Exercise - Create a MongoDB DB app using the Azure Cosmos DB for MongoDB

- 35 minutes

This module requires a sandbox to complete. A **sandbox** gives you access to free resources. Your personal subscription will not be charged. The sandbox may only be used to complete training on Microsoft Learn. Use for any other reason is prohibited, and may result in permanent loss of access to the sandbox.

Microsoft provides this lab experience and related content for educational purposes. All presented information is owned by Microsoft and intended solely for learning about the covered products and services in this Microsoft Learn module.

Sign in to activate sandbox

Choose your development language
C#JavaNode.jsPython

It's time to programmatically check-out how to create our Azure Cosmos DB for MongoDB databases, collections and add some data.

This exercise can be completed using a Microsoft Learn *sandbox*, which provides a temporary Azure subscription. To activate the sandbox subscription, you must sign in using a Microsoft account. The sandbox subscription will be automatically deleted when you complete this module. After the sandbox has been activated, sign into the Azure portal using the credentials for your sandbox subscription. Ensure you're working in the **Microsoft Learn Sandbox** directory - indicated at the top right of the portal under your user ID. If not, select the user icon and switch directory.

Tip

If you prefer, you can use your own Azure subscription. To do so, **sign into the Azure portal using credentials for your subscription**. Ensure you are working in the directory containing your subscription - indicated at the top right under your user ID. If not, select the user icon and switch directory.

Create MongoDB app using *Node.js* Azure Cosmos DB for MongoDB

In this exercise, you'll create an Azure Cosmos DB for MongoDB account, a database, a collection and add a couple of documents to the collection. You'll notice that this code will be identical to how you would connect to any MongoDB database. You'll then create a collection using extension commands that allow you to define the throughput in Request Units/sec (RUs) for the collection.

Prepare your development environment

If you haven't already prepared the Azure Cosmos DB account and environment where you're working on this lab, follow these steps to do so. Otherwise, go to the **Add the code to create the databases, collection and document to the App.js file** section.

1. In Azure Cloud Shell, copy and paste the following commands.

   BashCopy
   ```
   git clone https://github.com/MicrosoftLearning/mslearn-cosmosdb.git
   cd ~/mslearn-cosmosdb/api-for-mongodb/01-create-mongodb-objects/node/

   # Update Azure Cloud Shell node to Version 14.0.0, since the MongoDB
   driver requires ver 10+
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh
   | bash
   source ~/.nvm/nvm.sh
   nvm install 14.0.0
   npm install -g mongodb
   npm link mongodb
   # Check if the node version is now v14.0.0
   node --version

   # Create an Azure Cosmos DB for MongoDB account
   bash ../init.sh
   ```

   Tip

   If you are not using the sandbox for the lab, and you want to specify the location where you would like to create your database and storage objects, add a ***-l LOCATIONNAME*** parameter to the *init.sh* call. Additionally, if you would like to specify a resource group, add a ***-r YOURRRESOURCEGROUPNAMEHERE*** parameter to the *init.sh* call.

   Note

This bash script will create the Azure Cosmos DB for MongoDB account. *It can take 5-10 minutes to create this account* so it might be a good time to get a cup of coffee or tea.

Tip

If you come back and your cloud shell has reset, run the following commands in the cloud shell to use Node version 14, otherwise the code in the next section will fail.

   a. source ~/.nvm/nvm.sh
   b. nvm install 14.0.0
   c. npm link mongodb

2. When the bash *init.sh* file completes running, copy somewhere the ***Connection String***, ***Cosmos DB Account name*** and ***Resource Group name*** returned, we'll need them in the next section. You can also review the JSON returned by the account creation script that is located before the connection string. If you look somewhere in the middle of the JSON, you should see the property **"kind": "MongoDB"**.

Note

The ***Connection String***, ***Cosmos DB Account name*** and ***Resource Group name*** can also be found using the Azure portal.

Add the code to create the databases, collection and document to the App.js file

It's now time to add our JavaScript code to create a Database, a Collection and add a document to the collection.

