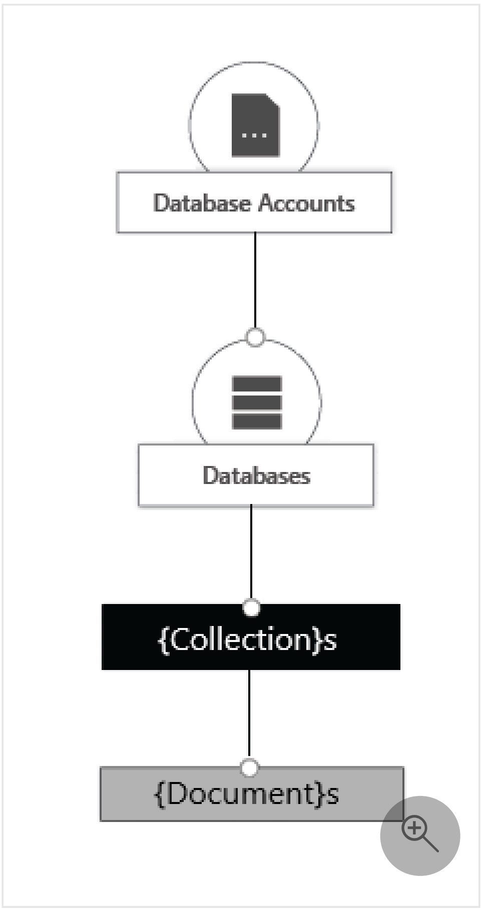
100 XP

# Create an account, database and collection using Azure Cosmos DB for MongoDB

8 minutes

Azure Cosmos DB is a fully managed platform-as-a-service (PaaS). To use this service, we'll need to first create an Azure Cosmos DB account under our subscription. Once our account has been created, we can then add database, collections and documents within it.



We'll take a look at a couple of ways to create the different Azure Cosmos DB for MongoDB model elements.

## Create an account, database and container for the Azure Cosmos DB for MongoDB using the Azure portal

One way of creating our Azure Cosmos DB account and its elements, is using the Azure portal. In this example, we'll use the Azure portal to create a simple Azure Cosmos DB account using the Azure Cosmos DB for MongoDB. We'll then add a database and a collection. For the moment, we won't worry about filling out more advanced settings that we'll cover in more detail in later modules. For this example, we'll just visit the Basics tab. Let's create our account and its elements.

