12 -srcstoretype PKCS12 -destkeystore client.jks -
deststoretype JKS
Enter destination keystore password:
Re-enter new password: <your-password-here>
Enter source keystore password: <your-password-here>
Entry for alias 1 successfully imported.
Import command completed:  1 entries successfully imported, 0 entries failed or cancelled
```

5. After importing the client's certificate into a java keystore, include it in the Aspire startup script (aspire.bat).

```
-Djavax.net.ssl.keyStore=C:\pathToKeyStore\client.jks
-Djavax.net.ssl.keyStorePassword=password
```

Encrypt Sensitive Fields in MongoDB

If you want to be extra safe and encrypt the URLs, IDs, or any other metadata stored in MongoDB, you can do by specifying the names of the fields to encrypt:

```
<!-- noSql database provider for the 3.3 connector framework -->
<noSQLConnectionProvider sslEnabled="false" sslInvalidHostNameAllowed="false">
  <implementation>com.searchtechnologies.aspire:aspire-mongodb-provider</implementation>
  <servers>mongodb-host:27017</servers>
  <encryptFields>
    <field>_id</field> <!-- Encrypts all the IDs -->
    <field>url</field> <!-- Encrypts the url fields -->
    <field>fetchUrl</field>
    <field>parentId</field>
  </encryptFields>
</noSQLConnecitonProvider>
```