1. In not already opened, open the Azure Cloud Shell.
2. Run the following command to open the code editor.

BashCopy
```
cd ~/mslearn-cosmosdb/api-for-mongodb/01-create-mongodb-objects/node/
code App.js
```

3. Copy the following code to the App.js file. *Don't forget that you'll need to replace the url value for the connection string copied in step 2 of the previous section*. This connection string should look like

mongodb://learn-account-cosmos-92903170:XvrarRd8LnqWNZiq3ahHXngbZoVRxVO192WahrcdsmHVivBGbRqnHx2cq0oMGnc0DUPAWpyGu7kt7APVH4nqXg==@learn-account-cosmos-92903170.mongo.cosmos.azure.com:10255/?ssl=true&replicaSet=globaldb&retrywrites=false&maxIdleTimeMS=120000&appName=@learn-account-cosmos-92903170@.

This part of the code uses the MongoDB drivers and uses the connection string to Azure Cosmos DB like you would normally use a connection string to any MongoDB server. The code then defines and opens the connection to the Azure Cosmos DB account.

JavaScriptCopy
```
// Uses the MongoDB driver
const {MongoClient} = require("mongodb");

async function main() {

  // Replace below "YourAzureCosmosDBAccount" with the name of your Azure Cosmos DB
  // account name and "YourAzureCosmosDBAccountKEY" with the Azure Cosmos DB account key.
  // Or replace it with the connection string if you have it.
  var url = "mongodb://YourAzureCosmosDBAccount:YourAzureCosmosDBAccountKEY@YourAzureCosmosDBAccount.mongo.cosmos.azure.com:10255/?ssl=true&retrywrites=false&replicaSet=globaldb&maxIdleTimeMS=120000&appName=@YourAzureCosmosDBAccount@";

  // define the connection using the MongoClient method ane the url above
  var mongoClient = new MongoClient(url, function(err,client)
    {
      if (err)
      {
        console.log("error connecting")
      }
    }
  );

  // open the connection
  await mongoClient.connect();
```

4. The next step connects to the **products** database. Note that if this database doesn't exist it will create it only if also creates a collection in the same connection or by using extension commands. Add the following to the script in the editor.

JavaScriptCopy
```
  // connect to the database "products"
  var ProductDatabase = mongoClient.db("products");
```

5. Next, we'll connect to the **documents** collection if it already exists, and then adds one document to the collection. Note that if the collection doesn't exist this code will only create the collection if it also performs an operation on that collection in the same connection (for example, like add a document to the collection) or by using extension commands. Add the following to the script in the editor.

JavaScriptCopy
```
  // create a collection "documents" and add one document for "bread"
  var collection = ProductDatabase.collection('documents');
  var insertResult = await collection.insertOne({ ProductId: 1, name:
"bread" });
```

6. Lets now search for the document we just inserted and display it to the shell. Add the following to the script in the editor.

JavaScriptCopy
```
  // return data where ProductId = 1
  const findProduct = await collection.find({ProductId: 1});
  await findProduct.forEach(console.log);
```

7. Finally let's close the connection and call the *main* function to run it. Add the following to the script in the editor.

JavaScriptCopy
```
  // close the connection
  mongoClient.close();
}

main();
```

8. The script should look like this:

JavaScriptCopy
```
// Uses the MongoDB driver
const {MongoClient} = require("mongodb");

async function main() {
  // Replace below "YourAzureCosmosDBAccount" with the name of your Azure
Cosmos DB
  // account name and "YourAzureCosmosDBAccountKEY" with the Azure Cosmos
DB account key.
  // Or replace it with the connection string if you have it.
  var url =
"mongodb://YourAzureCosmosDBAccount:YourAzureCosmosDBAccountKEY@YourAzure
CosmosDBAccount.mongo.cosmos.azure.com:10255/?ssl=true&retrywrites=false&
replicaSet=globaldb&maxIdleTimeMS=120000&appName=@YourAzureCosmosDBAccoun
t@";

  // define the connection using the MongoClient method ane the url above
  var mongoClient = new MongoClient(url, function(err,client)
    {
      if (err)
      {
        console.log("error connecting")
      }
    }
  );

  // open the connection
  await mongoClient.connect();

  // connect to the database "products"
  var ProductDatabase = mongoClient.db("products");

  // create a collection "documents" and add one document for "bread"
  var collection = ProductDatabase.collection('documents');
  var insertResult = await collection.insertOne({ ProductId: 1, name:
"bread" });

  // return data where ProductId = 1
  const findProduct = await collection.find({ProductId: 1});
  await findProduct.forEach(console.log);

  // close the connection
  mongoClient.close();
}

main();
```

9. Let's go ahead and save the JavaScript program. Select on the Upper right hand corner of the code editor and select **Save** (or Ctrl+S). Now select **Close Editor** (or Ctrl+Q) to go back to the Shell.
10. Let's now run the JavaScript App with the following command.