## Create an account for the Azure Cosmos DB for MongoDB

1. In the Azure portal, Select + **Create a Resource**, select **Azure Cosmos DB** and **Create**.

2. Select **Azure Cosmos DB for MongoDB**.

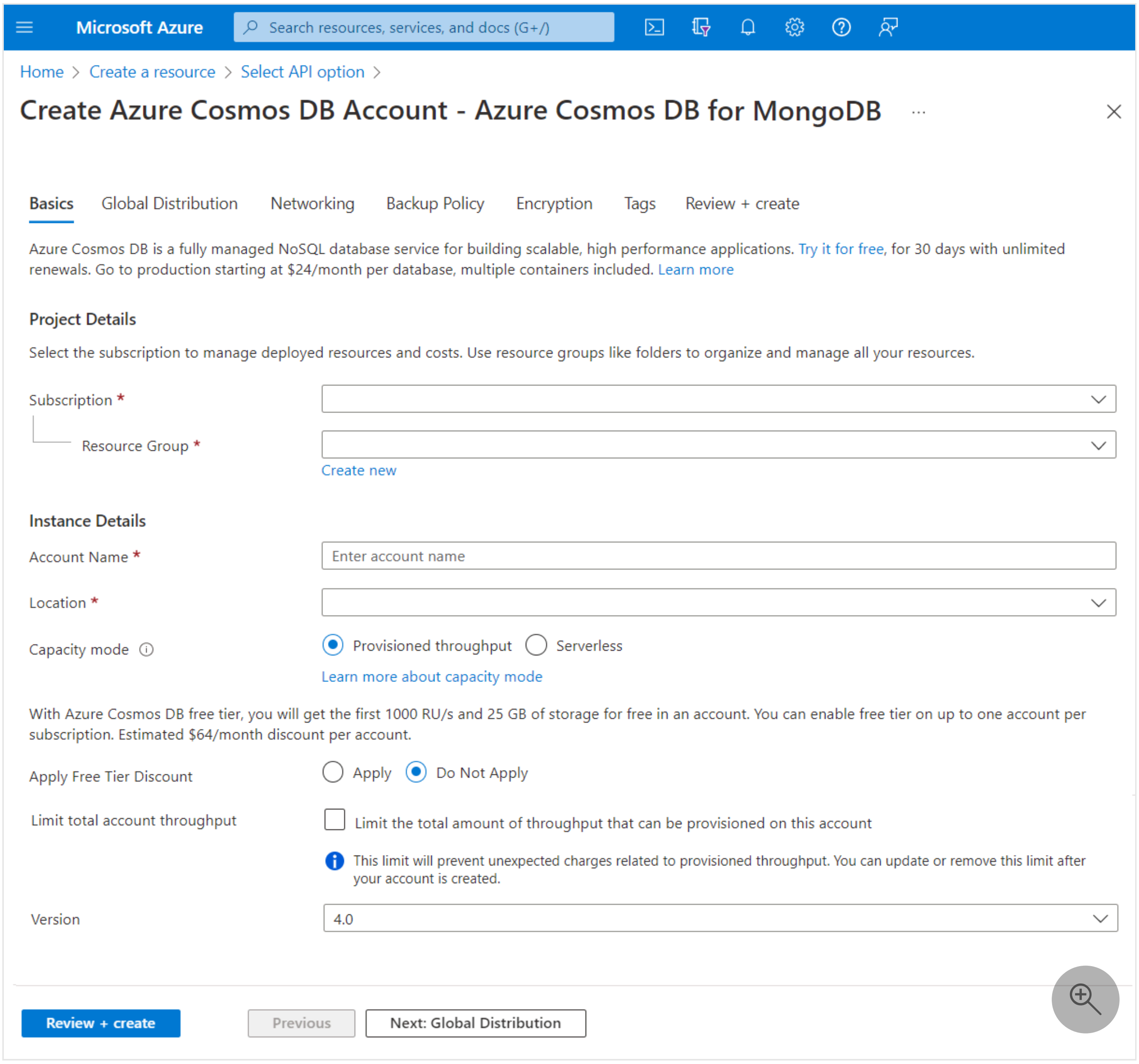
3. Input the following parameters.

    - **Subscription** - Your current Azure subscription.
    - **Resource Group** - A new or existing Azure Resource Group to create the Azure Cosmos DB account on.
    - **Account Name** - A unique name for your Azure Cosmos DB account. This name must be unique across the Azure. Your account URI will be *mongo.cosmos.azure.com* appended to your account name.
    - **Location** - The geographical location that will host your Azure Cosmos DB account. You should typically select a location close to your users or applications.
    - **Capacity mode** - As we discussed in a previous unit, you can select for your account to use *Provisioned throughput* or *Serverless*. Select **Provisioned throughput** for this example.
    - **Apply Free Tier Discount** - Select *Don't Apply* for this example.
    - **Limit total account throughput** - Leave *unchecked* for this example.
    - **Version** - Select *4.0*.

    > ⓘ **Note**
    >
    > To take better advantage of the supported features, we recommend that you use versions 3.6+ whenever possible.

4. Select **Review + Create**, and on a successful validation, select **Create**.

> ⓘ **Note**
>
> It can take several minutes to create your Azure Cosmos DB account.


Microsoft Azure
Search resources, services, and docs (G+/)
Home > Create a resource > Select API option >

Create Azure Cosmos DB Account - Azure Cosmos DB for MongoDB

Azure Cosmos DB is a fully managed NoSQL database service for building scalable, high performance applications. Try it for free, for 30 days with unlimited renewals. Go to production starting at $24/month per database, multiple containers included. Learn more

Project Details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *

Resource Group *

Create new

Instance Details

Account Name * Enter account name

Location *

Capacity mode: [✓ Provisioned throughput] Serverless

Learn more about capacity mode

With Azure Cosmos DB free tier, you will get the first 1000 RU/s and 25 GB of storage for free in an account. You can enable free tier on up to one account per subscription. Estimated $64/month discount per account.

