Direct MongoDB Access (3.5+)

This document describes how to connect to the MongoDB instance provided by **Namazu Elements**. It explains the configuration flow, how SSL is handled, and how an individual Element can establish a connection using the MongoDB driver of its choice.

This is availble in Namazu Elements 3.5 and above.

Using MongoDB Drivers

This guide only documents how to set up the connection to MongoDB. Making use of the drivers is beyond the scope of this document. To understand more about the drivers and their usages, please refer to the following from MongoDB's official Documentation:

- MongoDB Synchronous Driver
- MongoDB Reactive Streams Driver
- Hibernate ORM Extension for MongoDB
- Morhpia

Overview

Namazu Elements exposes a service named `MongoConfigurationService` that other Elements can use to retrieve the connection details for MongoDB. This service reads configuration from the application's configuration layer and returns an instance of `MongoConfiguration`.

Each Element then uses its preferred MongoDB driver and the retrieved configuration to establish a secure connection to the shared MongoDB instance.

MongoConfigurationService

`MongoConfigurationService` is the main entry point for obtaining your MongoDB configuration. Its responsibilities include:

- Reading application configuration settings that define how MongoDB should be accessed.
- Producing an initialized instance of `MongoConfiguration`.

Elements should request this service through the usual dependency injection or service lookup mechanisms used within Namazu Elements. Built in to Namazu Elements is a type

MongoConfiguration

The `MongoConfiguration` object returned by the service contains the information needed to connect to MongoDB.

Connection String

`MongoConfiguration` always includes a connection string. This connection string can include additional driver settings such as replica set information, authentication data, and flags for TLS.

Example:

```
mongodb+srv://example.mongodb.net/?retryWrites=true&w=majority
```

SSL Context

If SSL is configured for your application, `MongoConfiguration` will also include a `javax.net.ssl.SslContext`. This SSL context is prepared with the required trust manager and key manager settings.

This means the Element does not need to manually configure certificates, keystores, or truststores. All SSL requirements for talking to the MongoDB server are handled for you.

If SSL is not configured, the SSL context will be absent.

Connecting Using the MongoDB Java Driver

Elements can use any MongoDB driver they choose. This section demonstrates how to connect using the **MongoDB Java Driver**.

Below is a reference implementation illustrating how to:

- Extract the connection string.
- Build the SSL configuration.
- Apply both to the client settings.
- Create a `MongoClient` and verify the connection.

Example Code

```
public MongoClient connect() {

    final ElementRegistry registry = /* fetch registry */;


    final MongoConfigurationService mongoConfigurationService = registry
        .find("dev.getelements.elements.sdk.mongo")
        .findFirst()
        .get()
        .getServiceLocator()
        .getInstance(MongoConfigurationService.class);


    final MongoConfiguration conf =
        mongoConfigurationService.getMongoConfiguration();


    final ConnectionString connectionString = new ConnectionString(conf.connectionString());

    final SslSettings sslSettings = conf
        .findSslConfiguration()
        .map(MongoSslConfiguration::newSslContext)
        .map(sslContext -> SslSettings.builder()
            .enabled(true)
            .context(sslContext)
```

```
        )
        .orElseGet(() -> SslSettings.builder().applyConnectionString(connectionString))
        .build();


    final MongoClientSettings clientSettings = MongoClientSettings
        .builder()
        .applyConnectionString(connectionString)
        .applyToSslSettings(builder -> builder.applySettings(sslSettings))
        .build();


    return MongoClients.create(clientSettings);

}
```

How the Example Works

1. **Connection String**: Retrieved directly from the configuration.
2. **SSL Configuration**: If present, an SSL context is created and injected into `SslSettings`.
3. **Fallback Behavior**: If SSL is not configured, the SSL settings fall back to whatever is specified in the connection string.
4. **Client Settings**: Both the connection string and SSL settings are applied to `MongoClientSettings`.
5. **Client Lifecycle**: A `MongoClient` is created, used, and automatically closed via the try-with-resources block.
6. **Verification**: The cluster description is logged to confirm that the connection was successful.

Using Guice Injection

Elements that use Guice for dependency injection can simplify access to the `MongoConfigurationService`. Namazu Elements exposes its services using standard annotations.

package-info.java

```
@ElementPublic
@ElementDependency("dev.getelements.elements.sdk.mongo")
```

```
package dev.getelements.elements.sdk.mongo;


import dev.getelements.elements.sdk.annotation.ElementDependency;
import dev.getelements.elements.sdk.annotation.ElementPublic;
```

Injecting the Service in Your Game Code

You can inject the `MongoConfigurationService` directly into your Element or game service.

```
@Inject
public void setMongoConfigurationService(MongoConfigurationService service) {
    this.service = service;
}
```

Once injected, you can build your Mongo client as shown earlier by constructing the connection settings from the injected service.

Summary

Connecting to MongoDB through Namazu Elements involves three main steps:

1. Retrieve a `MongoConfiguration` instance using `MongoConfigurationService`.
2. Extract the connection string and optional SSL context.
3. Use your chosen MongoDB driver to configure and initialize a client.

Namazu Elements handles most of the complexity for you, especially around SSL, so Elements can focus on application logic while maintaining secure and consistent database access.