BashCopy
```
node App.js
```

11. This should return a similar result to the one below. This means that we created the database, collection and added a document to it.

JSONCopy
```
{
  _id: new ObjectId("62aed08663c0fd62d30240db"),
  ProductId: 1,
  name: 'bread'
}
```

As you should have noticed, this code is the same code you would run to create a database, collection and document on a MongoDB database. So programming for Azure Cosmos DB for MongoDB should be transparent to you if you're already familiar with creating apps that connect to MongoDB.

Using extension commands to manage data stored in Azure Cosmos DB's API for MongoDB

While the code above, except for the connection string, would be identical between connecting to a MongoDB Server then connection to our Azure Cosmos DB for MongoDB account, this might not take advantage of Azure Cosmos DB features. What this means is using the default driver methods to create our collections, will also use the default Azure Cosmos DB Account parameters to create those collections. So we won't be able to define creation parameters like our throughput, sharding key or autoscaling settings using those methods.

By using the Azure Cosmos DB's API for MongoDB, you can enjoy the benefits of Cosmos DB such as global distribution, automatic sharding, high availability, latency guarantees, automatic, encryption at rest, backups, and many more, while preserving your investments in your MongoDB app. You can communicate with the Azure Cosmos DB's API for MongoDB by using any of the open-source MongoDB client drivers. The Azure Cosmos DB's API for MongoDB enables the use of existing client drivers by adhering to the MongoDB wire protocol.

Let's create some code that will allow us to create a collection and define its sharding key and throughput.

1. In not already opened, open the Azure Cloud Shell.
2. Run the following command to open the code editor.

BashCopy
```
cd ~/mslearn-cosmosdb/api-for-mongodb/01-create-mongodb-objects/node
code App.js
```

3. Copy the following code and *replace the existing content* from the app.cs file. *Don't forget that you'll need to replace the uri value for the connection string copied in step 2 of the previous section*. This part of the code uses the MongoDB drivers and uses the connection string to Azure Cosmos DB like you would normally use a connection string to any MongoDB server. The code then defines and opens the connection to the Azure Cosmos DB account.

JavaScriptCopy
```
// Uses the MongoDB driver
const {MongoClient} = require("mongodb");

async function main() {

  // Replace below "YourAzureCosmosDBAccount" with the name of your Azure
Cosmos DB
  // account name and "YourAzureCosmosDBAccountKEY" with the Azure Cosmos
DB account key.
  // Or replace it with the connection string if you have it.
  var url =
"mongodb://YourAzureCosmosDBAccount:YourAzureCosmosDBAccountKEY@YourAzure
CosmosDBAccount.mongo.cosmos.azure.com:10255/?ssl=true&retrywrites=false&
replicaSet=globaldb&maxIdleTimeMS=120000&appName=@YourAzureCosmosDBAccoun
t@";

  // define the connection using the MongoClient method ane the url above
  var mongoClient = new MongoClient(url, function(err,client)
    {
      if (err)
      {
        console.log("error connecting")
      }
    }
  );

  // open the connection
  await mongoClient.connect();
```

4. The next step connects to the **employees** database. Note that if this database doesn't exist it will create it only if also creates a collection in the same connection or by using extension commands. Add the following to the script in the editor.

JavaScriptCopy
```
  // connect to the database "HumanResources"
  var EmployeeDatabase = mongoClient.db("HumanResources");
```

5. So far it looks pretty much like the code in the previous section. In this step, we'll now take advantage of the extension commands and create a custom action. This action will allow us to define the throughput and the sharding key of the collection, which will in turn give Azure Cosmos DB the parameters to use when creating the collection. Add the following to the script in the editor.

JavaScriptCopy
```
  // create the Employee collection with a throughput of 1000 RUs and
with EmployeeId as the sharding key
  var result = EmployeeDatabase.command({customAction:
"CreateCollection", collection: "Employee", offerThroughput: 1000,
shardKey: "EmployeeId"});
```

6. The rest will be pretty identical to the previous example, we will connect to the collection, insert some rows, finally query and output a row back. Add the following to the script in the editor.