Apply Free Tier Discount: Apply [✓ Do Not Apply]

Limit total account throughput: ☐ Limit the total amount of throughput that can be provisioned on this account

This limit will prevent unexpected charges related to provisioned throughput. You can update or remove this limit after your account is created.

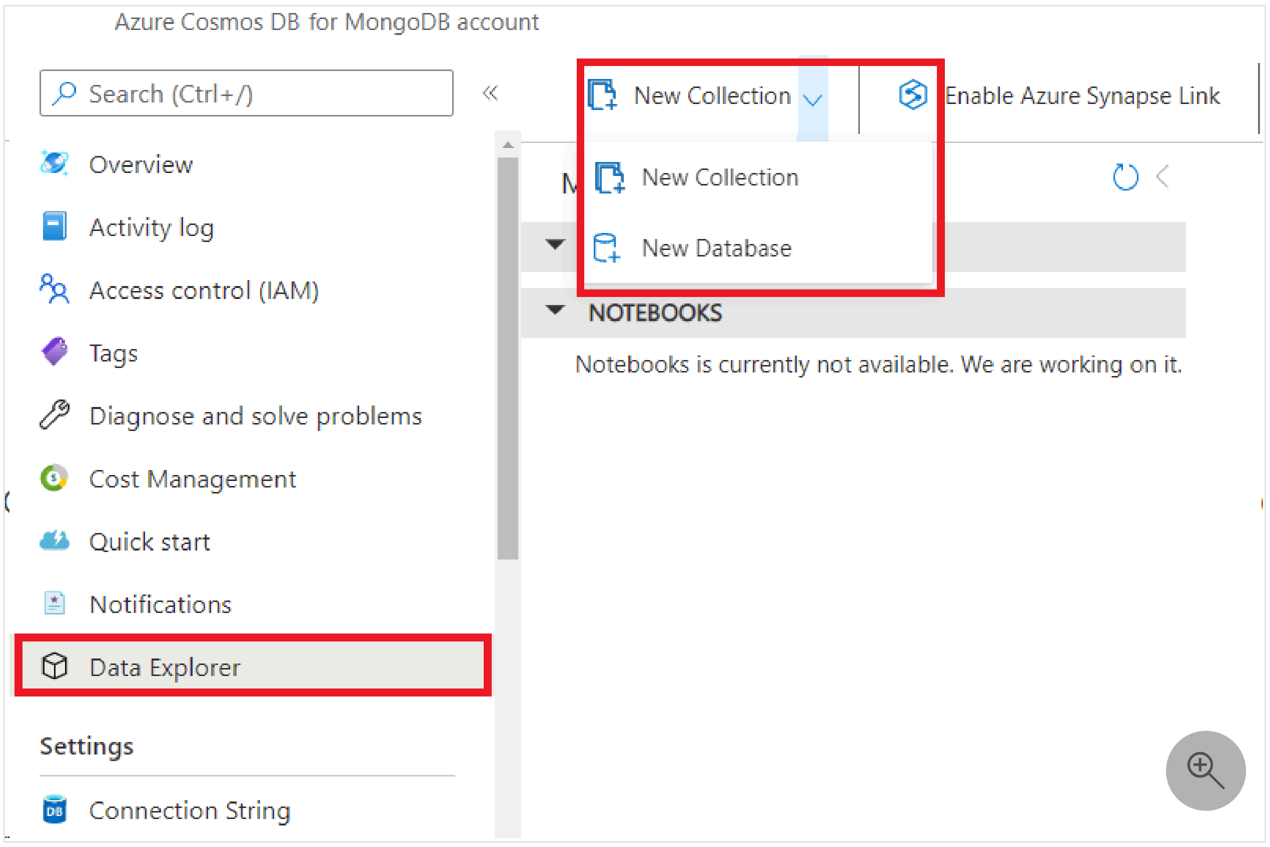
Version: 4.0

## Create a database and container for the Azure Cosmos DB for MongoDB

Creating a database and container in the Azure portal is straight forward, let's first create a database.

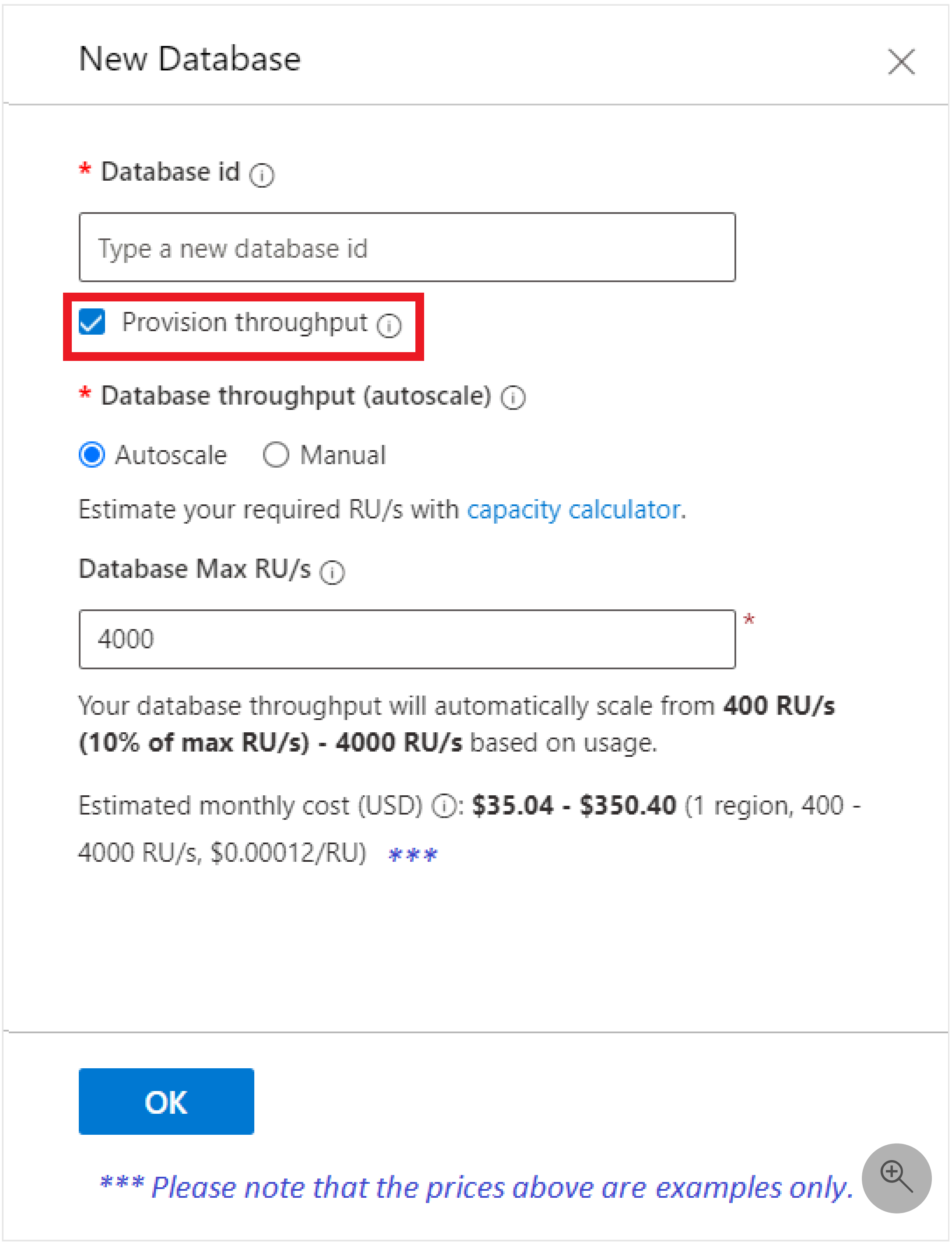
1. On the *Azure Cosmos DB for MongoDB account* left-hand menu, select **Data Explorer**.