JavaScriptCopy
```
  // Connect to the collection "Employee" and add two documents for
"Marcos" and "Tam"
  var collection = EmployeeDatabase.collection('Employee');

  var insertResult = await collection.insertOne({EmployeeId: 1, email:
"Marcos@fabrikam.com", name: "Marcos"});
  insertResult = await collection.insertOne({EmployeeId: 2, email:
"Tam@fabrikam.com", name: "Tam"});

  // return data where ProductId = 1
  const findProduct = await collection.find({EmployeeId: 1});
  await findProduct.forEach(console.log);

  // close the connection
  mongoClient.close();
}

main();
```

7. The script should look like this:

JavaScriptCopy
```
// Uses the MongoDB driver
const {MongoClient} = require("mongodb");

async function main() {

  // Replace below "YourAzureCosmosDBAccount" with the name of your Azure
Cosmos DB
```

```
  // account name and "YourAzureCosmosDBAccountKEY" with the Azure Cosmos
DB account key.
  // Or replace it with the connection string if you have it.
  var url =
"mongodb://YourAzureCosmosDBAccount:YourAzureCosmosDBAccountKEY@YourAzure
CosmosDBAccount.mongo.cosmos.azure.com:10255/?ssl=true&retrywrites=false&
replicaSet=globaldb&maxIdleTimeMS=120000&appName=@YourAzureCosmosDBAccoun
t@";

  // define the connection using the MongoClient method ane the url above
  var mongoClient = new MongoClient(url, function(err,client)
    {
      if (err)
      {
        console.log("error connecting")
      }
    }
  );

  // open the connection
  await mongoClient.connect();

  // connect to the database "HumanResources"
  var EmployeeDatabase = mongoClient.db("HumanResources");

  // create the Employee collection with a throughput of 1000 RUs and
with EmployeeId as the sharding key
  var result = EmployeeDatabase.command({customAction:
"CreateCollection", collection: "Employee", offerThroughput: 1000,
shardKey: "EmployeeId"});

  // Connect to the collection "Employee" and add two documents for
"Marcos" and "Tam"
  var collection = EmployeeDatabase.collection('Employee');

  var insertResult = await collection.insertOne({EmployeeId: 1, email:
"Marcos@fabrikam.com", name: "Marcos"});
  insertResult = await collection.insertOne({EmployeeId: 2, email:
"Tam@fabrikam.com", name: "Tam"});

  // return data where ProductId = 1
  const findProduct = await collection.find({EmployeeId: 1});
  await findProduct.forEach(console.log);

  // close the connection
  mongoClient.close();
}

main();
```

8. Let's go ahead and save the Node.js program. Select on the Upper right hand corner of the code editor and select **Save** (or Ctrl+S). Now select **Close Editor** (or Ctrl+Q) to go back to the Shell.

9. Let's now run the Node.js App with the following command.

   BashCopy
   ```
   node App.js
   ```

10. This should return a similar result to the one below. This means that we created the database, collection and added a document to it.

    JSONCopy
    ```
    {
      _id: new ObjectId("62aed08663c0fd62d30240db"),
      EmployeeId: 1,
      email: 'Marcos@fabrikam.com'
      name: 'Marcos'
    }
    ```

11. However this last result set only confirmed that we indeed created a database, collection and documents, but what about our shard key and throughput, did they really change? On the Cloud Shell let's run the following commands to verify our changes took effect.
    a. Let's verify that our Shard key changed to ***EmployeeId*** (the default is *id*). *Don't forget to change the **resource group name** and **account name** for the names we saved at the beginning of this lab.*

       BashCopy
       ```
       az cosmosdb mongodb collection show --name Employee --database-
       name HumanResources --resource-group learn-20c8df29-1419-49f3-
       84bb-6613f052b5ae --account-name learn-account-cosmos-845083734
       ```

       The result should include the property **"shardKey": {"EmployeeId": "Hash"}**.

    b. Let's verify that our Throughput changed to ***1000*** (the default is *400*). *Don't forget to change the **resource group name** and **account name** for the names we saved at the beginning of this lab.*

       BashCopy
       ```
       az cosmosdb mongodb collection throughput show --name Employee -
       -database-name HumanResources --resource-group learn-20c8df29-
       1419-49f3-84bb-6613f052b5ae --account-name learn-account-cosmos-
       845083734
       ```

       The result should include the property **"throughput": 1000**.

This code illustrated the power of using extended commands in our code, which allows us to define the Azure Cosmos DB creation parameters. This allows us to take advantage of controlling how our collections will be created and processed by Azure Cosmos DB.

Once you've completed this exercise, continue to the knowledge check questions for this module.

Next unit: Knowledge check

Continue