2. Select the pulldown to the right of the *New Collection* icon and select **New Database**.



3. Give your database a new name under the **Database id** textbox.

4. While we could define the **Provision throughput** in the *New Database* dialog, in most cases you would select your provisioned throughput at the container level. Let's *uncheck* the *Provision throughput* checkbox and select **Ok** for this example.


New Database

* Database id

Type a new database id

[✓] Provision throughput

* Database throughput (autoscale)

[✓ Autoscale] Manual

Estimate your required RU/s with capacity calculator.

Database Max RU/s

4000

Your database throughput will automatically scale from **400 RU/s (10% of max RU/s) - 4000 RU/s** based on usage.

Estimated monthly cost (USD): **$35.04 - $350.40** (1 region, 400 - 4000 RU/s, $0.00012/RU) *****

**** Please note that the prices above are examples only.*

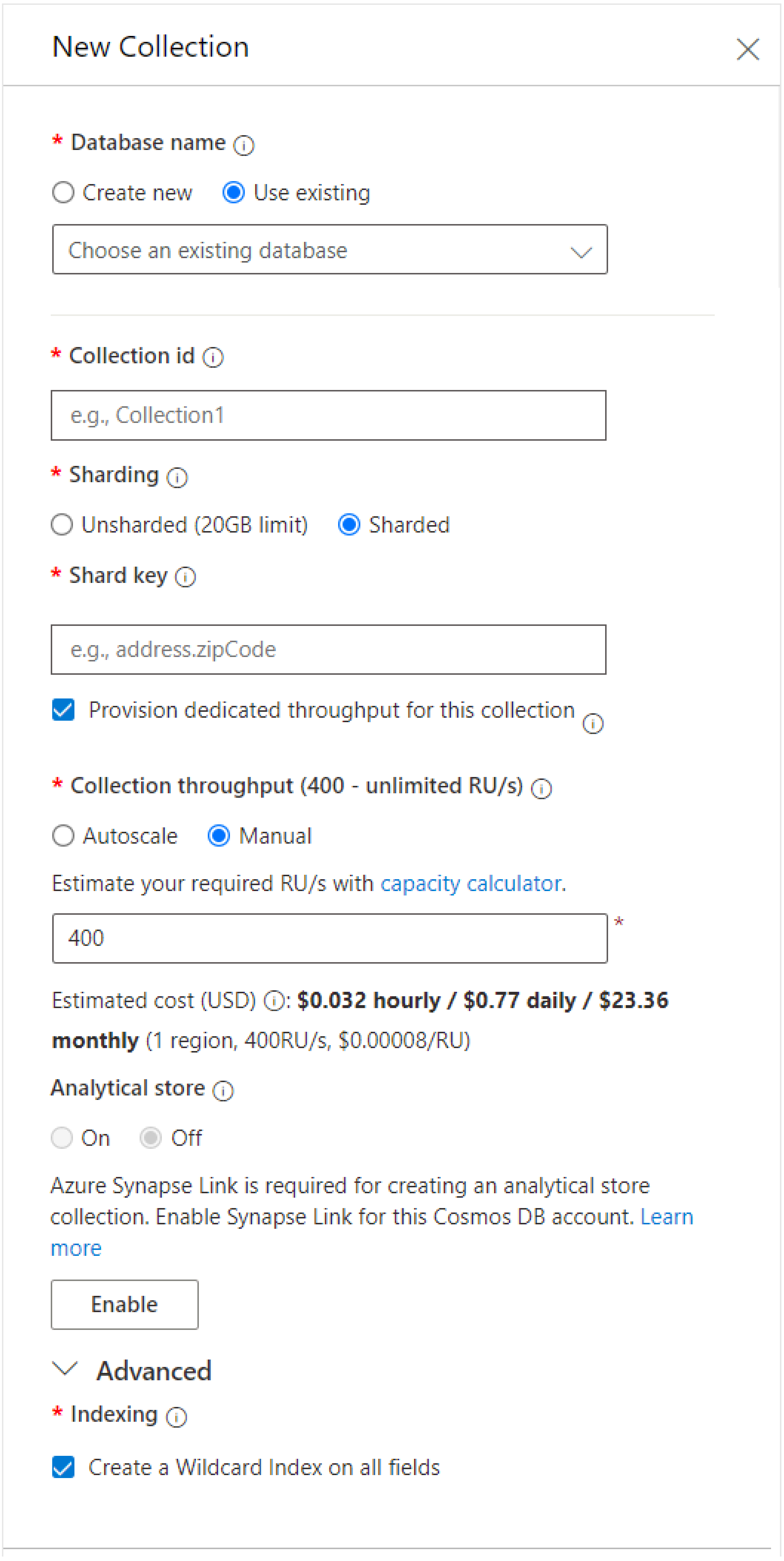
We should now notice our new database under the *MONGO API* section. It's time to create our new collection.

5. Select the **New Collection** icon.

6. Input the following parameters under the *New Collection* dialog.

   - **Database name** - You'll notice that you have two options here, *Create new* or *Use existing*. This option allows you to create a new database at the time you're creating the new collection. Since we already created a new database in our previous steps, select *Use existing* and pick the database name from the pulldown options.
   - **Collection id** - This parameter will be the name you'll give your collection.

- **Sharding** - We usually would want to select **Sharded**. This option allows Azure Cosmos DB to create containers that are sharded across multiple tenants based on the *Shard key*. With large containers, Azure Cosmos DB spreads your tenants across multiple physical nodes to achieve a high degree of scale. We'll discuss sharding in more detail under the *Design* module. Select ***Sharded***.
    - **Shard key** - Selecting *Sharded* will require you to add a *Shard key*. This is the partition key that defines your partitioning strategy, for example in and IOT collection it could be */deviceid* or maybe */region* depending on the partitioning strategy you choose. We'll discuss partitioning strategy in more detail under the *Design* module.
    - **Provision dedicated throughput for this collection** checkbox - Usually you'll want to provision the throughput of your collection either as *Autoscale* or *Manual*. This setting allows you to have better control of the cost of your individual collection. We'll discuss throughput in more detail in the *Design* module. For now, check the checkbox and select **Autoscale**.
    - **Collection throughput** or **Collection Max RU/s** - Depending if you selected *Manual* or *Autoscale* you'll need to input the *Collection throughput* or *Collection Max RU/s* respectively, or in other words the throughput mode. The main difference is that in *Manual* mode will charge you the amount the RU/s chosen regardless if you use them or not, and *Autoscale* will only charge what you use up to the Max RU/s you selected. In either case, your collection will start throttling once its throughput reaches the selected value. Leave the predefined value for this example.
    - **Analytical Store** - The Azure Cosmos DB analytical store is beyond the scope of these lessons. Review the What is Azure Cosmos DB analytical store article for more information on that topic.
    - **Advanced/Indexing** - We'll discuss indexes in more detail under the *Design* module. Leave the checkbox checked.



New Collection

* Database name ⓘ

○ Create new [✓ ● Use existing]

Choose an existing database

* Collection id ⓘ

e.g., Collection1

* Sharding ⓘ

○ Unsharded (20GB limit) [✓ ● Sharded]

* Shard key ⓘ

e.g., address.zipCode

☑ Provision dedicated throughput for this collection ⓘ

* Collection throughput (400 - unlimited RU/s) ⓘ

○ Autoscale [✓ ● Manual]

Estimate your required RU/s with capacity calculator.

400 *

Estimated cost (USD) ⓘ: **$0.032 hourly / $0.77 daily / $23.36 monthly** (1 region, 400RU/s, $0.00008/RU)

Analytical store ⓘ

○ On [✓ ● Off]

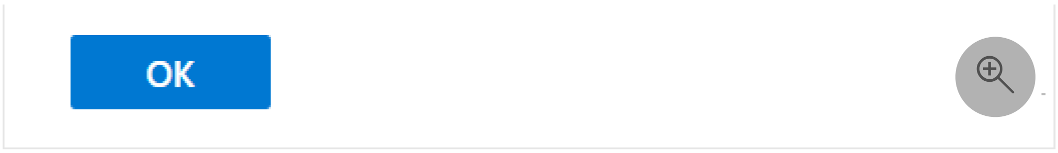
Azure Synapse Link is required for creating an analytical store collection. Enable Synapse Link for this Cosmos DB account. Learn more

Enable

Advanced

* Indexing ⓘ

☑ Create a Wildcard Index on all fields



We should now have a database and one collection to connect to. In the next unit, we'll go into more detail on connecting to an Azure Cosmos DB for MongoDB account. Before that, let's review another way of creating our Azure Cosmos DB for MongoDB databases and collections.

## Create or connect to a database and collection for the Azure Cosmos DB for MongoDB

You should be able to create or connect to your Azure Cosmos DB for MongoDB account using the Java, Python, Node.js, .NET or other programming language with a MongoDB driver. Let's introduce some functions you would use to create these elements using some of the different languages. At the beginning of this module, we stated that *the developers can keep on using MongoDB drivers, SDKs and tools they're familiar with to create apps and connect to Azure Cosmos DB*. We'll leverage on those drivers and their properties and methods to program our access and operations against our Azure Cosmos DB for MongoDB account.

### Create or connect to a database for the Azure Cosmos DB for MongoDB

You can use your favorite development tool to create your Azure Cosmos DB for MongoDB application. We'll leverage on the MongoDB driver for each respective programming language to create our databases and collections. Let's review the code to connect to the Azure Cosmos DB for MongoDB accounts and to connect to the *products* database.

***Node.js***

JavaScript

```javascript
const {MongoClient} = require("mongodb");

async function main() {

  // We will discuss connection string in more detail in the next unit of this module.
  // Remember to replace below "YourAzureCosmosDBAccount" with the name of your Azure Cosmos DB
  // account name and "YourAzureCosmosDBAccountKEY" with the Azure Cosmos DB account key.
  var url = "mongodb://YourAzureCosmosDBAccount:YourAzureCosmosDBAccountKEY@YourAzure-
CosmosDBAccount.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();

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

  // Add code to connect to a collection and add an entry here

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

main();
```

It's that simple, once we connected using the driver, we either create a new database, or pointed to an existing one with the *GetDatabase* or similar methods depending on the language. Our application can now use the **ProductDatabase** variable to reference to the desired database. Creating or connecting to a collection will be as simple as it was to create a new database.

## Create a collection for the Azure Cosmos DB for MongoDB

To create or access an existing collection, we'll use a get collection method or reference depending on the programming language. Let's add some code to the previous example to create/connect to a collection and add one entry on that collection.

***Node.js***

JavaScript

```
          // Add code to connect to a collection and add and find an entry
here
          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);
```

In the next unit, we'll see how exactly did we created our connection string.

## Use MongoDB extension commands to manage data stored in Azure Cosmos DB’s API for MongoDB

As we discussed earlier, Azure Cosmos DB for MongoDB gives us the ability to use the same drivers and code we used to access and create our objects in a MongoDB server for our Azure Cosmos DB account. However, using that code to create our databases and collections will use the default Azure Cosmos DB creation parameters. To take advantage of Azure Cosmos DB features, we'll need to be able to control our database and collection creation parameters like throughput, autoscaling, assigning shard keys, and defining indexes. Azure Cosmos DB for MongoDB gives us this ability by using extended commands to define those parameters. These commands allow us to code more precise instructions on how to create or modify our databases and collections specifically for Azure Cosmos DB.

Azure Cosmos DB for MongoDB provides extension commands for the following request types:

- Create database
- Update database
- Get database
- Create collection
- Update collection
- Get collection

The MongoDB drivers provide a function to run a command against a database, we'll use this function to send our extended commands to Azure Cosmos DB. Let's take a look at the code to create an IOT Device collection with a throughput of 2000 RUs and a shard key of DeviceId.

***Node.js***

JavaScript

```
    // create the Devices collection with a throughput of 2000 RUs and with
DeviceId as the sharding key
```

```
    var result = IOTDatabase.command({customAction: "CreateCollection",
collection: "Devices", offerThroughput: 2000, shardKey: "DeviceId"});
```

In a similar fashion we can modify a collection or create or modify a database. Review the Use MongoDB extension commands to manage data stored in Azure Cosmos DB's API for MongoDB article for more information.

## Next unit: Connect and query with an Azure Cosmos DB for MongoDB

Continue >